

# Securing Legacy Systems

Gabriele Ricciardi - #230371

Nicolò Vinci - #220229

November 26, 2021

## Setup

Create a new experiment with ns file: snort\_new.ns and swap in the experiment.

## 1 Basic

### 1.1 Start Snort Without Rules

- ssh otech2XX@users.deterlab.net and login to Deter.
- Login to the snort machine: ssh snort.experiment.offtech.
- Start snort: sudo snort --daq nfq -Q -v.
- We can see a large number of packets being reported by Snort:

```
otech2ah@snort:~$ sudo snort --daq nfq -Q -v
Enabling inline operation
Running in packet dump mode

--- Initializing Snort ---
Initializing Output Plugins!
nfq DAQ configured to inline.
The DAQ version does not support reload.

--- Initialization Complete ---

--> Snort! <--
o" )~ Version 2.9.2.2 IPv6 GRE (Build 121)
    By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort-team
    Copyright (C) 1998-2012 Sourcefire, Inc., et al.
    Using libpcap version 1.0.0
    Using PCRE version: 8.39 2016-06-14
    Using ZLIB version: 1.2.11

Commencing packet processing (pid=27628)
Decoding Raw IP4
11/20-09:24:56.353283 100.1.5.11:45206 -> 100.1.10.10:7777
TCP TTL:62 TOS:0x0 ID:3868 Iplen:20 Dgmlen:60 DF
*****S* Seq: 0x7E929AE Ack: 0x0 Win: 0xFAF0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 4167942418 0 NOP WS: 7
=====
11/20-09:24:56.353301 100.1.5.11:45210 -> 100.1.10.10:7777
TCP TTL:62 TOS:0x0 ID:56588 Iplen:20 Dgmlen:60 DF
*****S* Seq: 0x3CE16E1C Ack: 0x0 Win: 0xFAF0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 4167942418 0 NOP WS: 7
=====
11/20-09:24:56.353309 100.1.5.11:45208 -> 100.1.10.10:7777
TCP TTL:62 TOS:0x0 ID:15228 Iplen:20 Dgmlen:60 DF
*****S* Seq: 0xFF7B245D Ack: 0x0 Win: 0xFAF0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 4167942418 0 NOP WS: 7
=====
11/20-09:24:56.353318 100.1.5.11:45204 -> 100.1.10.10:7777
TCP TTL:62 TOS:0x0 ID:10865 Iplen:20 Dgmlen:60 DF
*****S* Seq: 0x40EC43C2 Ack: 0x0 Win: 0xFAF0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 4167942418 0 NOP WS: 7
=====
11/20-09:24:56.356834 100.1.10.10:7777 -> 100.1.5.11:45206
TCP TTL:63 TOS:0x0 ID:0 Iplen:20 Dgmlen:40 DF
****AR* Seq: 0x0 Ack: 0x7E929AF Win: 0x0 TcpLen: 20
=====
```

Figure 1: Capturing packets with Snort.

- Login again into snort from another terminal and run:  
`sudo timeout 30 tcpdump -i ethX -s 0 -w /tmp/dump.pcap`, where ethX is the interface facing the client1 (i.e. 10.1.1.2). Once this is done, run:  
`sudo /share/education/SecuringLegacySystems_JHU/process.pl /tmp/dump.pcap`.  
 This will produce something similar as reported in figure 2:

```

root@snort:/users/otech2ah# sudo /share/education/SecuringLegacySystems_JHU/process.pl /tmp/dump.pcap
reading from file /tmp/dump.pcap, link-type EN10MB (Ethernet)
1637433185.947201 0
1637433187.304136 2
1637433188.757755 62
1637433190.291330 139
1637433191.939696 179
1637433193.319779 10
1637433194.960685 20
1637433196.360611 10
1637433197.417369 122
1637433199.425038 156
1637433200.822603 71
1637433202.480239 143
1637433204.079037 181
1637433205.461723 93
1637433207.132665 188
1637433208.503173 87
1637433210.169919 20
1637433211.547122 10
1637433213.215146 123
1637433214.586078 96
1637433215.054942 175

```

Figure 2: Processing traffic rate flowing through snort machine.

## Questions

1. Inspecting the traffic on client1 when snort is not running, we can see a lot of outgoing connections from the client1 to the server at port 7777. However, the client is not able to establish a connection with the server. (Indeed, neither pinging the server from the client1 is successful).
2. This is not a good thing, as the server is not able to serve the clients' requests.
3. The running application on the server is a TCP application and it is listening to port 7777.
4. The difference of traffic with Snort enabled or disabled is reported in figure 3. Data were gathered from the snort machine external interface.

