CPSC 526 - Assignment 5

The due date is posted on D2L.
Weight: 15% of your final grade.
Group work is allowed, maximum group size is 2 people.

For this assignment you will write a simple firewall simulator. Your simulator will read in a set of rules and a set of packets from two input files. It will then process each packet by finding and reporting a rule that matches the packet. There will be no network programming in this assignment.

Starter code

Start by downloading starter code from GitLab:

```
$ git clone https://gitlab.com/cpsc526/w25/a5-fwsim.git
$ cd a5-fwsim
```

The starter code contains the following files:

fwsim.py	This is the only file you need to modify and submit for grading. It contains an **incomplete** implementation of the function fwsim(), which you need to re-write. Feel free to reuse any or none of the code in this file. The only requirement is that the fwsim() function you implement follows the correct signature, so that it can be used with the provided fw.py driver.
fw.py	This is the driver code that accepts command line arguments, calls the fwsim() function that you need to implement, and then prints out formatted results. Do not modify this file, and do not submit this file for grading.
rules*.txt	Sample rules files for testing your code.
packets*.txt	Sample packet files.
results*.txt	Expected results for the above.
badrules.txt	Examples of badly formed rules.
badpackets.txt	Examples of badly formed packets.

The starter code includes a small sample of input files. These will not test your code thoroughly and you should design your own test inputs.

Simulator

You must implement your simulator by writing a function fwsim() in the file fwsim.py. The function accepts two filenames as parameters, one for the rules, the other for the packets:

```
def fwsim(rules_fname: str, packets_fname: str) -> list[list[str]]:
```

The function must return a list of tuples of strings, where each tuple of strings will represent the filtering decision for each packet. For each packet, the simulator will find the first rule that matches it and append to the results a tuple consisting of 6 values: action, rule line number, direction, ip, port and flag, as described later. If the packet does not match any rule, set the action to “default”, and the rule line number to empty string. Your solution should follow this pseudocode:

```
1:  rules , packets = read_rules() , read_packets()
2:  results = []
3:  for each packet p in packets:
4:     match = none
5:     for each rule r in rules:
6:        if rule r matches packet p:
7:           match = r
8:           break
9:     if match found:
10:       results.append (match.action, match.line_number, p.direction, p.ip, p.port, p.flag)
11:    else:
12:       results.append (“default”, “”, p.direction, p.ip, p.port, p.flag)
13: return results
```

Rules file format

The rules file will contain an ordered list of firewall rules, one rule per line. The order of the rules is important, as we discussed during lectures. Each rule has 4 mandatory fields and one optional field:

```
direction action ip-range port [flag]
```

The fields will be separated by one or more white spaces. The meaning of each field is:

direction	Specifies the direction of traffic to which this rule applies. Allowed values are "in" and "out".
action	Specifies action to be taken if packet matches this rule. Allowed values are "accept", "drop" and "deny".
ip-range	Defines IP range for this rule. Can be given using CIDR notation, e.g. 136.159.22.0/24, or using a single asterisk "*" to match any address, which is equivalent to 0.0.0.0/0.
ports	Defines list of destination ports for which this rule applies. It is a list of integers 0-65535, separated by commas, with no white spaces in-between. It is also possible to specify *any* port by using a wildcard "*".
flag	This is an **optional** field that describes whether the rule will be applied only to packets that are part of an established session or to all packets. If the field is not present, the rule will apply to all packets. If present, the only allowed value is "established", and it will mean the rule will apply only to packets belonging to established connections.

Here is an example of a file with 5 rules:

```
$ cat rules0.txt
in  accept 136.159.0.0/16  22,443
in  accept *               *       established
in  deny   *               80
in  drop   *               *
out accept *               *       established
```

- The 1st rule accepts packets from IP addresses 136.159.*.* with destination ports 22 or 443.
- The 2nd rule accepts all incoming packets marked as "established".
- The 3rd rule denies incoming packets destined to port 80.
- The 4th rule drops all incoming packets.
- The 5th rule accepts all outing packets that are flagged as "established".

If the simulator cannot find a rule for a packet, it will apply the "default" action to the packet.

Packet file format

The packet file will contain packets, one per line, each consisting of 4 fields:

```
direction ip port flag
```

The fields will be separated by one or more white spaces. The meaning of each field is:

direction	Specifies the direction of the packet. Each packet is either incoming or outgoing. The only allowed values are "in" and "out".
ip	Packet's IPv4 address, specified in dot-decimal notation, e.g. 136.159.5.22. For incoming packets this is the source IP address, and for outgoing packets, this is the destination IP address.
port	Specifies the destination port of the packet, and it is an integer between 0-65535.
flag	This binary flag specifies whether the packet is part of a new (0) session or established (1) session.

