Lorenzo Cavada Mat. 220502 ft. Tommaso Sacchetti

Last exercise of OffTech about Snort Exercise. A description of the exercise can be found here. All the code presented in this recap can istead be found at this link.

BASIC TASK

1. What happens to the traffic to client1 when Snort is not running?

When snort is not running **client1** cannot reach the server. This happens because the snort machine also performs some routing, so if snort is not running the machine does not route the packets correctly.

2. Is this a good thing?

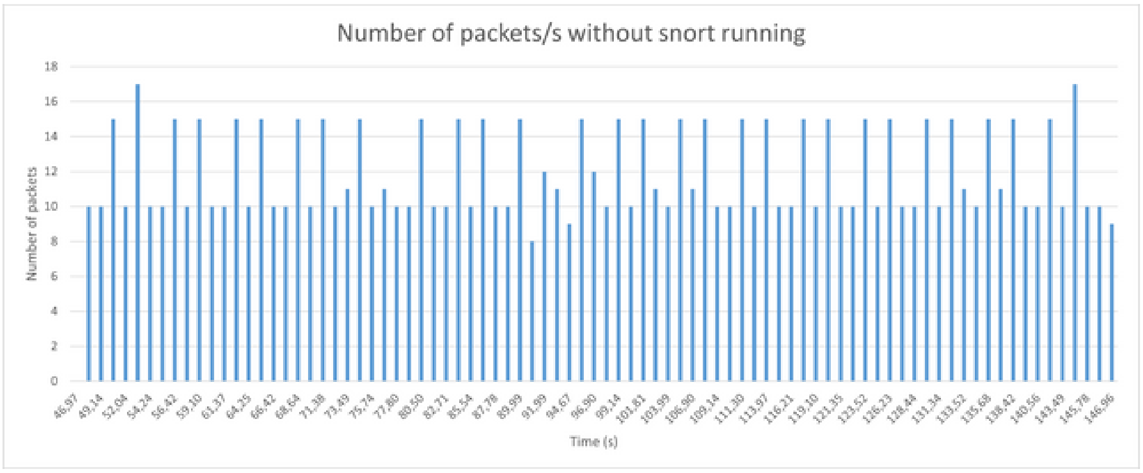
It depends. With the `--daq nfq` option snort uses iptables to handle the routing, this behaviour means that all the routing is done only when snort is no and so there is not separation of duties. On the other hand this also allows for better overall security because since no routing happens when snort is off, malicious packets are blocked by default. The problem is that also legitimate packets are blocked and so the server will appear offline.

3. Based on Snort's output what can you say about the application? What port does it connect to?

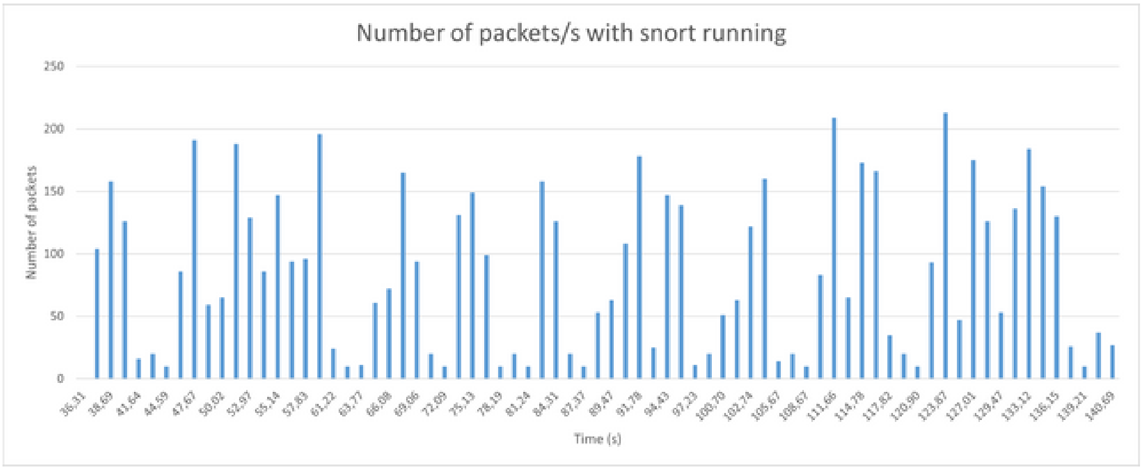
The application is listening on port `7777`. The server receive TCP request from other node of the network.

4. Please attach a graph of the traffic over time to your answers

Graph showing the packet rate over time obtained by sniffing from the Snort machine snort is not running.



Graph showing the packet rate over time obtained by sniffing from the Snort machine snort is running.



5. What does the "-Q" option do in Snort?

```
-Q # Enable inline mode operation.
```

Snort in inline mode creates a transparent bridge between two network segments. In this case allow the connection from the client to the server.

6. What does the "--daq nfq" option do in Snort?

```
--daq <type> # Select packet acquisition module (default is pcap).
```

On Ubuntu, you can run Snort two different ways in inline mode, with `AFPACKET` or with `NFQ`. `NFQ` lets you leverage the power of iptables to make routing decisions.

5. The request that the client sends the server is broken into four parts. What are these parts and what order does they appear in? How are these parts seperated in the request?

