Hw3secws documentation - ori petel

The submission consists 3 parts:

- /module – The kernel space code directory.
- /user – The user space code directory.
- /hw3secws_doc.docx – Dry documentation (this file)

Kernel Side

Let us describe our code design first.

The code is divided into several logical sections.

For each logic section we will specify the following:

- What role does it play?
- Where is it located in the code?
- What logic happens inside **on high level**, and which **modules** is it using?

Implementation details will appear later, for each module apart.

Hooking

On module_init: we register our hook function on “forward” point, using Netfiler API.

On module_exit: we unregister our hook function.

We perform the registration/ unregistation in "hw3secws.c" module.

Register/ Unregister Devices

On module_init: we register our devices, and defining operations on the device.

On module_exit: we unregister our devices.

We perform the registration/ unregistation in "hw3secws.c" module.

Inspection

The inspection takes place at "filter" module

The flow of the inspection does the following:

1. Our hook function " fw_filter ()" is called for each packet passing through the Firewall.
2. We first extract the required fields from the packet, and put it in an appropriate structure.
   we also detect and notify in the following cases:
   - If the packet is a loopback packet
   - If the packet isn't above TCP, UDP, ICMP
   - If the packet is a "Christmas tree packet".

This is all done by module "parser".

3. We iterate over the rules in the table and looking for a match.
   If no rule matched, we drop the packet (white list)
4. We log the action with the packet’s data, and aggregate the logs.
   This is done by the module "logger".

Loading & Showing rule table

Loading – raw data of the rules is passed to us, and we locate it in an array of rule structures (of size MAX_RULES=50). We override previous rules if exist.
If the rules passed are invalid, we disable the firewall and turn on the "inactive" flag.

Showing – We pass raw data of the rules to the user. (In case "active" flag is on)

We perform this logic in the module "ruler".

Showing & Resetting logs

Showing – We pass raw data that represents the logs to the user space.

Resetting – In case of any kind of writing into the file is preformed, we reset the logs, and free unneeded resources.

We perform this logic in module "logger".

Kernel side modules

fw

Contains all the general or mutual auxiliary function/ structures our kernel module use.

Also implements useful Macros and tools for debugging. (To turn on of debug mode uncomment the #define DEBUG line)

Parser

Contains all the functions relating to the parsing of a socket buffer (packet) at hook phase.

Finally, it returns the information in an organized packet_t structure.

In addition, it detects and notifies in case of special packet (loopback packet, unsupported protocol, XMAS packet).

Ruler

Contains all the functions relating to the work with rules:

- Special structure rule_table & getters and setters
- Functions for conversion between rule and buffer (the buffer is a more efficient and succinct representation of the rule for transferring through files)
- Rules validation
- Implementation of sysfs show, store attributes.

Logger

Contains all the functions relating to the work with log:

- Linked list structure for the log
- Pool structure & getter for mass memory (kmalloc/ kcalloc) allocation, instead individual allocation for each log entry added.
- Logic functions – Finding a match between log entries, Logging the action on a packet, Log cleanup
- Conversion of log_row structure to a buffer (the buffer is a more efficient and succinct representation of the rule for transferring through files)
- Char device open, read functions implementation.

- Sysfs store implementation for clearing the log.

Filter

This module implements the actual filtering that gets done when hooking packet through NetFilter:

- Function for finding a match between rule and packet
- Function for creating a log_row template from a packet.
- Function for the filtering flow

This module actually unifies and interact with all three modules above.

Hw3secws

The skeleton of the kernel module.

Contain the init, exit function. Handle all the registration/ unregistration of hooks and devices, and the module technical issues.

User-space

I chose to implement the user space program with **c language**.
In order to produce the executable file "main", you should run the command "make" in /user directory.

Modules

Interface

Provides all the general or mutual auxiliary function/ structures our user space program uses.

Also implements useful Macros and tools for debugging. (To turn on of debug mode uncomment the #define DEBUG line)

Rule handler

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

- Rule_t structure and required enums (that match to the kernel side).
- Functions for conversion between rule and buffer.
- Functions for conversion between rule fields and strings.
- Function for conversion between the rule itself and a string that representing the rule (specified in the examples files)

Log handler

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

- Log_row_t structure and required enums (that match to the kernel side).
- Conversion of buffer to a log_row structure.
- Functions for conversion between rule fields and strings.
- Function for conversion between the log_row itself and a string that representing the rule (specified in the examples files)

User (contains the "main" function)

This module preforms the actual reading and writing flow to and from the firewall devices.