Here is an example of a file with 3 packets:

```
$ cat packets0.txt
in  136.159.5.22 22 1
in  10.0.0.44    80 0
out 10.0.1.1     80 0
```

- The 1st packet is an incoming packet, arriving from address 136.159.5.22 and its destination is port 22 (SSH). This packet is part of an existing session (established).

- The 2nd packet is arriving from 10.0.0.44 and its destination is port 80 (HTTPS). This packet is not part of an existing session, and therefore rules with “established” flag set will not apply to this packet.
- The 3rd packet is an outgoing packet, going to 10.0.1.1 to port 80 (HTTP), and it is not part of an established connection.

Output

For each packet your simulator must produce exactly one tuple in the result:

```
(action, lineNo, direction, ip, port, flag)
```

where:

- action is a string, matching the action of the first rule that matched the packet. If no rule could be found that matches the packet, the action should be “default”.
- lineNo is the line number of the first rule that matches the packet. If no rule matches a packet, the line number should be an empty string. The rule lines are numbered consecutively, starting from 1, and all lines are numbered, including blank lines and comment lines.
- direction, ip, port and flag are the parsed fields of the packet.

The fw.py driver will format the output of fwsim(), e.g.

```
$ ./fw.py rules0.txt packets0.txt
accept(1)    in  136.159.5.22    22    1
deny(3)      in  10.0.0.44       80    0
default()    out 10.0.1.1        80    0
```

The above output means:

- the first packet was accepted due to the rule on line 1;
- the second packet was denied by the rule on line 3;
- no rule matched the third packet.

The results above are correct, as they were created using a finished version of the simulator. If you run the incomplete version included with the starter code, the results will be incorrect.

Error handling

Your simulator must not crash under any circumstances and instead it must handle malformed rules and packets in a sensible way. As soon as it detects an invalid input (rules or packets), it must raise a Warning exception. The text of the exception must have the following format:

```
filename:line-number: error-message
```

The filename will be either the rules or the packets filename, the line-number will indicate the line where the error occurred, and the error-message will be a human-readable description of the error. There are some examples of how to raise the Warning exception in the fwsim.py starter code for some malformed input. But please note that the starter code does not handle all errors. Here are some examples of how errors should be reported:

```
$ head -1 badrules.txt
in drop 1.2.3.4/20 21 established x
$ ./fw.py badrules.txt packets0.txt
Simulator error: badrules.txt:1: rule must have 4 or 5 fields
$ head -1 badpackets.txt
in 1.2.3.4 333 1 extra_field
$ ./fw.py rules0.txt badpackets.txt
Simulator error: badpackets.txt:2: packet needs 4 fields
```

You should detect errors such as input files that cannot be read, wrong number of fields in rules and packets, invalid directions, invalid actions, invalid IPs and IP ranges, invalid ports, invalid flags.

Comments, blank lines and white spaces

The input files for both rules and packets may contain blank lines and Python-style comments, which your simulator should ignore during parsing. Blank lines are lines that contain only white spaces, i.e. they contain no visible characters.

A text on a line that follows and includes the hash '#' symbol is a comment. Your parser should strip away all comments from the line before parsing it. It is possible that after stripping a comment from a line, the line becomes blank.

However, the simulator must take blank lines into account when reporting line numbers in results and in error messages. The reported line numbers must match the line numbers in the original input. For example, if your simulator reports there is an error on line 10 of some file, you should be able to open that file in a text editor and find the error on line 10.

The rules and packets lines may contain extra leading and trailing white spaces, as well as multiple white spaces between the individual fields. Make sure you take this into consideration when parsing the input files. I suggest using str.split().

Additional notes

- Your entire solution must be submitted in a single file fwsim.py.
- Your solution must run on the CPSC Linux machines.
- You may not call any external programs, and you may not use any libraries to help you with parsing CIDR notation, IP address parsing, or IP address matching. You must implement all such functionality yourself.
- Because the due date of this assignment is close to the end of the semester, there won't be time for demos. Instead, we will mark your code without you, using automated grading script. It is **important** that the fwsim.py you submit runs with the original fw.py driver.

Marking

We will mark your code by running it on our own test cases. Each test case will be given a score, calculated based on how many packets were correctly matched to the rules. The total mark for your assignment will be the sum of the individual test scores. Here is a possible way we will execute the tests:

```
$ python3 fw.py rules10.txt packets10.txt | diff -y results10.txt -
```

We may apply penalties for the following:

Penalties	
Incorrect error handling	-5% per, up to -30%
Crashes (unhandled exceptions)	-10% per crash, up to -30%
Ugly/undocumented source code	up to -20%

Submission

Both group members must submit 2 files on D2L to get any marks for this assignment:

fwsim.py	Your simulator implementation. Make sure all your code works on the Linux lab machines.
readme.txt	Include names and IDs of all group members at the top. Include any notes you want us to read during marking.