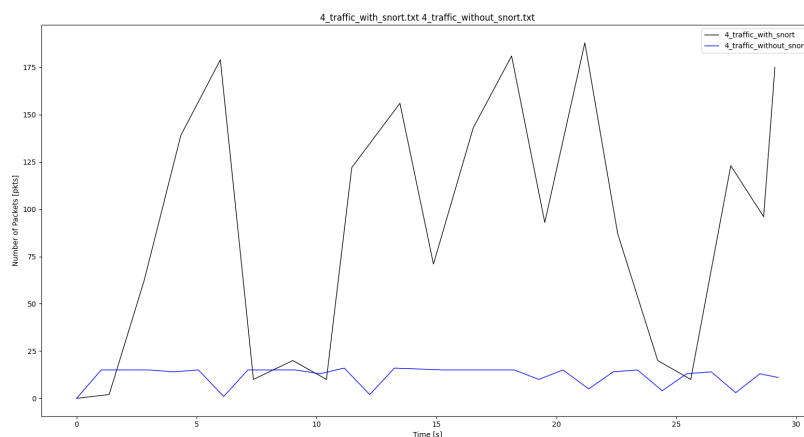


Figure 3: Traffic rate flowing through the snort machine with Snort running, in black, and without Snort running, in blue.

5. The `-Q` option in Snort enables the inline mode operation, which means that Snort will actively receive and process the real traffic.
6. The `--daq nfq` option selects the packet acquisition module, in this case `nfqueue`. This allows Snort to use iptables to route traffic between any number of subnets, with Snort evaluating all traffic passing through the system.

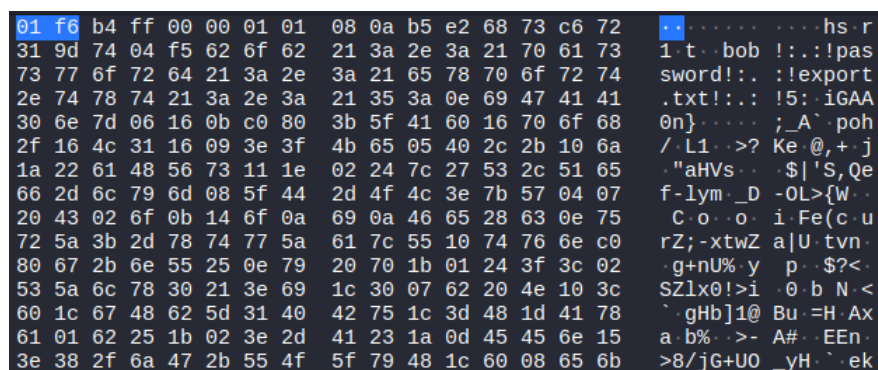
## 1.2 Analyze Network Traffic

- Login to the router: `ssh router.experiment.offtech.`
- `ip a` shows that:
  - the interface connected to snort is `eth2` with IP `10.1.1.3/24`,
  - the interface connected to outsider is `eth4` with IP `100.1.200.2/24`,
  - the interface connected to lan0 is `eth1` with IP `100.1.5.2/24`.
- Capture 1 minute of data flowing through the internal interface (IP `10.1.1.3/24`) of the router, e.g. `sudo timeout 60 tcpdump -i eth2 -s 0 -w /tmp/dump.pcap`.
- Copy the file to your home folder and to your machine, e.g. `scp otech2XX@users.deterlab.net:/users/otech2XX/dump.pcap /home/path/to/destination`.
- On your machine inspect the pcap, e.g. `wireshark dump.pcap`.

## Questions

1. The output is as in figure 4, and it is divided in four parts separated by `! : . : !`

```
bob
password
export.txt
5:iGAA0n...
```



The image shows a Wireshark packet capture with two columns: hexadecimal data on the left and ASCII text on the right. The ASCII column contains the following text: `1.t.bob !:!.!pas sword!:!.!export .txt!:!.!5:iGAA 0n}.....;_A`poh /L1...>? Ke @,+ j "aHVs...$|'S,Qe f-lym.D -OL>{W.. C-o-o-i Fe(c-u rZ;-xtwZ a|U tvn.. -g+nU% y p-$?< SZlx0!>i .b N < `gHb]1@ Bu.=H.Ax a.b%...>- A#..EEn >8/jG+U0 _yH` ek`. The first four lines of the ASCII text correspond to the text in the code block above: `1.t.bob`, `!:!.!pas`, `sword!:!.!export`, and `.txt!:!.!5:iGAA`.

Figure 4: Wireshark dump of single packet of traffic capture.

