NETWORK SECURITY HW2

GUR TELEM 206631848

1. Question - Port scanning and Wireshark sniffing

1.1. **Part 1.**

(1) The command I ran was `sudo nmap -T4 -p- -sS -A scanme.nmap.org | tee nmap_full_out.txt`
(2) The IP addresses of `scanme.nmap.org` IPv4: `45.33.32.156` ; IPv6: `2600:3c01::f03c:91ff:fe18:bb2f` (this address wasn't scanned because we weren't required to scan IPv6).
(3) The open ports are:
  (a) `22` for `SSH` service.
  (b) `80` for `HTTP` service
  (c) `9929` for `nping-echo` service
  (d) `31337` for `tcpwrapped` service
(4) According to `nmap` the application listening for port `80` is `Apache/2.4.7 (Ubuntu)` (version included)
(5) The flag I used was `-A`. Using this flag (specifically script scanning) is considered intrusive so systems on the host's side may be in place to detect such attempts and alert someone or even initiate tighter security. It's also somewhat heavy to use this flag (some of the features like OS detection may slow down scans of many hosts).
(6) Features enabled:
  (a) traceroute - traces the route to the host using multiple packets with increasing TTL values

```
TRACEROUTE (using port 80/tcp)
HOP RTT       ADDRESS
1   1.41 ms   OpenWrt.lan (192.168.1.1)
...
9   201.74 ms if-2-4.csw6-fnc1.linode.com (173.230.159.87)
10  201.79 ms scanme.nmap.org (45.33.32.156)
```

  (b) OS detection - tries to detect which OS the host is running:

```
Aggressive OS guesses: Linux 5.0 (95%), Linux 5.0 - 5.4 (95%), Linux 5.4 (94%),
```

1.2. **Part 2.**

(7) A closed port is a port that no application is listening on. The response to the SYN request is RST/ACK.
(8) A filtered port is a port that packets sent to it are ignored and no response is sent at all (blocked by firewall). No TCP packet is sent in response.
(9) `nmap` is sending the following HTTP requests: GET, OPTIONS, PROPFIND, POST

2. Question

2.1. **Section - Stateless FW rules table.** This is the table of rules for the **stateless** firewall

Rule	Direction	Src Addr	Dst Addr	Prot	Src Port	Dst Port	Ack	Action
spoof1	inbound	Internal	Any	Any	Any	Any	Any	Deny
spoof2	outbound	external	Any	Any	Any	Any	Any	Deny
GS_i_in	inbound	External	$IP(GS_i)$	UDP	Any	10000	Any	Permit
GS_i_out	outbound	$IP(GS_i)$	External	UDP	10000	Any	Any[1]	Permit
GW_in	inbound	External	$IP(GW)$	TCP	Any	20000	Any	Permit
GW_out	outbound	$IP(GW)$	External	TCP	20000	Any	Yes	Permit
Default	Any	Any	Any	Any	Any	Any	Any	Deny

Any other communication is blocked using the "Default" rule. So the FS will not be allowed to pass the FW1.

[1] In stateless we don't know if it's a response when using UDP

2.2. **Section.** This is the table of rules for the **stateful** firewall (checked only against $ACK = 0$)

Rule	Direction	Src Addr	Dst Addr	Prot	Src Port	Dst Port	Action
spoof1	inbound	Internal	Any	Any	Any	Any	Deny
spoof2	outbound	external	Any	Any	Any	Any	Deny
GS_i_in	inbound	External	$IP\left(GS_i\right)$	UDP	Any	10000	Permit
GW_in	inbound	External	$IP\left(GW\right)$	TCP	Any	20000	Permit
Default	Any	Any	Any	Any	Any	Any	Deny

And in addition there's the dynamic table which will let any open sessions to continue. Including the method to create a dummy session for UDP that was taught in class.

2.3. **Section.** Are the clients at vulnerable?