Let's consider the tcp stream saved in `./base/part_2/connection.pcap`. The 6th message is sent by the client to the server and, in the data field, contain the following line:

```
joe!:.:!password1!:.:!classified.txt!:.:!8en8J...
```

This is the "protocol" that the application use to send the data to the server. It is a primitive protocol in which the request is a plain string, each request has four fields that are separated by the substring `!:.:!`. The fields are, in order: the username of the client, the password, the requested file and some other data which appears to be random.

6. Is this is a secure way for the client to send requests to the server? Explain your answer.

It is not. Even accepting the "basic string custom protocol" used for the communication, it still lacks authentication and encryption so any malicious entity could eavesdrop the communication, tamper with it or impersonate other entities.

7. Can you recover one of the files sent by the server to a client? If so attach the file, a pcap the relevant packets and indicate which client this was sent to.

Yes, it is possible. In the attached `connection.pcap` we can observe it and in the `connection.pdf` we can see a copy of the exchanged file.

8. What rule did you use to secure the "classified" file?

```
reject tcp 100.1.200.0/24 ANY -> 100.1.10.10 7777 (msg: "Attempt to access classified content"; sid:1000001; content:"classified";)
```

This rule will prevent any host beloging to the `100.1.200.0/24` subnet to retrive the `classified` file.

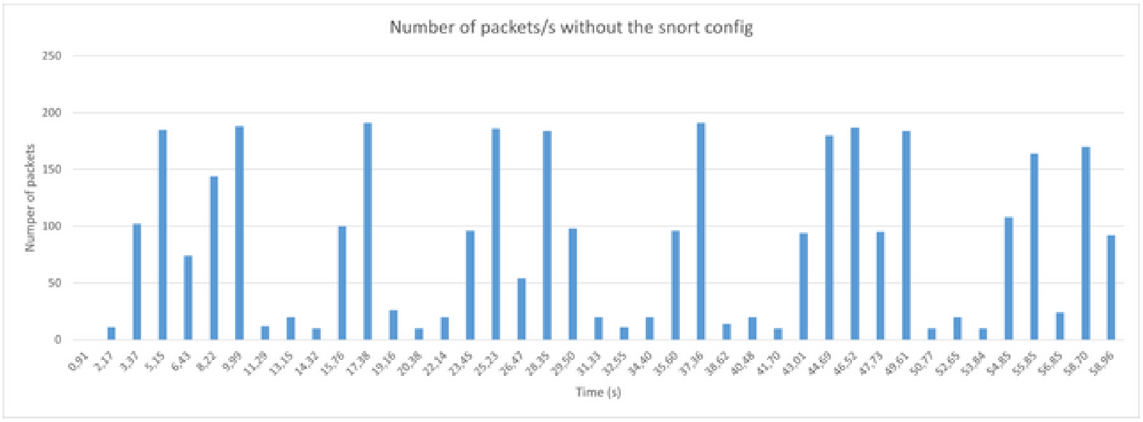
9. Capture and compare the network traffic for the server when filtering these results using your configuration file and when no file is used. Attach the graph showing packet rate over time for both of these cases to your submission.

To launch snort with the previosuly described rule type in the folder `./base/part_3/`:

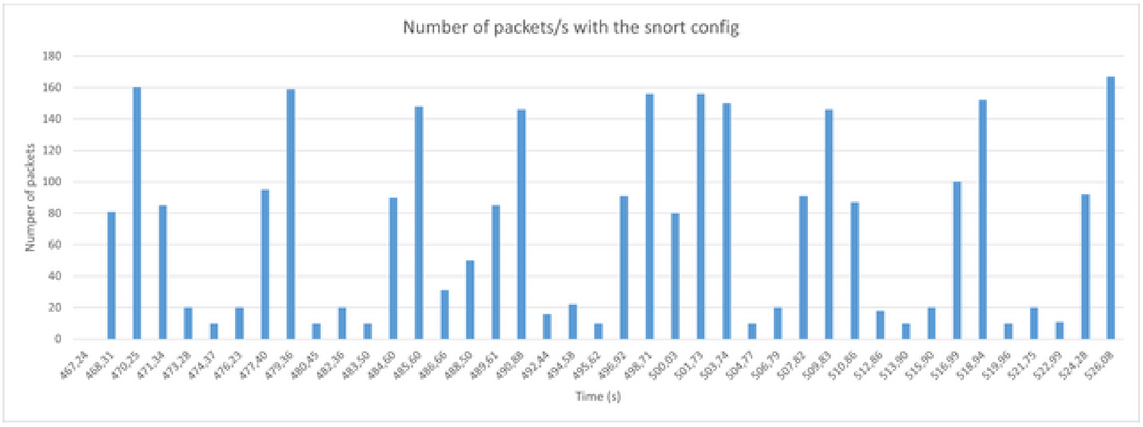
```
sudo snort --daq nfq -Q -c snort.config -l alerts
```

This will start snort with the rules specified in `snort.config`.

Graph showing the packet rate over time obtained by sniffing from the Snort machine on the interface connected with the router when no particular configuration is running. It is interesting to look at the TCP stream number 8 (`tcp.stream eq 8`), we can see how a request for a classified file is performed and completed by the outsider machine.



Graph showing the packet rate over time obtained by sniffing from the Snort machine on the interface connected with the router when the configuration made for preventing the leak of classified information is running. It is interesting to look at the TCP stream number 5 (`tcp.stream eq 5`), we can see see how the request for a classified file is performed by the outsider but rejected by the snort machine.