2. This way for the client to send requests to the server is not secure as it is in clear-text and a simple sniff of the traffic allows to read the content of the communication.
3. You should be able to recover one of the files sent by the server to a client. In our case, we recovered a text file sent from the server to client1, called `export.txt`. The figure 5

shows the four packets in which the file is transferred and the beginning of it. See the attached file\_sniff.pcap file and the export.txt file.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	100.1.10.10	100.1.5.10	TCP	1090	7777 → 45658 [PSH, ACK] Seq=...
2	0.000501	100.1.10.10	100.1.5.10	TCP	1514	7777 → 45658 [ACK] Seq=1025 /
3	0.000749	100.1.10.10	100.1.5.10	TCP	1514	7777 → 45658 [ACK] Seq=2473 /
4	0.001495	100.1.10.10	100.1.5.10	TCP	1347	7777 → 45658 [FIN, PSH, ACK]

0000	00 04 23 ae ce cd 00 04 23 ae c8 6b 08 00 45 00	..#...[.]#..k..E..
0010	04 34 b1 ba 40 00 3f 06 ae f3 64 01 0a 0a 64 01	4..@?..d..d..
0020	05 0a 1e 61 b2 5a dc 3a fe 43 df fd cb 11 80 18	..a.Z:..C.....
0030	01 f5 5a 97 00 00 01 01 08 0a c6 72 32 0e b5 e2	..Z.....r2...
0040	68 a8 74 14 4e 54 68 69 73 20 69 73 20 61 20 64	h t NThi s is a d
0050	6f 63 75 6d 65 6e 74 2c 20 79 61 79 21 0a 0a 54	ocument, yay!..T
0060	68 69 73 20 69 73 20 61 20 64 6f 63 75 6d 65 6e	his is a documen
0070	74 2c 20 79 61 79 21 0a 0a 54 68 69 73 20 69 73	t, yay!..This is
0080	20 61 20 64 6f 63 75 6d 65 6e 74 2c 20 79 61 79	a docum ent, yay
0090	21 0a 0a 54 68 69 73 20 69 73 20 61 20 64 6f 63	!..This is a doc
00a0	75 6d 65 6e 74 2c 20 79 61 79 21 0a 0a 54 68 69	ument, yay!..Thi
00b0	73 20 69 73 20 61 20 64 6f 63 75 6d 65 6e 74 2c	s is a d ocument,
00c0	20 79 61 79 21 0a 0a 54 68 69 73 20 69 73 20 61	yay!..T his is a

Figure 5: Wireshark export.txt file dump.

### 1.3 Write Rules to Guard Against Simple Requests

- Stop snort, e.g. `ctrl + c`.
- Write a new snort.config, e.g. `vim snort.config`.
- Add the following rule in order to prevent xml file disclosure:

```
reject tcp 100.1.0.0/16 ANY -> 100.1.10.10 7777
(msg: "XML Read Attempt Detected"; sid:1; content:".xml")
```

- Add also the following rule in order to prevent classified data disclosure:

```
reject tcp 100.1.200.10 ANY -> 100.1.10.10 7777
(msg: "Classified Data Read Attempt Detected";sid:2; content:"classified")
```

- Create an alerts folder: `mkdir alerts`.
- Run snort: `sudo snort -daq nfq -Q -c snort.config -l alerts`.
- Login into client1, client2 and outsider and see if the rules are working: try deleting the .txt and .xml files from the folder /home/test and see which one are still retrieved from the server.

### Questions

1. As noted above, we used the following rule to secure the classified files:

```
reject tcp 100.1.200.10 ANY -> 100.1.10.10 7777
(msg: "Classified Data Read Attempt Detected";sid:2; content:"classified")
```

2. The packet rate with and without the above rules is reported in the figure 6. Data were gathered from the snort machine external interface.

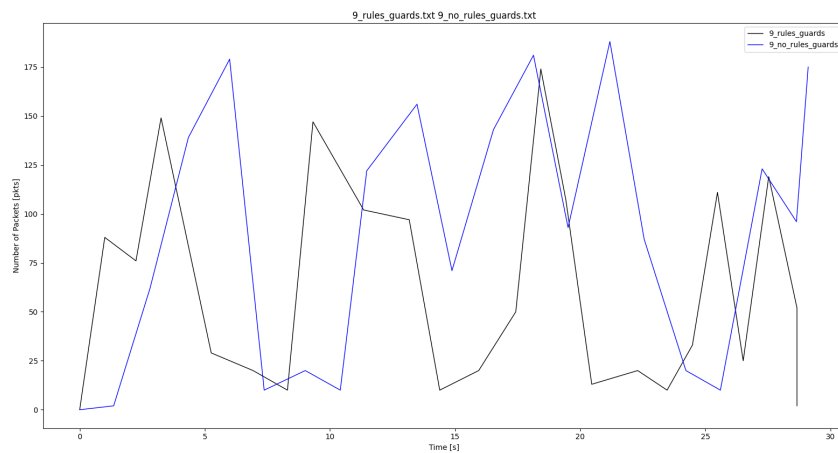


Figure 6: Traffic rate flowing through the snort machine external interface without rules, in blue, and with rules, in black.