No. Because only the GS_i servers are connecting with the game's client and we trust them not to be malicious (if the company was malicious then the client itself could simply be a malware and it doesn't matter if we have Log4j or not).

Also, the client doesn't listen to any ports so no attacker can initiate a session to any of the clients so they're not at risk.

2.4. **Section.** The servers are also using Log4j.

Now this is a problem because a malicious client can craft a special packet to get control of a GS_i server and then they can connect with the FS using the GS_i as a proxy, thus, bypassing FW1 and being able to retrieve the source code for the game.

The step the attacker would do are something like this

(1) Craft packet that exploits Log4j's one-day exploit.
(2) Take control of say GS_1.
(3) Run a proxy server on it that listens on port 10000 UDP (we can implement TCP over UDP so there's not really a problem with reliability)
(4) Using the proxy server, access FS and download the source code of the game.

2.5. **Section.** Why is that important? All of the steps are possible with stateful packet filtering as well as stateless. We can initially send packets over UDP to port 10000 and then we simply communicate "normally" over the same port that we're allowed to. The rest of the traffic isn't even going through FW.

Even the implementation of the TCP over the UDP protocol is possible with stateful, you don't even need to turn off the server for other clients, the attack can simply wrap the server with almost 0 downtime and simply reroute unrelated packets (unrelated to the game) and treat them as communication of the malicious proxy server.

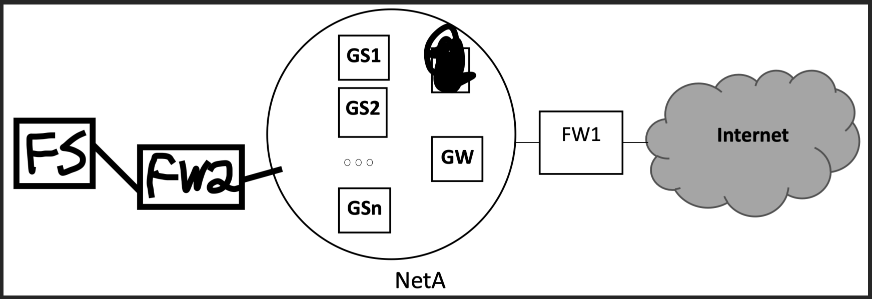
I'm not sure I understand what the problem with stateful is.

2.6. **Section.** I have a little more than 2 suggestions but I put the best ones first:

(1) In the FTP settings, permit access only read access and only to the folder where the binaries of the game are located (and make sure it doesn't contain the sources).
(2) Set the authentication keys on the GS_i servers accessibly from a different user (say root) than the one running the game and Log4j and trust Linux's security to not allow privilege escalation. Detailed changes would be (assuming the FTP authentication is done using RSA private/public keys):
    (a) Create users: `game_usr` and `update_usr` and add them to the same group `service_usrs`
    (b) Change ownership of the public and private keys of the RSA key to `update_usr`
    (c) Set "other" and "group" permissions for the keys to 0 (so no READ/WRITE/EXECUTE).
    (d) Set ownership of the files required by the game server to be owned by the group `service_usrs` and make sure to set RD/WR permissions to both owner and group.
    (e) When wanting to update a GS_i we can simply connect from the user `update_usr` on GS_i to FS using FTP and update the files. Make sure that the executables used for the update don't have any access from the user `game_usr`.
    (f) Now connect to the user `game_usr` and simply restart the game server (not the machine itself). NOTE: It's important to run the game server from this user so we don't expose.
    (g) OPTIONAL: We can modify the kernel such that it will only allow the `game_usr` to communicate with the client and not send packets to internal hosts, specifically FS.
    (h) No extra devices needed on the network.
(3) Initiate request from the FS:
    (a) Put a second firewall (FW2 - stateless is fine) between the FS and the rest of the internal network (isolating it)
    (b) Disable the FTP server on FS and run an FTP service on each of the GS_i.
    (c) Deny any sessions initiated not by the FS in FW2 but allow FS to initiate connections to the GS_i's FTP services.
    (d) When it's time to update, upload from FS to each of GS_i using their FTP service.
    (e) FW1 is unchanged
    (f) Need one extra device - firewall (or even only software firewall because speed is not of the essence for updating)
(4) Add a proxy server in the internal network for outbound traffic:

   (a) The proxy can inspect each packet going back and forth and make sure no data similar to the source files are sent outside the network.
   (b) I would also suggest escaping and/or capping messages the clients send to prevent even using the vulnerability but I guess this is considered "patching" the exploit and not allowed.