This analysis show how the snort rule are running as expected, blocking the request for the `.xml` file and the request from the `outsider` for the classified file.

10. Can you think of any others files or extensions that should be filtered against?

It depends on the purpose of the application. If it used just to retrieve `.txt` files then every other file extension should be blocked.

INTERMEDIATE TASK

To start the flood run the following command from the `client1`:

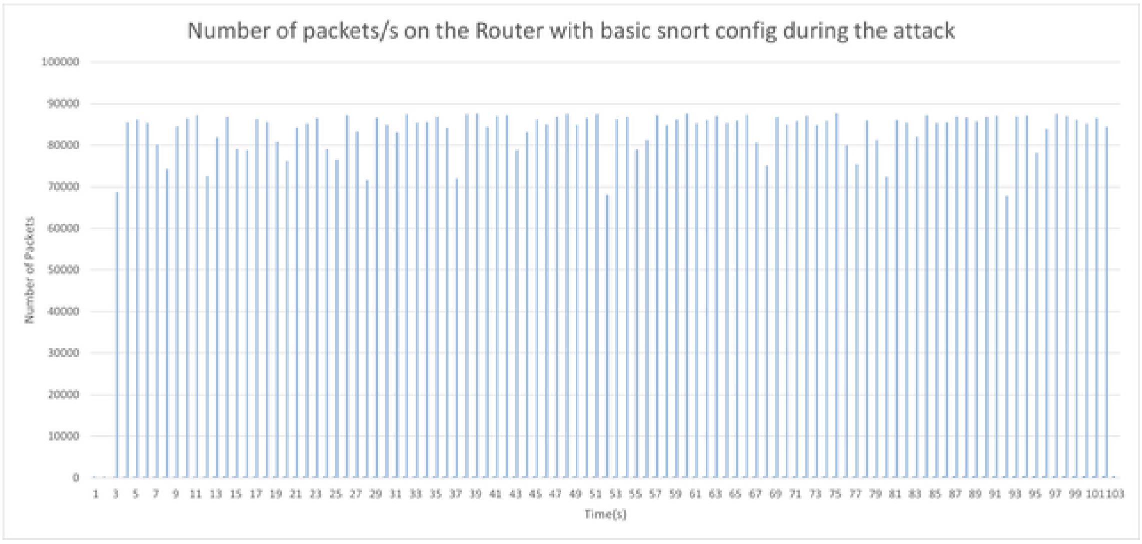
```
sudo timeout 100 flooder --dst 100.1.10.10 --highrate 100000 --srcmask 255.255.255.255
--src 100.1.5.10
```

This will generate for 100 seconds a flood of upd packets from the `client1` and directed to the server.

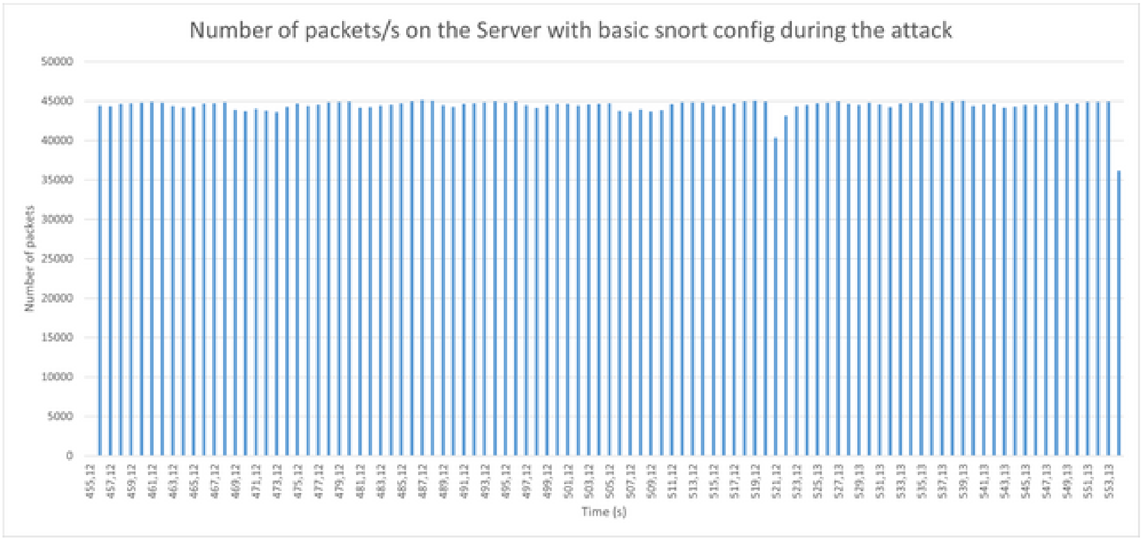
The snort configuration against this attack can be found in `./intermediate/snort.config`.

11. Collect the traffic at the server and the router when there is and is not an attack with rules in place that only guard against the attacks metioned in the basic exercises. Create a graph of this traffic over time.

