**Week 4 - Chapter 4 - L\_Sniff-Spoof-1**  **[ Packet Sniffing and Spoofing ]**

**A close-up of a computer

Description automatically generatedA close-up of a sign

Description automatically generated**

|  |  |
| --- | --- |
| **4 -> Precisely, sniffer can only sniff the packets that arrive at the sniffing device. It is impossible to sniff a device far from you.**  **5**   * **The normal sockets can only receive packets that are intended for it** * **It goes through TCP/IP stack and finally deliver the message to the application.** | **A screenshot of a computer  Description automatically generated** |

**A screenshot of a computer

Description automatically generatedA close-up of a computer error

Description automatically generated**

|  |  |
| --- | --- |
| **6 ->** Usually, we want to make use of the packet headers in order to modify them and create some attacks. Then, the normal socket does not help. The raw socket can capture the packet of link layer. It is similar to the normal socket with different parameters.  **9 ->** To construct packets using python Scapy package, we can run the command line script. After sudo python3,  >> From scapy.all import \*  >> ls(IP()) # this will see the fields of IP header. To construct an ip header, just assign values to the fields. If you do not give values to some fields, the system will do this for you.  Important: If chksum=None in IP header or ICMP, the system will calculate the chksum for you when sending the packet out. This is useful if you want to modify some captured packets. In this case, you do not set chksum=None, it might have checksum error when sending out (you can verify this using Wireshark).  # ICMP, ARP, UDP packets can be constructed similarly. | **A screen shot of a computer  Description automatically generated**  **A close-up of a sign  Description automatically generated** |

**A close-up of a computer program

Description automatically generatedA close up of a piece of paper

Description automatically generated**

|  |  |
| --- | --- |
| **11 ->** In this program, pkt is an array of length 5. So pkt[2] is the third captured packet. Pkt[2].show2() is to give the details of this packet, including the linker layer information.  iface is your interface that the sniffer is sniffing on. If you do not specify, it will sniff enp0s3 interface in our VM. In our docker-compose file, we create subnet 10.9.0.0/24. If you want to sniff this subnet, the iface is br-xxx. Check ifconfig in your attacker VM.  **12 ->** prn is the assigned with the name of the callback function. This is the function you can define how to process the captured packet.  **13 ->**   * Check BPF.pdf for more description. For the operator, **&&** is the same as **and**; **!** is the same as **not**;  **or** is the same as **||**. * You need to know the key words: **host, net, port, src, dst** * When we say ip address a.b.c.d, we say **host a.b.c.d** for the expression. * Key words can be used in combination. For example, **src host 10.0.2.1** | **A screenshot of a computer program  Description automatically generated**    **A screen shot of a computer  Description automatically generated** |

**A screen shot of a computer program

Description automatically generated A screen shot of a computer

Description automatically generated**

|  |  |
| --- | --- |
|  | **16 ->** In the assignment, you will be asked to send frame packet using sendp() and you need to set iface. |
| **A screenshot of a computer program  Description automatically generated** | **17 ->** **netcat** is useful command. In the nc command, l for listening, n for numeric, u for udp, v for verbose, p for port number. nc –lnuvp 5000 will create a udp server with port 5000. |

**18 ->** spoof\_pkt(pkt) defines the function how to process each captured packet pkt.

ICMP has type field: type 8 stands for request and type 0 stands for reply. You do not remember fields of ICMP other mentioned in the program. It has a data field (just like TCP has a data field which is application data, also like IP packet has a data field which might a TCP segment).

All IP() and TCP(), Ether(), UDP(), ICMP() are packet class. It field can be accessed. For example, if pkt=IP(dst=8.8.8.8)/ICMP(), we have pkt.dst=“8.8.8.8”.

Again, pkt[UDP] is the UDP subpacket of pkt.

**A screen shot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**Week 5 - Chapter 5 - L\_TCP\_Attack-UW-1**   **[ TCP Attacks ]**

**A group of people walking on a street

Description automatically generatedA screenshot of a computer

Description automatically generated**

|  |  |
| --- | --- |
| **3 ->** You can think DoS attack as counterpart of Ambassador bridge blockade in the computer network scenario. In this setting, attacker sends a lot of packets to a server in a short time, render the server is inaccessible to a legal user. In this work, we will focus on a specific DoS attack called syn flooding attack. You will learn the technical details of the attacker. You will perform the detailed attacks using c and python programs respectively. You will also learn the counter measure for this attack. | **4 ->** SYN flooding attack work on any TCP server. Every TCP connection starts with a 3-way handshake protocol, which will establish the connection between client and server. The attack is going to prevent the client completing this protocol execution so that the client-server connection will never be established. |