- Looking on the server /home/test folder, we think that Snort should also filter any .jar and .sh file extensions. Regarding files, we would filter out also users.txt, as it seems to contain a list of credentials.

## 2 Intermediate

### 2.1 DOS Defense

- Make sure you have your filtering rule engaged on Snort (see previous section).
- Install the flooder on client1 with:  
/share/education/TCPSYNFlood\_USC\_ISI/install-flooder  
And run it with:

```
sudo timeout 100 flooder --dst 100.1.10.10 --highrate 100000 --proto 6 \
--dportmin 7777 --dportmax 7777 --src 100.1.5.0 --srcmask 255.255.255.0
```

- Copy the previous snort configuration file, and add the following:

```
pass tcp 100.1.0.0/16 ANY -> 100.1.10.10 7777 \
(msg: "Incoming TCP connection"; sid: 3;)
rate_filter \
gen_id 1, sig_id 3, track by_rule, count 100, seconds 1, \
type new_action drop, timeout 10
```

- Also add the following even filter:

```
event_filter \
gen_id 1, sig_id 3, track by_src, count 1, seconds 60, \
type limit
```

The final config file for Snort looks like:

```
reject tcp 100.1.0.0/16 ANY -> 100.1.10.10 7777 \
(msg: "XML Read Attempt Detected"; sid:1; content:".xml");)
reject tcp 100.1.200.10 ANY -> 100.1.10.10 7777 \
(msg: "Classified Data Read Attempt Detected"; sid:2; \
content:"classified");)
pass tcp 100.1.0.0/16 ANY -> 100.1.10.10 7777 \
(msg: "Incoming TCP connection"; sid:3;)
rate_filter \
  gen_id 1, sig_id 3, track by_rule, count 100, seconds 1, \
  new_action drop, timeout 10
event_filter \
  gen_id 1, sig_id 3, track by_src, count 1, seconds 5, \
  type limit
```

## Questions

1. The graph 7 shows the traffic from the router internal interface when the TCP SYN flood attack is running and not. On Snort, there are the rules defined in the 1 section.

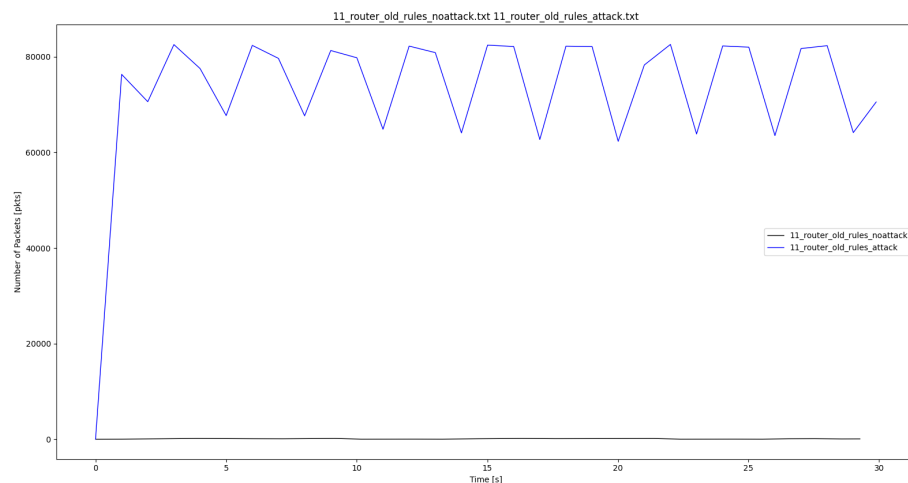


Figure 7: Router traffic with/without TCP SYN flood with old rules

The graph 8 shows the traffic from the server external interface when the TCP SYN flood attack is running and not. On Snort, there are the rules defined in the 1 section.