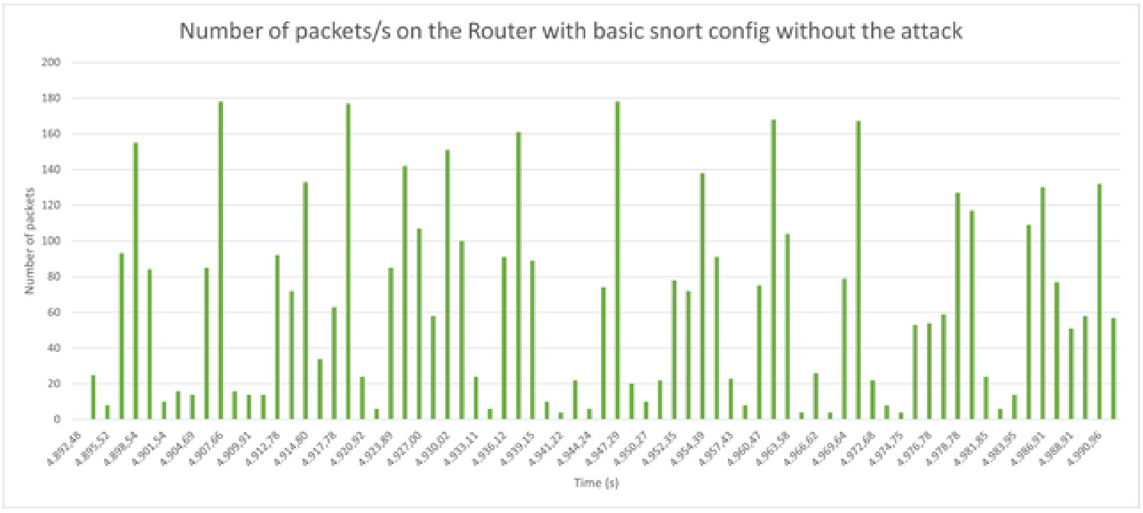
Graph showing the packet rate over time obtained by sniffing from the Router machine on the interface connected with the Snort machine when only the basic configuration is running and the client is performing the attack.



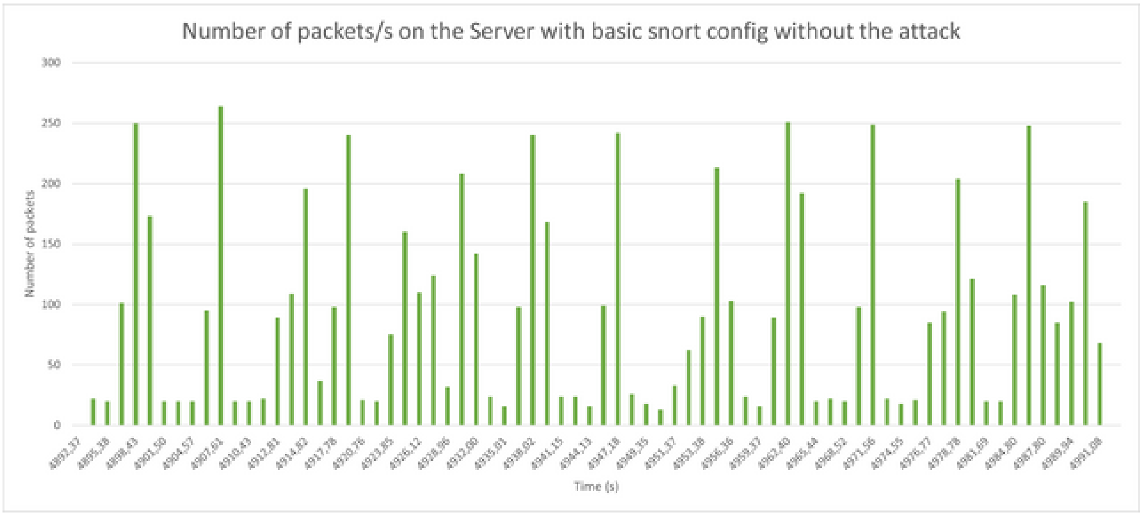
Graph showing the packet rate over time obtained by sniffing from the Server machine on the interface connected with the Snort machine when only the basic configuration is running and the client is performing the attack.



Graph showing the packet rate over time obtained by sniffing from the Router machine on the interface connected with the Snort machine when only the basic configuration is running and the client is not performing the attack.



Graph showing the packet rate over time obtained by sniffing from the Server machine on the interface connected with the Snort machine when only the basic configuration is running and the client is not performing the attack.