There will be no demos for this assignment.

General information about all assignments:

- All assignments are due on the date listed on D2L. Late penalties will be applied as described in the official course outline and as discussed during the first week of classes.
- Extensions may be granted only by the course instructor.
- Most assignments will be marked by your TAs. If you have questions about assignment marking, contact your TA first. If you still have questions after you have talked to your TA, then you can contact your instructor.
- All programs you submit must run on departmental Linux machines. If we are unable to run your code on the departmental Linux machines, you will receive 0 marks.
- Assignments must reflect your own, or your group's individual work. Here are some examples of what you are not allowed to do for assignments: you are not allowed to copy code or written answers (in part, or in whole) from anyone else; you are not allowed to collaborate with anyone; you are not allowed to share your solutions (including code or pseudocode) with anyone; you are not allowed to sell or purchase a solution; you are not allowed to make your code available publicly, you are not allowed to AI tools to generate your code nor answers. This list is not exclusive. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k.html .
- We may use automated similarity detection software to check for plagiarism. Your submission will be compared to other students (current and previous), as well as to any known online sources. Any cases of detected plagiarism or any other academic misconduct will be investigated and reported.

Appendix – examples

Rules:

```
$ cat rules1.txt
in  accept 136.159.5.5/32     22
in  accept 136.159.5.5/16     80,8080
in  accept *                  443
in  accept 10.0.0.0/31        *
in  deny   *                  21
in  accept *                  *        established
out accept 137.159.0.0/8      *
out deny   10.0.0.0/8         *
out deny   *                  22
out accept *                  *
```

Packets:

```
$ cat packets1.txt
in 136.159.5.5 22 0
in 136.159.5.5 23 0
in 136.159.5.6 22 1
in 136.159.255.5 80 0
in 136.159.0.5 8080 1
in 136.159.255.0 8080 0
in 136.158.5.5 8080 0
in 24.25.26.27 443 0
in 24.25.26.27 444 0
in 24.25.26.27 444 1
in 24.25.26.27 21 1
in 10.0.0.0 1000 0
in 10.0.0.1 1000 0
in 10.0.0.2 1000 0
out 137.255.0.255 33 0
out 10.0.0.133 1 0
out 5.5.5.5 22 0
out 5.5.5.5 22 1
out 5.5.5.5 23 1
```

Simulator output:

```
$ ./fw.py rules1.txt packets1.txt
accept(1)    in  136.159.5.5     22    0
default()    in  136.159.5.5     23    0
accept(6)    in  136.159.5.6     22    1
accept(2)    in  136.159.255.5   80    0
accept(2)    in  136.159.0.5     8080  1
accept(2)    in  136.159.255.0   8080  0
default()    in  136.158.5.5     8080  0
accept(3)    in  24.25.26.27     443   0
default()    in  24.25.26.27     444   0
accept(6)    in  24.25.26.27     444   1
deny(5)      in  24.25.26.27     21    1
accept(4)    in  10.0.0.0        1000  0
accept(4)    in  10.0.0.1        1000  0
default()    in  10.0.0.2        1000  0
accept(7)    out 137.255.0.255   33    0
deny(8)      out 10.0.0.133      1     0
deny(9)      out 5.5.5.5         22    0
deny(9)      out 5.5.5.5         22    1
accept(10)   out 5.5.5.5         23    1
```

Rules:

```
$ cat rules2.txt
in deny 1.2.3.4/32 55
 # rule below should never be applied, because it is
 # 100% shadowed by the previous rule
in drop 1.2.3.4/32 55
 # this rule is not entirely shadowed by the previous one
in drop 1.2.3.4/32 55,56
 # few more rules
out accept 200.200.200.200/25 1,2,3 established
out drop 1.2.3.4/1 *
```

Packets:

```
$ cat packets2.txt
# packets to test first 3 rules
in 1.2.3.4 55 0
in 1.2.3.4 56 0
in 1.2.3.4 57 0
# packets to test last 2 rules
out  200.200.200.127 2  1
out  200.200.200.128 2  1
out  200.200.200.128 2  0
out  127.20.3.5      10 0
out  128.20.3.5      10 0
```

Simulator output:

```
$ ./fw.py rules2.txt packets2.txt
deny(1)      in  1.2.3.4         55    0
drop(6)      in  1.2.3.4         56    0
default()    in  1.2.3.4         57    0
default()    out 200.200.200.127 2     1
accept(8)    out 200.200.200.128 2     1
default()    out 200.200.200.128 2     0
drop(9)      out 127.20.3.5      10    0
default()    out 128.20.3.5      10    0
```