# **Hw3secws documentation - ori petel**

The submission consists 3 parts:

* /module – The kernel space code directroy:  
  hw3secws.c, Makefile
* /user – The user space code directroy:  
  user.c , Makefile
* /hw3secws\_doc.docx – Dry documentation (this file)

## Kernel Side

Let's describe first our code design.

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.

The registration is done in "hw3secws.c".

### Register/ Unregister Devices

On module\_init: we register our devices, and defining operations on the device.

On module\_exit: we unregister our devices.

The registration/ unregistration is done in "hw3secws.c" module.

### Inspection

The inspection takes place "filter" module

The flow of the inspection does the following:

1. Our hook function " fw\_filtering()" 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 warn in the following cases:

* If the packet is a loopback packet
* If the packet is IPv6 packet, or isn't above TCP, UDP, ICMP
* If the packet is a "Christmas tree packet".

**This is all done by module "parser".**

1. We iterate over the rules in the table and looking for a match.

If no rule matched, we drop the packet.

1. 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 is passed to us (succinct representation), and we locate it in an array of rule structures (of size MAX\_RULES=50). We override previous rules if exist.

Showing – We pass raw data to the user space (succinct representation), that represents the rules.

This is done in the module "ruler".

### Showing & Resetting logs

Showing – We pass raw data to the user space (succinct representation), that represents the rules.

Resetting – if a string starting at '\*' is passed by the user, then we reset the logs, and free unneeded resources.

This is done in the module "logger".

## User-space

I chose to implement the user space program with **c language**.

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