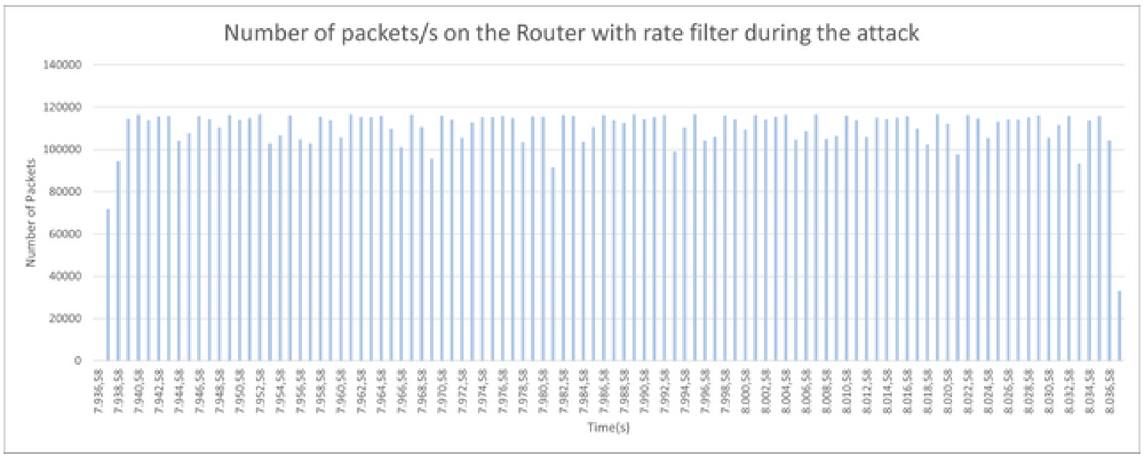


12. Repeat the previous step but now with the rate filtering rules enabled.

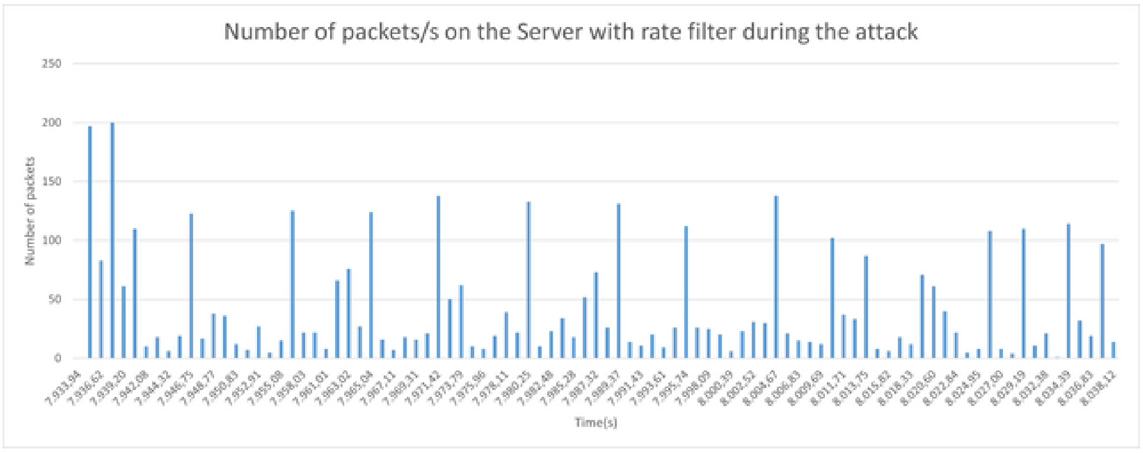
The rules used to apply the `rate_filter` to the traffic are the following:

```
pass udp 100.1.5.10 ANY -> 100.1.10.10 ANY (msg: "UDP SYN flood detected";
sid:1000003;)

rate_filter gen_id 1, sig_id 1000003, track by_rule, count 100, seconds 1, new_action
drop, timeout 20

event_filter gen_id 1, sig_id 1000003, track by_src, count 1, seconds 5, type limit
```

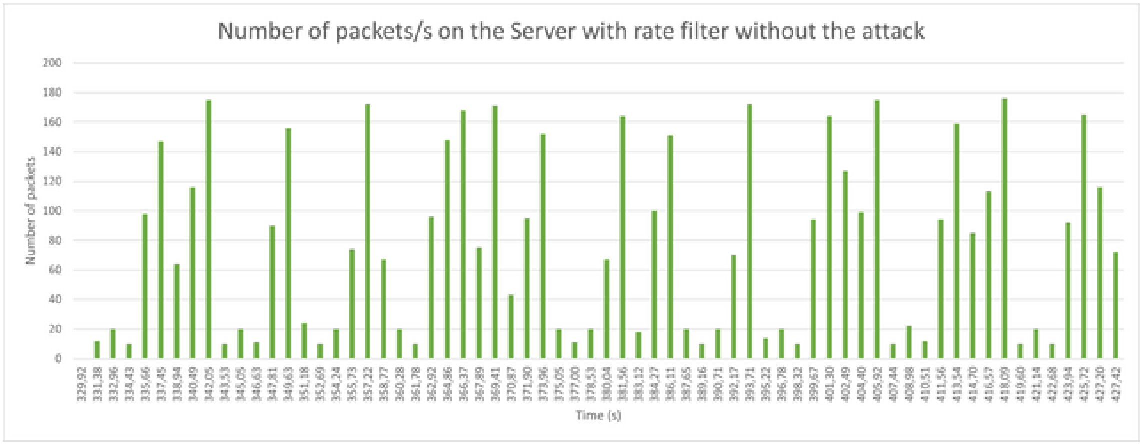
Graph showing the packet rate over time obtained by sniffing from the Router machine on the interface connected with the Snort machine when the rate filtering configuration is running and the client is performing the attack.