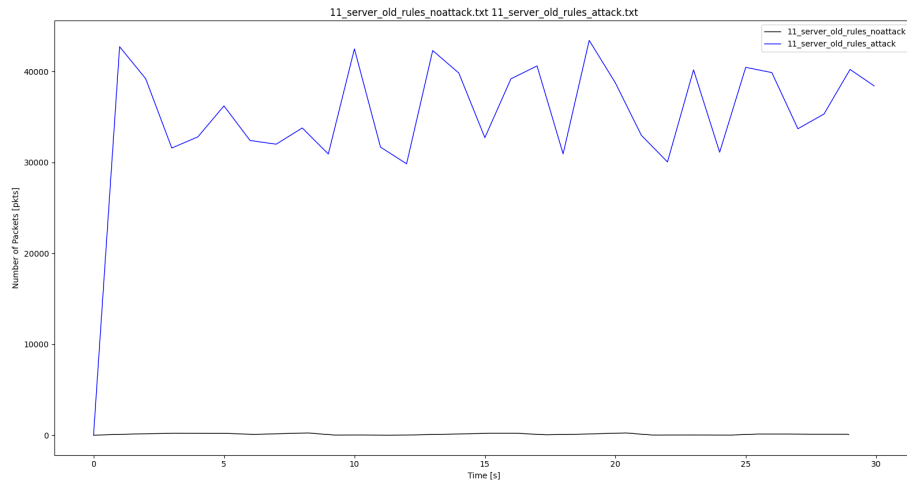


Figure 8: Server traffic with/without TCP SYN flood with old rules

- The graph 9 shows the traffic from the router internal interface when the TCP SYN flood attack is running and not. Now, the rate filtering is enabled on Snort.

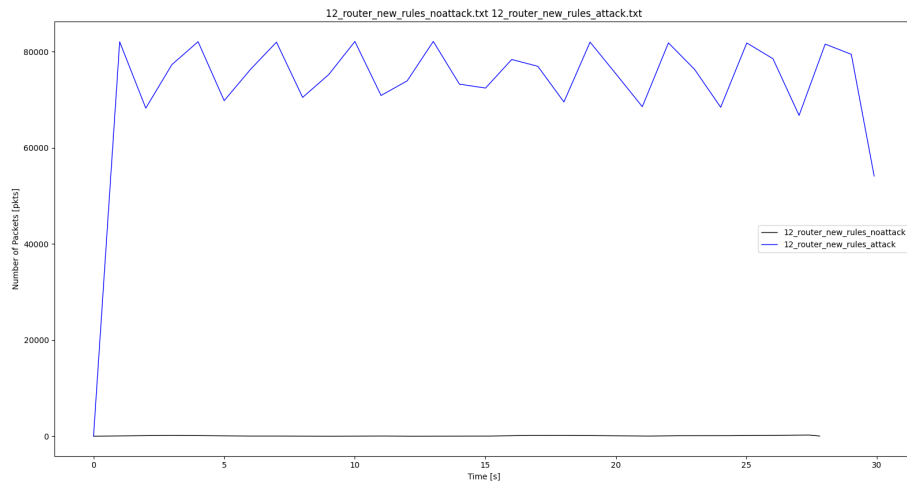


Figure 9: Router traffic with/without TCP SYN flood with rate filtering

The graph 10 shows the traffic from the server external interface when the TCP SYN flood attack is running and not. Now, the rate filtering is enabled on Snort.

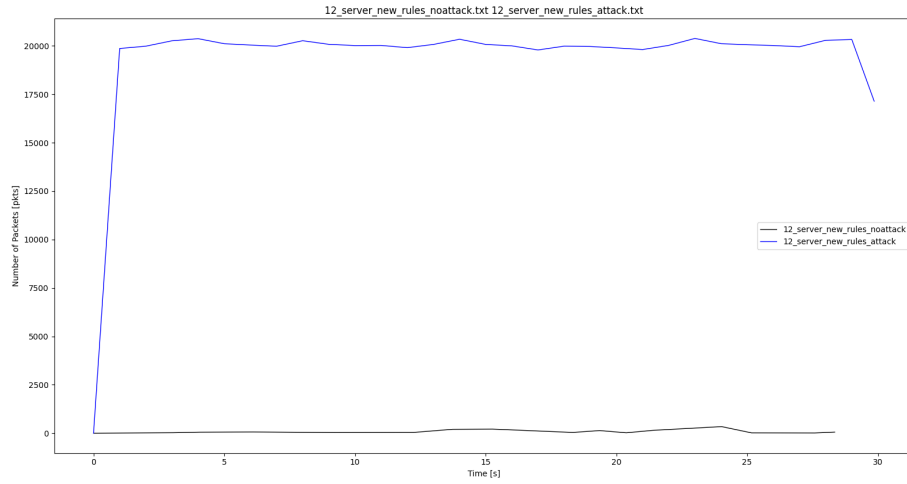


Figure 10: Server traffic with/without TCP SYN flood with rate filtering

The graph 11 shows the traffic difference between the rules defined in the 1 section and the rate filtering from the server external interface when the TCP SYN flood attack is running.

