Hw4secws documentation - Ori Petel

The submission consists five items:

- /module – The kernel space code directory.
- /user – The user space code directory.
- /proxy – The proxy code directory
- /documentation.pdf – Dry documentation (this file)
- /prev_doc.pdf – Dry documentation of the previous assignment

Remark: This implementation is built upon the implementation of the previous exercise.
You can find its documentation under the name "prev_doc.pdf"(see attached)

Kernel Side Flow

Inspection

The inspection flow has changed, since the proxy logic has come to equation.

The inspection flow now is the following:

1. First check if the packet involves proxy, based on the connection table.
   If so, we change fields of the packet.
2. Accept any packet in LOCAL_OUT hook point that does not designated to the proxy.
3. If the packet is not TCP packet, we perform stateless packet filtering.
4. If the packet is a CHRISTMAS packet, we drop.
5. In case the connection related to the packet doesn't exist:
   - If it is a SYN packet, create new entry in the connection table.
   - If it is specifically SYN packet from internal network to external HTTP/ FTP server, construct the proxy.
   - Else – drop the packet, since the connection does not exist.
6. In case the connection exist and not proxy, perform statefull inspection
   (i.e. enforce the state machine of the connection)

Kernel side modules

Tracker

This new module plays a crucial role in the statefull inspection. It contains:

- Contains the structure "ctable" the represents all the current connections over the firewall.
- Provides connections tracking functionality:
  get connection, add connection, remove connection, free connections.
- Enforces validity of TCP connection (by TCP state machine).
- Provides char device functionality for "Showing connection table".

An interesting fact about my design is that I chose to implement **one entry for both** c2s (client to server) and s2c (server to client) sides of the connection.

Every packet passes through the firewall, given the current state, defines the next state of the connection and **the direction of the next packet that we expect to receive**.

In this way (by remembering the expected direction), we are able to enforce TCP validity more efficiently in both space and time.

Proxy

This new module plays a crucial role in the proxy logic & routing. It contains:

- Functions for finding proxy connection in the table by $client(ip, port)$ or proxy port of the local machine.
- Proxy inspection operations: fixing the checksum of a packet after altering it, constructing proxy connection on the kernel side, routing proxy packet.
- Proxy device operations: sending the proxy port of the local machine to the kernel, sending the connection details of a new FTP DATA CHANNEL announced.

User-space

I chose to implement the user space program in c language.

In order to produce the executable file "main", you should run the command "make" in /user directory.

Conn handler

This module handles all the connections issues in the user space program:

- Connection_t structure and required enums (that match to the kernel side).
- Conversion of buffer to a connection structure.
- Functions for conversion between connection fields and strings.
- Function for conversion between the connection itself and a string that representing the connection

User (contains the "main" function)

This module preforms the actual reading and writing flow to and from the firewall devices. (Exactly as before)

Local-Machine Proxy

I chose to implement the user space **proxy** with in Python language.

In order to activate the **HTTP** proxy functionality you should run the command "python http_proxy.py" from /proxy directory, before loading the module.

In order to activate the **FTP** proxy functionality you should run the command "python ftp_proxy.py" from /proxy directory, before loading the module.

Remark: make sure you have up to date version of python3 on your computer, since it uses relatively new library called: "subprocess"