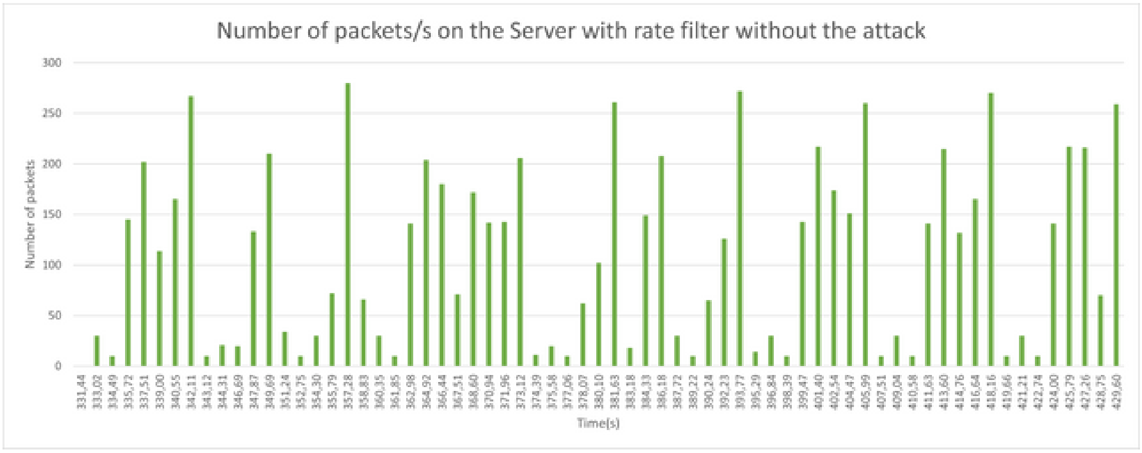
Graph showing the packet rate over time obtained by sniffing from the Server machine on the interface connected with the Snort machine when the rate filtering configuration is running and the client is performing the attack.



Graph showing the packet rate over time obtained by sniffing from the Router machine on the interface connected with the Snort machine when the rate filtering configuration is running and the client is not performing the attack.



Graph showing the packet rate over time obtained by sniffing from the Server machine on the interface connected with the Snort machine when the rate filtering configuration is running and the client is not performing the attack.



13. For a DOS attack should rate filtering rules be paired with event filtering rules?

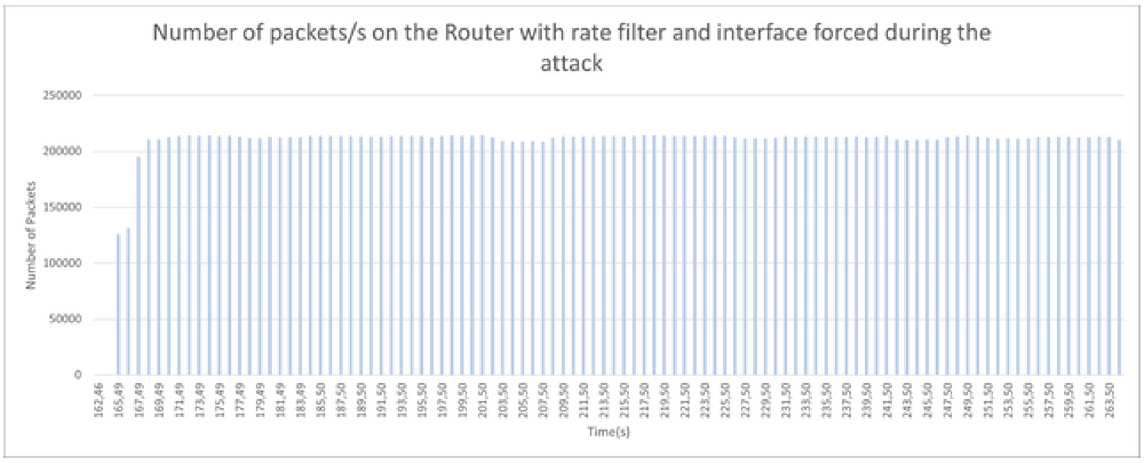
Yes, the event filtering rules are used to prevent a "spam" of alerts when a certain rule is often matched. During a dos attack the rate filtering rule will be matched many times, an event rule can prevent to have too many alerts.

14. Try changing the new action in the rate filter to "reject" instead of "drop". What does this do to the traffic and why do you think this is? Is this a good or a bad thing?

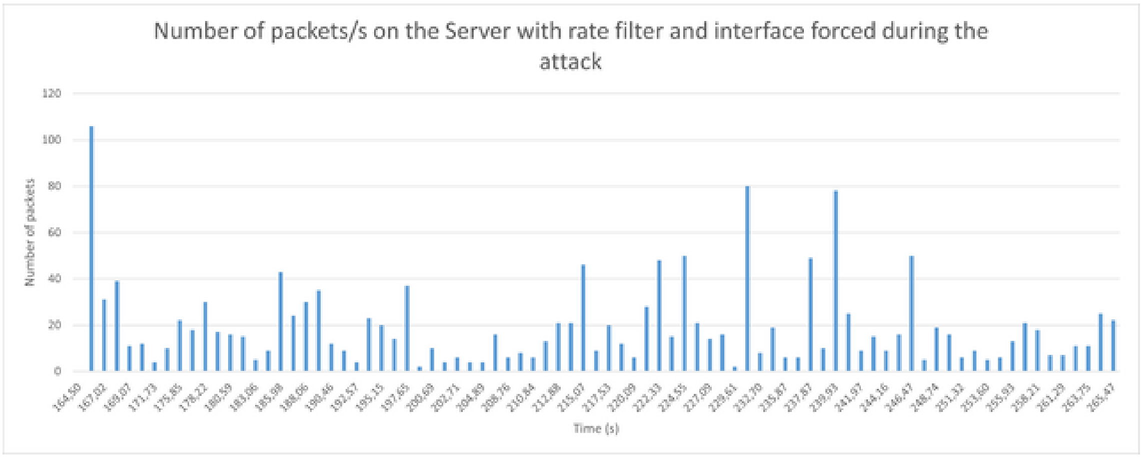
With the drop options the packets matching the rule will be simply droped while with the reject options an `RST` or an `ICMP port unreachable` message will be sent to the client. Using the drop options should be the best practice because responding does not have any purpose and those responses will be dropped by the attacker. Moreover the snort machine will use computational power and bandwith to send them.