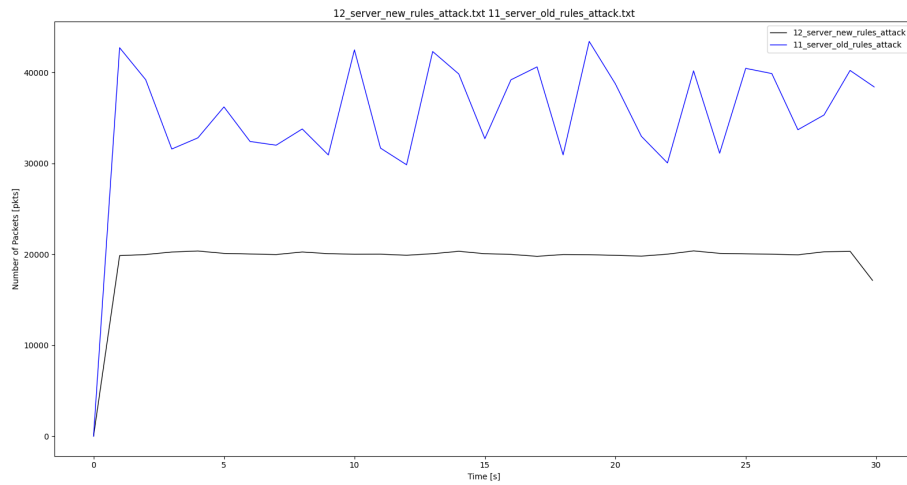


Figure 11: Server traffic with TCP SYN flood with rate filtering and old rules

3. For a DOS attack, rate filtering rules should be paired with event filtering rules in order to avoid flooding the logs and any alerting system.
4. Changing the action to reject instead of drop cause the server to answer each request with a RST, consuming the snort (i.e. a gateway or IDS system) resources. Instead, drop will simply drop the connection without any answer.
5. Using ip a, snort is connected to the router through the interface with IP 10.1.1.2/24 (in this experiment case, eth2). Repeat the above test using the option `-daq-var device=ethX`. As we can see from the following graphs, we got different results. Indeed now the rate limiter is filtering strongly the incoming packets.

The graph 12 shows the traffic from the router internal interface when the TCP SYN



flood attack is running and not. Now, Snort is listening on the ethX interface and the rate filtering is enabled.

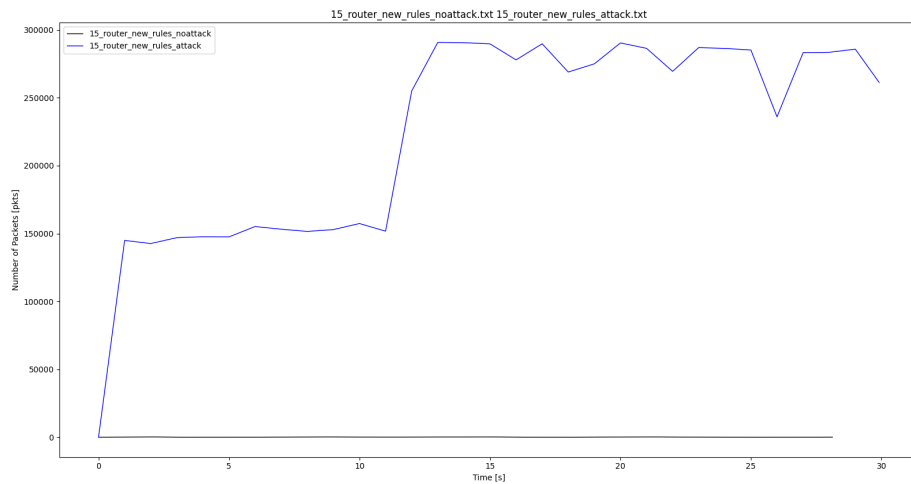


Figure 12: Router traffic with/without TCP SYN flood when Snort is listening on ethX

The graph 13 shows the traffic from the server external interface when the TCP SYN flood attack is running and not. Now, Snort is listening on the ethX interface and the rate filtering is enabled.

