Workshop in Information Security – Homework #4

Goal:

- Implementing statful connection tracking
- Be familiar with common protocols
- Support deep inspection in common complex protocols
- Create the infrastructure for the next level

Description:

In this assignment, we will add the statful inspection over our previous stateless firewall.

The main two features we will add:

- Tracking the state of TCP connections and enforce its correctness
- Inspection the DATA as a stream and not only the headers

You should **MOVE** your hook function from “forward” point to “pre-routing” point. Assume every packet here is not targeted to the FW just only passing through.

Part 1 – Connections Table:

This is the meaning of **stateful** inspection, instead of checking each packet regardless the connection state, here we want to make the relation between the packet and the connection.

Note: this assignment is only for TCP connection. For other connection leave it as in previous assignments.

If the incoming packet is with ACK==0, it means it’s the first packet of a connection (SYN packet). We will check the packet against the **stateless rule table**. If the verdict is accept, we will create a new entry in our connection table.

If the incoming packet is with ACK==1, we will skip the stateless checking and will check it against the dynamic connection table and will update the state if needed.

Each entry in the connection table will be:

- Source IP
- Source port
- Destination IP
- Destination port
- State

Each connection should have one entry for c2s side and one entry for s2c side.

We will track and enforce the TCP states according to the TCP state machine. Find it in the web.

Store the connection table in a dynamic data structure such as klist, dynamic allocated table etc.

It's highly recommended to complete this part before moving to the second part.

The drivers:

Description	Driver path	Permissions	Passing data
Showing connection table	`/sys/class/fw/conns/conns`	R	List of connection

Part 2 – Inspecting the Data:

We will implement this part as man in the middle (MITM) in order to avoid building the stream of the data as the TCP stack is doing.

For TCP ports which are **not** 21 (FTP) and 80 (HTTP), just skip this part and keep the logic of part 1.

Run 2 user-space sockets, one for each protocol. They will listen on ports 210 (for FTP MITM) and 800 (for HTTP MITM).

When the packet arrives to pre-routing point, after doing all the logic from part 1, change the destination IP address to the FW IP address and the destination port to 210 (if the original was 21) or to 800 (if the original was to 80).

Then the listening process will get the **stream of the data**. After the inspection (as will be explained shortly) the process will write it to the original destination. So we need to fake in local-out point the source IP to the original host's IP.

In fact, there are 4 points that we need to re-write a fields in the IP/TCP headers (and of course fixing the checksum after it):

- Client-to-server, inbound, pre-routing, we need to change the dest IP and dest port
- Client-to-server, outbound, local-out, we need to change the source IP
- Server-to-client, inbound, pre-routing, we need to change the dest IP
- Server-to-client, outbound, local-out, we need to change the source IP and source port

So, instead of a regular identifier of a connection, which is sIP, sport, dIP, dport, we have here more one: the local-process source port towards the server. We need to save it in the connection table and create a driver to get/set it from user-space.

Supporting complex protocols:

- HTTP:
    - We want to block an HTTP response of type “text/csv” or “application/zip”. This information can be found in the HTTP header “Content-Type”
    - In the local user-space HTTP program, look for this header, and if it contains this type, don’t pass the data to the client
- FTP:
    - We want to dynamically allow active FTP data connections
    - We will track in the local user-space FTP program until we see the PORT command. Then we will pass the arguments to the kernel through a special driver and we will open in the connection table the requested entry
    - Of course, when the first packet of this connection will come, it will be with ACK==0. We need to escape this case from checking it against the static rule table as usual

Testing:

Install on host2 an FTP server and HTTP server, and test all the cases.

Final logic in pre-routing hook:

If it’s non TCP and it’s supported protocol --> check against the **static rule table.**

If it’s TCP and ACK == 0 --> check against **static rule table**. If accepted, create entry in the **connection table.** If the destination port is 21 or 80, do in addition the **proxy** stuff. See exception for FTP data connection later.

If it’s TCP and ACK == 1 --> check against **connection table** and update the state if needed. If the destination port is 21 or 80, do in addition the **proxy** stuff.

If it’s FTP data connection with ACK == 0 --> check against **connection table**.

User space program:

Add to the program from previous assignment the next option: “show_conns” as an argument. This will print the connection table.

Submission

Prepare a ZIP file, contains:

- “module” folder, includes hw4secws.c file (the module) and the Makefile
- "user" folder, includes the user-space program and (if needed) the Makefile
- “http” folder, includes the HTTP proxy user-space program
- “ftp” folder, includes the FTP proxy user-space program
- Dry documentation

General rules for submission, valid for the next assignments as well:

- Document your code
- If you use a code from the internet, document it and add the source
- Individual submission
- If needed, split the files into a modular files

Good luck!