15. Check which interface Snort connects to the router to using ifconfig. Once you have this information try the above test against while specifying that interface instead of using the default value. This argument will look something like --daq-var device=eth1. Did this change your results. If so attach a graph and explain the change occured.

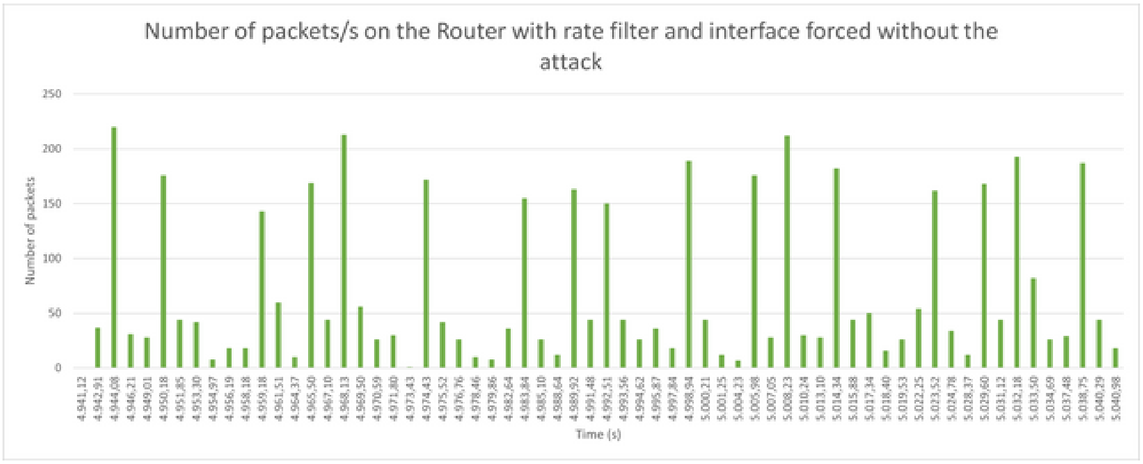
Graph showing the packet rate over time obtained by sniffing from the Router machine on the interface connected with the Snort machine when the rate filtering configuration is running, the interface is forced and the client is performing the attack.



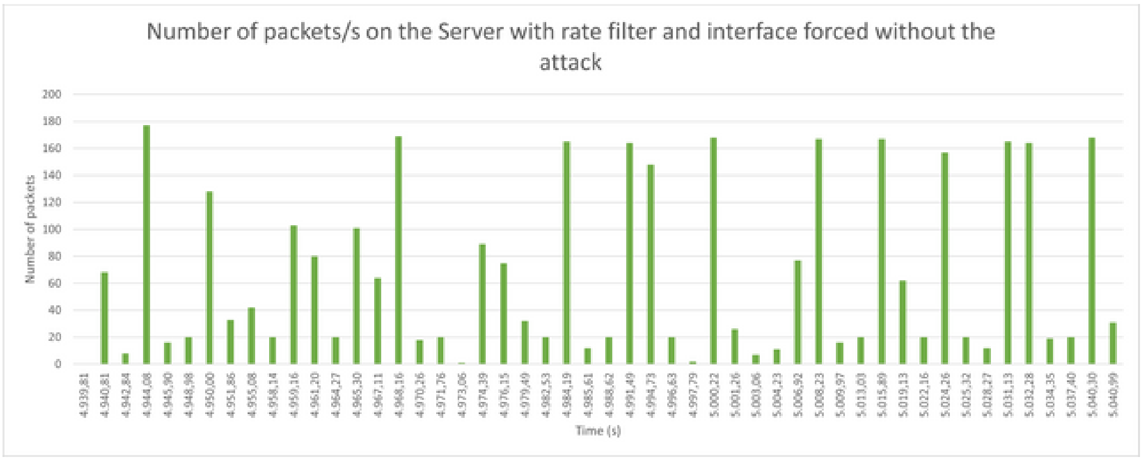
Graph showing the packet rate over time obtained by sniffing from the Server machine on the interface connected with the Snort machine when the rate filtering configuration is running, the interface is forced and the client is performing the attack.



Graph showing the packet rate over time obtained by sniffing from the Router machine on the interface connected with the Snort machine when the rate filtering configuration is running, the interface is forced and the client is not performing the attack.



Graph showing the packet rate over time obtained by sniffing from the Server machine on the interface connected with the Snort machine when the rate filtering configuration is running, the interface is forced and the client is not performing the attack.



16. Are there any changes you can make outside of Snort to help guard against this attack? If so what are they?

Other than Snort a DOS attack can be mitigate by setting up a good firewall or eventually relying on a load balancer.

17. Try running the FileClient program from client2 while this attack is underway. What happens when you have the various rulesets configured?