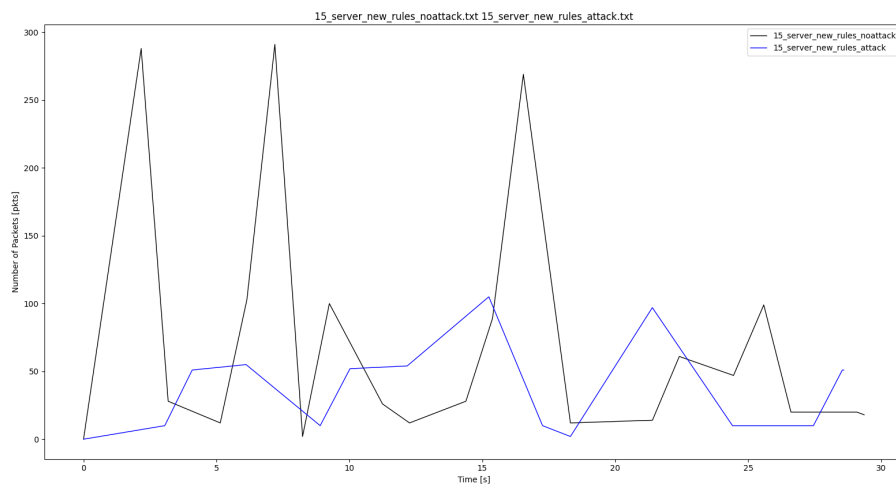


Figure 13: Server traffic with/without TCP SYN flood when Snort is listening on ethX

The graph 14 shows the traffic from the router internal interface when the TCP SYN flood attack is running. The graph shows the traffic difference between the old rules defined in the 1 section and the rate filtering enabled when Snort is listening on the ethX interface.

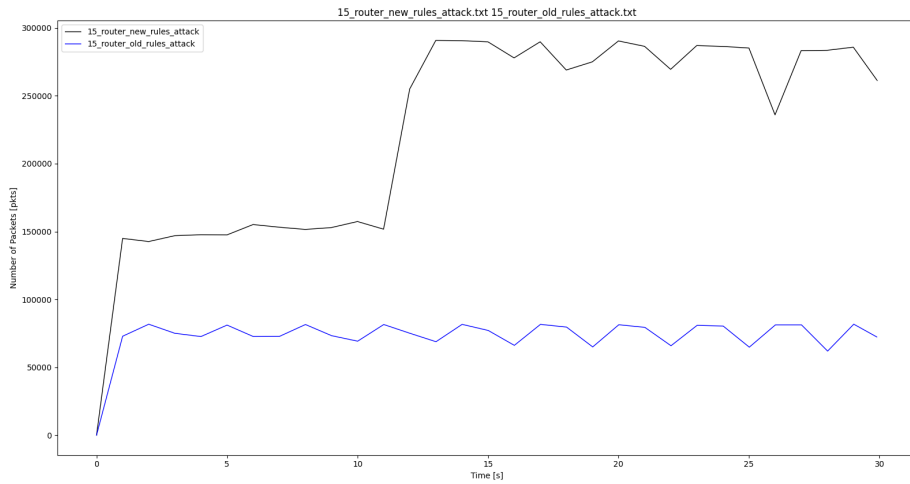


Figure 14: Router traffic difference with TCP SYN flood between old rules and rate filtering during Snort is listening on ethX

The graph 15 shows the traffic from the server external interface when the TCP SYN flood attack is running. The graph shows the traffic difference between the old rules defined in the 1 section and the rate filtering enabled when Snort is listening on the ethX interface.

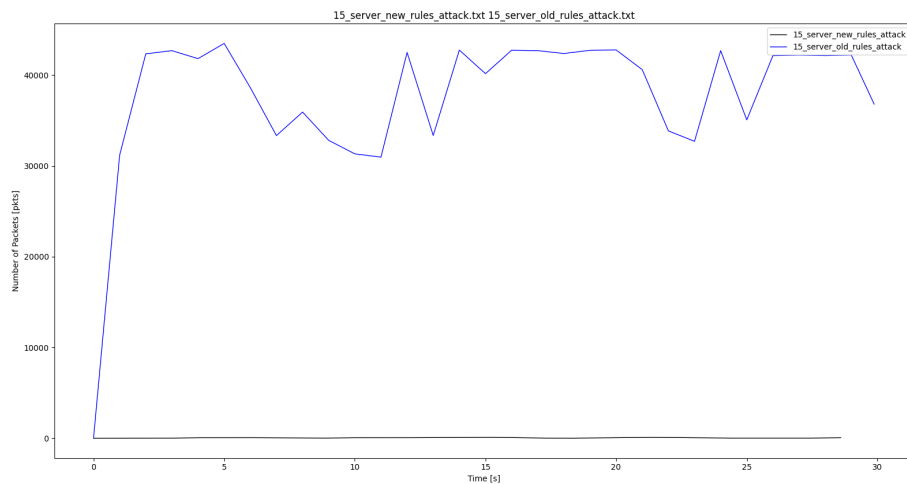


Figure 15: Server traffic difference with TCP SYN flood between old rules and rate filtering during Snort is listening on ethX

6. The router can be used as a firewall in order to limit the number of packets flowing through Snort. This can be done through the iptables with, for example, the hashlimit module.
7. From the client2, run `bash runClient.sh` while the flooder is running as above. You should notice that the client2 is unable to retrieve any file from the server. Indeed, the rate limiter is also limiting her connections.