**A white background with black text

Description automatically generatedA close up of a text

Description automatically generated**

|  |  |
| --- | --- |
| **A screenshot of a computer  Description automatically generated** | **6 ->**There are 6 flag bits URG, ACK, PSH, RST, SYN, FIN on TCP header. If SYN is marked as 1, then it indicates the current packet is a SYN packet, which is the first packet between client and server. The server then replies with a SYN-ACK packet, in which SYN and ACK bits are both marked as 1. |

**5 ->** TCP transmits a TCP packet (called **TCP segment**), consisting of a TCP header and the data. Here is the TCP head format.

It contains a source port and destination port number. For the client-server connection, the client port # is usually randomly chosen while the server port # must be known publicly as it must be known to the client before the client can make a connection to the server. Well-known server ports # are telnet 23, SSH 22, HTTP 80 and HTTPS 443. The sequence number in the format is used to make sure the packets are received by the recipient in order.

At the sender side, the intial sequence # is picked randomly. If the current outgoing packet has a sequence # s and the data size of Data is N, then the sequence # of the next outgoing packet is s+N. It should emphasized that due to the difference of routing paths for different packets, a packet will larger sequence number might arrive at the destination earlier than a packet with a smaller sequence #. But looking at the packet sequence numbers, the receiver can restore the original packet order. The acknowledge number. The acknowledgement number is to tell the other side the sequence # of the next packet expected.

|  |  |  |
| --- | --- | --- |
| A diagram of a server  Description automatically generated | | **7 ->** This is the 3-way handshake protocol. After the protocol execution, the connection between them is established. It should be emphasized that the initial sequence number x is chosen by client randomly and sequence y is chosen randomly by server. |
| A diagram of a computer  Description automatically generated | | 8 -> The three-way handshake protocol is between a client socket and a server socket. This server socket is called a welcome socket. All clients will establish the connection with the welcome socket. The server socket, when receiving the SYN packet, creates a data structure TCB block to store the client state (including source/destion IP, source/destition port and sequence numbers. This TCB block will be saved a queue called SYN queue. Then the server socket sends SYN-ACK packet to the client and waiting for the client to finish with a ACK packet. If this ACK packet indeed returns to the server. The server will move the TCB block to another queue called accept queue (which stores the TCB blocks for all clients who have completed 3-way handshake protocols). But this is the end of TCP establishment. The server will in fact create a new socket to communicate with the client socket. After this the connection is established and the TCB block will be removed from accept queue. |
| **9 ->** When the server sends out SYN-ACK packet and before client returns the ACK packet, the 3-way handshake protocol is not completed and so it is called half-open connection. If an ACK packet is returned to server for long, then the server will retransmit it. After waiting for a while, if it is still not returned, then it will be retransmitted again. In SEED Ubuntu, it will has 6 retranmissions. This retransmissions make the TCB block stays in the SYN queue for long. Since more TCB records for other clients keep coming in, this might make the queue full. | | |
| A screenshot of a computer  Description automatically generated | **10->** Now if an attacker sends a lot of SYN packets to the server but do not answer the SYN-ACK packets, then TCB records might make the SYN queue full quickly. In this case, if a new client sends a SYN packet to server, the server will not answer it as there is no space to store this client’s TCB record. In the client’s view point, the server is unavailable. That is the denial of service. By the attacker not snwering SYN-ACK packet, we implicitly assume that the source IP in the attacking SYN packet is the attacker’s IP address. This is certainly not good as it can be easily blocked by a firewall at the server. Practically, the attacking SYN packet actually uses random source IP address. | | |

**A white background with black text

Description automatically generatedA screenshot of a computer

Description automatically generated**

|  |
| --- |
| **12->** In our experiment, we created a subnet 10.9.0.0/24. It contains a victim server 10.9.0.5 and an attacker 10.9.0.1 and two legal users 10.9.0.6 and 10.9.0.7. We will target on telnet server at 10.9.0.5. SYN flooding attack is not a new attack. It already has counter-measure. In Ubuntu, the counter-measure has been implemented. So before our attack, we need to distable this counter measure. This is done by setting syncookies bit in server to 0. Our C program attack is done using synflood.c . This program keeps creating random SYN packets sending to the server. |
| **13 ->** Now if you telnet from user machine 10.9.0.6 to the server, it will fail. Then, if you check the server TCP connection status, you will find that there are a lot of SYN\_RECVs status for the telnet server. These are the half-open connections. We use the word counter command to check the number of connections, it is 97. |