To run the client2 java application is first needed to stop the one which is running in background (`sudo ps -aux` then search for a running bash and kill the process). Then, to actually request a file type:

```
java -jar FileClient.jar candice monday server ducky.txt
```

Sometimes some request is lost because the link is overloaded, but sometimes the client should be able to receive the requested file.

18. Based on the traffic you analyzed what changes could be made to the network to enhance the security of communications coming from client1, client2 and outsider? What software packages would this require and where should these be installed? Would this cause problems for Snort?

We could implement and force the usage of TLS on the server, keeping the "basic protocol" intact. We could eventually implement an HTTP server instead and still secure it with TLS. Any software should be installed on the server. This will cause problems for snort because it will need to do DPI (Deep Packet Inspection) decrypting the TLS payloads before allowing TLS traffic to flow in and this is resource expensive.

20. How should the server be configured to prevent internal attacks? Would this require you to change your Snort configuration in any way?

The server cannot only rely on the protection offered by Snort. Forcing the internal traffic to be filtered by snort is a good practise to avoid any attack sent by an internal user but is still not enough. A good solution may be to separe the server from the local network and allow only the traffic coming from the snort machine (even better if is possible to directly connect the server to the snort machine). In this way it will became possible to also hide all the information about the server which are not strictly necessary to ensure its correct functioning. To do this we will need to change the Snort machine configuration because with this solution the request will be sent to its IP instead of the server one.

USEFUL COMMANDS

```
# Start snord using the snort.conf configuration file
sudo snort --daq nfq -Q -c snort.config -l alerts

# Start snort as before but force snort to listen on the interface connected with the
router
sudo snort --daq nfq -Q -c snort.config -l alerts --daq-var device=eth1

# Install the flooder
/share/education/TCPSYNFlood_USC_ISI/install-flooder

# Start the flooding
sudo timeout 100 flooder --dst 100.1.10.10 --highrate 100000 --srcmask 255.255.255.255
--src 100.1.5.10

# Create a tpdump
sudo tcpdump -i eth -s 0 -w name.pcap

# Get number of packets/s from a pcap file
sudo /share/education/SecuringLegacySystems_JHU/process.pl name.pcap

# Require a file from the server
java -jar FileClient.jar candice monday server ducky.txt

# Manually send UDP traffic to the server
sudo echo -n "hi" >/dev/udp/100.1.10.10/7777
```

```
# Force the internal machine to send the traffic to snort
sudo route add -host server gw snort

# Require all the .txt file
java -jar FileClient.jar bob password server export.txt & \
java -jar FileClient.jar joe password1 server classified.txt & \
java -jar FileClient.jar candice monday server ducky.txt & \
java -jar FileClient.jar billy thursday server users.txt

# Require the .xml file
java -jar FileClient.jar bob password server export.xml
```

ADVANCED TASK

Code Execution Vulnerability

The **FileServer.jar** executable contains a code execution vulnerability, to find it we must estract the file from the `/home/test` folder and decompile it using any Java decompiler. The decompiler will give us two **.java** files from which we can understand how the server works. By looking at the **ConnectionHandler.java** file we can see that around line 50 the method `java.lang.Runtime.exec` is called and the input is the name of the file requested by the client. An attacker can forge a message to exploit this.

22. What are the conditions required for this attack to take place?

The attack succeeds if the message is larger than 2000 bytes and contains at least an occurrence of a string that matches the following regular expression: `/z.{0,2}a.{0,2}q.{0,2}r`. At this point the command contained in the *fileName* field is executed. A PoC can be found in the attached *rce.java*.

23. Create a Snort rule to defend against this attack. You may want to be use pcre instead of content for this rule.

To detect this code execution using Snort we must implement a new rule to filter out the regex expression a/o the size of the message. The rule is the following:

`reject tcp any any -> 100.1.10.10 7777 (msg: "RCE exploiting attempt"; pcre: "/z.{0,2}a.{0,2}q.{0,2}r/"; sid:9000000)`

24. What effect does this rule have on legitimate traffic?

This regex could match even legitimate packets, bringing to higher proportion of network traffic generating false alerts and blocking legitimate requests to the server.

Defend Against ASCII Encoding

From the server source code we can see that it implements ASCII decoding, this feature could allow us to bypass our new snort rule since the regex does not match ASCII.

25. Were you able to bypass your existing rules because of this feature? If so what input strings did you use?

Yes. It is enough to change the string matching the regular expression with its corresponding ascii encoding, so `z.a.q.r` becomes `%122.%97.%113.%114` and it can bypass the rule we added before.

26. Can you think of a content rule to effectively defend against an attack that uses this feature? Would this affect legitimate traffic?

We should add one (or more) rule(s) to match the ascii encoding and eventually also other encoding that could bypass the original rule. Adding regex rules to match ASCII encoded string could give more false alerts and block more legitimate traffic that in question 24.

27. Snort includes support for user written preprocessors that can render data for Snort's other rules. How would the use of a preprocessor help with this task?

We could use a preprocessor to decode any encoding present in the packets before it is processed with the Snort rules, so that the analyzed content will be identical to the one received by the server.

28. Write a preprocessor to help with this task. Please attach all of the functions you used and the snort.config file that called the preprocessor.

The attached **spp_example.c** source file contains the sample preprocessor extracted from the exercise with some modification to decode ASCII according to the given specifications.

To install the preprocessor just follow the Deterlab instruction substituting the **spp_example.c** file with our modified version.