Comp 4580 A5

Katrina Dotzlaw [redacted]

Task 1: SYN Flooding

To build the docker container images, I used `docker-compose build`. Then, I started the docker containers with `docker-compose up`. The image below shows that there are 4 container images running: attacker, user1, user2, and victim.

```
[04/03/24]seed@VM:~/.../volumes$ docker-compose build
attacker uses an image, skipping
Victim uses an image, skipping
User1 uses an image, skipping
User2 uses an image, skipping
[04/03/24]seed@VM:~/.../volumes$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating user1-10.9.0.6  ... done
Creating seed-attacker   ... done
Creating user2-10.9.0.7  ... done
Creating victim-10.9.0.5 ... done
Attaching to seed-attacker, user1-10.9.0.6, victim-10.9.0.5, user2-10.9.0.7
user1-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
victim-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
user2-10.9.0.7 |  * Starting internet superserver inetd                 [ OK ]
```

Using `dockps` gets a list of all the containers.

```
[04/03/24]seed@VM:~/.../volumes$ dockps
a45d5632f4fb  victim-10.9.0.5
82231ce1dbac  user2-10.9.0.7
f64f7fc583e6  seed-attacker
8ceb22052d14  user1-10.9.0.6
```

Looking at this image, I can see that the victim container has an id beginning with `a4` and the attacker has an id beginning with `f6`. To get on the attacker container, I use `docksh f6` and to get on the victim container, I use `docksh a4` on a seperate terminal. Additionally, a 4th terminal is open to my VM so I can use telnet later on and compile programs using gcc.

Since we were only instructed to do tasks 1.2 and 1.3, the queue size remains 128.

Task 1.2: Launch the Attack Using C

The synflooding attack will be launched using the `synflood.c` program provided. After navigating to the volumes folder on my host VM, I compiled `synflood.c` with `gcc -o synflood synflood.c`

```
[04/03/24]seed@VM:~/.../volumes$ gcc -o synflood synflood.c
```

On the victim container, I double checked that the syncookie countermeasure was off.

```
[04/03/24]seed@VM:~/.../volumes$ docksh a4
root@a45d5632f4fb:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
```

On the attacker container, I launched the attack with `synflood 10.9.0.5 23`.

```
[04/03/24]seed@VM:~/.../volumes$ docksh f6
root@VM:/# cd volumes
root@VM:/volumes# synflood 10.9.0.5 23
```

Then on the victim container, I checked the queue usage with `netstat -tna | grep SYN_RECV | wc -l`, which counts the number of SYN_RECV connections. As shown in the image below, there are 97 SYN_RECV connections. Note that the queue size is 128, but 1/4th of it is reserved for previous connections.

```
root@a45d5632f4fb:/# netstat -tna | grep SYN_RECV | wc -l
97
```

In order to accurately test the synflooding attack without the countermeasure, I flushed all past records of connection on the victim container. Without flushing, the previous connection will bypass the synflooding attack and allow connection anyways because part of the queue stores previous connections.

```
root@a45d5632f4fb:/# ip tcp_metrics flush
```

Now that past connection records have been flushed, I can attempt to telnet onto the victim container from my host VM.

```
[04/03/24]seed@VM:~/.../volumes$ telnet 10.9.0.5 23
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
```

As shown above, the connection times out and I am unable to connect to the victim container using telnet. This is because the queue is full of partial SYN_RECV connections and the previous connection records, so any new connections are refused.

Task 1.3: Enable the SYN Cookie Countermeasure

The syncookie countermeasure essentially causes connections to not be held in the queue. Instead, the victim container passes a cookie with its SYN+ACK and drops the connection from its queue. Then if the victim gets an ACK response with a cookie, it recalculates the cookie using known connection information. If the newly calculated cookie matches the cookie the victim recieved with the response, the connection is ESTABLISHED. With this countermeasure in place, the queue doesn't need to store previous connection records, since connections rely on calculating and matching cookies.

First, I needed to turn the syncookie flag back on in the victim container. I also flushed past connections again just in case.

```
root@a45d5632f4fb:/# ip tcp_metrics flush
root@a45d5632f4fb:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
```

Next, I ran the attack again from the attacker container using `synflood 10.9.0.5 23`.

```
[04/03/24]seed@VM:~/.../volumes$ docksh f6
root@VM:/# cd volumes
root@VM:/volumes# synflood 10.9.0.5 23
```

Then I went back to the victim container and checked the number of SYN_RECV partial connections.

```
root@a45d5632f4fb:/# netstat -tna | grep SYN_RECV | wc -l
128
```

There are 128 partial connections, so the entire queue is being used for SYN_RECV connections. It makes sense that there are more connections in the queue because it doesn't store past connection records. Additionally, the way the countermeasure works, a response is needed before a connection is established, and initial connection attempts are constantly dropped from the queue.

On my host VM, I can successfully telnet onto the victim container. This shows that the syncookie countermeasure helps mitigate the synflooding attack.

```
[04/03/24]seed@VM:~/.../volumes$ telnet 10.9.0.5 23
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
```