In suggestion 3 the network looks like that:



In the first two suggestions the network would look the same.

I'm not bothering with suggestion 4's network map because it's simply a bad idea (explained in next section).

2.7. **Section.** Compare between the options.

First of all, note that in real life, the exploit should be fixed. It's a rookie mistake to ignore vulnerabilities.

Let's inspect each suggestion:

(1) Prevents anything except binaries to be downloaded from the FS (even inside the internal network)
   (a) (good) This will prevent a compromised host (GS_i) to download source files from FS
   (b) (good) It requires no extra hardware and doesn't affect the connection to the clients.
   (c) (good) It's the easiest to configure. A matter of minutes and requires very little testing to confirm it's working.
   (d) (bad?) This assumes FTP mitigates properly the access to folders and files.
   (e) This is a good idea but not perfect. Since the attacker can still access the binaries and transfer those and possible (with effort) reverse engineer them, possibly finding more vulnerabilities but also being able to get disassemble and modify the executable.
   (f) (bad) It doesn't isolate the server containing the sensitive information so an attacker can scan for other weak points in the internal network to access the FS.
(2) Prevent connection to FTP from the compromised account
   (a) (good) This option limits the surface of attack by isolating the a component that is vulnerable as much as possible.
   (b) (good) Prevent the attacker from even connecting to the FTP so they couldn't download anything, let alone source code.
   (c) (good) It also requires no extra hardware but a little more complex to configure.
   (d) (good) It also doesn't affect the connection to the clients.
   (e) This option is good and only but again not perfect as the attacker can still access the binaries that are located on the compromised server.
   (f) (bad) It doesn't isolate the server containing the sensitive information so an attacker can scan for other weak points in the internal network to access the FS.
(3) Move the FTP server to the GS_i and initiate the update from FS
   (a) This completely removes the option for an attacker to initiate a connection to FS
   (b) It also removes the ability of a compromised server to choose what data to retrieve even once an update has started.
   (c) It also doesn't affect the connection to clients.
   (d) Separating sensitive servers and servers exposed to the Internet is a good idea. Allowing a few points as possible between the FS and the GS_is and monitoring that traffic by any combination of stateless/stateful/proxy is a good idea.
   (e) This option requires a little more maintenance and possibly heavier on resources as instead of a single FTP server we now have n servers (for each GS_i).
   (f) It's not as bad though because most of the time the FTP servers are idle (except during update when the GS_i can't serve clients anyways so it has excess resources either way).
(4) Proxy to inspect data going out of the internal network and blocking data not permitted
   (a) This is a very bad option for a few reasons
   (b) A game server needs to be **fast** and a proxy would add noticeable latency for all clients.
   (c) An attacker can encrypt the data before sending so the proxy wouldn't be able to inspect it.
   (d) Even if the proxy block anything **except** permitted packets, it's still possible to encode the source code into a bunch of "permitted" packets (for example, go left is 0 and go right is 1 and continuously send packets representing the 0s and 1s that are the raw bit of the source code).

(e) It's also expensive because it requires a whole new host machine and a lot of man hours to code the detection of things. And it's a new type so a whole new maintenance procedure.
(f) I'm not even going to bother paining it.

That's basically it. I'd probably use a mix of the first 3 suggestions. Will possibly transfer the game files (zipped) over SSH (that is probably present anyways for server machines).