**A screenshot of a computer program

Description automatically generatedA screenshot of a computer error

Description automatically generated**

**A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated**

|  |
| --- |
| **15 ->** The program first generates a SYN packet using iph as ip header, tcph as tcp header and pkt that combines iph and tcph, to be syn packet. While-loop generates random source IP, source port # and random seq #. This sends a lot of SYN packets. |
| **16 ->** The SYN flooding attack makes use of the limited size for SYN queue. The counter measure is going to remove the SYN queue. That means now the server will not store any client state. |
| **18 ->** But if the server does not store any client information, then the client can just send a ACK to the server. The server has to accept it because it is possible that the client already sent SYN packet while the server can not verify this. This makes the protocol is still vulnerable., because the attacker can just send a lot of ACK to the server and the server has to allocate a socket with some space (that is intended to communicate with the client who does not exist). |

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**A diagram of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated**

|  |
| --- |
| **19 ->** To avoid the problem, the server in fact generates y using hash function on a secret key K and the TCB record (without y), where K is only known to server. Now when the client returns a ACK to server, server can extract the TCB record from ACK and compute the hash value H(K, TCB) and compare with y extracted from ACK. |
| **20 ->** Now if the attacker only sends ACK to the server, he must compute y=H(K, TCB) in order to pass the verification. But this is impossible, as only the server knows K. |
| **21 ->** There is one more way to disconnect. |

A diagram of a block

Description automatically generatedA computer screen shot of a computer connection

Description automatically generatedA screen shot of a computer program

Description automatically generatedA diagram of a server

Description automatically generated

|  |
| --- |
| **23 ->** In this program, to create a response packet, the source/destination IP are swapped and source/destination port # are swapped too. The seq # in the response packet is the ack # in the receiving packet. When you sniff, you need to sniff the subnet of 10.9.0.0/24 (on the attacker VM, this is to use **iface br-xxx**). Check ifconfig to see this interface. Usually, you can sniff a particular protocol, port, ip address, this is what specified in the **MyFilter** variable. |
| **24 ->** Hijacking attack is techinically similar to reset attack. In reset attack, you send a reply TCP packet with flagbit R. In Hijacking attack, you send a reply TCP packet with some message. To Hijiacking Telnet, this message is a certain telnet command. So the attack behaviour in the attacker program is similar.  As for the reset attack, you need to make sure the four-tuple should be correct. The sequence # should not be too large or too small; otherwise, it will be rejected. |

A screenshot of a computer error

Description automatically generatedA close up of a text

Description automatically generated

|  |
| --- |
| 25 -> To make sure it not too large, |

A screenshot of a computer code

Description automatically generatedA screenshot of a computer program

Description automatically generated

A white text with black text

Description automatically generatedA screenshot of a computer error

Description automatically generated

A screenshot of a computer error

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Week 6 - Chapter 6 - L\_CRYPTO**  **[ Introduction to Cryptography ]**

A diagram of a function

Description automatically generatedA white paper with black text

Description automatically generated

A screenshot of a phone

Description automatically generatedA white background with black text

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screenshot of a computer error

Description automatically generatedA screenshot of a computer screen

Description automatically generated

A diagram of keys and words

Description automatically generatedA black and white text with black text

Description automatically generated

A black and white text with numbers and symbols

Description automatically generatedA white background with black text

Description automatically generatedA screenshot of a computer

Description automatically generated

A white background with black text

Description automatically generatedA screenshot of a math problem

Description automatically generated

A math problem with numbers and equations

Description automatically generated with medium confidenceA screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA white paper with black text

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A diagram of a key

Description automatically generatedA white paper with black text and black text

Description automatically generated

A screen shot of a computer

Description automatically generated A screenshot of a computer code

Description automatically generated

A diagram of a credit card

Description automatically generatedA screenshot of a computer code

Description automatically generated

A black text on a white background

Description automatically generated

A diagram of a computer codebook

Description automatically generatedA close-up of a graph

Description automatically generated

A diagram of a block chain

Description automatically generated

A black text on a white background

Description automatically generated

A screenshot of a computer program

Description automatically generatedA white background with black text

Description automatically generated

A white board with text and a cartoon character

Description automatically generated