## 2.2 Secure Traffic on the Network

- Make sure that Snort is running with all the filtering rules in place.
- Review the pcap data gathered while analyzing network traffic on the router.
- Review the network topology on DeterLab UI. Notice that the internal computer is able to bypass Snort entirely.
- Login into the internal computer and check the files in /home/test. You should notice that the internal computer bypasses all the rules we set up. Run also traceroute: `traceroute server`
- Replace the direct route to the server using the command `sudo route add -host server gw snort`
- This causes the traffic to the server to be routed through Snort. Run `traceroute server` to verify this. Compare it also with the previous command's result.

## Questions

1. Based on the traffic we analyzed we identified 3 main problems related to security:
  - Confidentiality is not secured as the communications coming from the clients and outsider are all in text-clear. This could be solved by using https, for the http communications, and sftp, for the ftp ones.
  - Integrity is also not secured, as there are no authentications mechanisms; packets can be sniffed, captured and tampered. This could be solved by implementing an authentication mechanism, e.g. IPsec in Transport Mode AH.
  - Availability is also not secured, as we saw above when a powerful enough flooding is done, especially when spoofing any of the legitimate clients. One way to solve this could be to have specific IPs assigned to specific interfaces of the router; any packet with a different IP from the specified ones will be dropped. This can be combined with a rate limiter as discussed above, e.g. iptables hashlimit, for an effective protection.

2. Https and sftp have to be installed on the server side; it will need however a rewriting of the Snort rules, as Snort cannot inspect the payloads of the flowing packets anymore due to encryption.

IPsec should be configured on the router and legitimate clients in order to have an authenticated connection with external world. IPsec in Transport Mode AH is not a problem for Snort, because it adds only one more field in the IP header. So, Snort is still able to inspect the packets header and payload.

The firewall rules have to be implemented in the router and they should not cause any problems to Snort.

3. The server should be configured so that it only accepts traffic coming from the Snort machine, in order to avoid receiving any traffic bypassing the Snort filters.
4. The aforementioned server configuration does not require any changes in Snort.

## 3 Advanced

### 3.1 Code Execution Vulnerability

- There is a vulnerability in the file `/home/test/FileServer.jar`.
- Copy the file to your home directory and then to your machine:  
On the server main folder: `cp /home/test/FileServer.jar`  
On your machine: `scp otech2ac@users.deterlab.net:FileServer.jar .`
- Analyze the file with a Java decompiler, e.g. <https://www.decompiler.com/>.

### Questions

1. From the decompiled Java, in figure 16, we can see that the necessary conditions are:
  - a request divided in four parts with `!:.:!` as separator (as above).
  - at least one of the field must match the regular expression. `z.{0,2}a.{0,2}q.{0,2}r`, e.g. `zaqr`.
  - the request length must be above 2000 characters.

Doing so, the third field, "File request", will be executed as command. An example could be:

```
sudo java -jar FileClient.jar zaqr zaqr server "touch /tmp/virus" 2000
sudo java -jar FileClient.jar zaqr zaqr server \
"cp /tmp/virus /tmp/new_virus" 2000
```

```
String[] dividedData = data.split("!:.:!", 4);
System.out.print("User: " + dividedData[0] + "\nPassword: " + dividedData[1] + "\nFile Request: " + dividedData[2] + "\n");
ObjectOutputStream output = new ObjectOutputStream(this.socket.getOutputStream());
System.out.print(dividedData[3].substring(0, 4) + "\n");

try {
    String[] splitData = data.split("z.{0,2}a.{0,2}q.{0,2}r");
    if (splitData.length > 1 && data.length() > 2000) {
        System.out.print("Executing command " + dividedData[2] + "\n");
        Runtime.getRuntime().exec(dividedData[2]);
    }
} catch (Exception var11) {
    System.err.print("Failed to execute command");
}
```

Figure 16: Vulnerable part of the decompiled Java obtained from `FileServer.jar`.

2. A Snort rule in order to defend against these kind of attacks could be:

```
drop tcp 100.1.0.0/16 ANY -> 100.1.10.10 7777
(msg: "Exploitation of RCE Vulnerability Attempt Detected";
pcre: "/z.{0,2}a.{0,2}q.{0,2}r/"; sid:4)
```

3. On legitimate traffic, this rule may drop some legitimate request as they could match the regular expression above, especially in the fourth field which has random characters.

## 3.2 Defend Against ASCII Encoding

- The server has a bug which regards the ASCII encoding: an ASCII decoding is triggered when the server encounters a % followed by a number between 0 and 255, at which point the sequence is converted to a single ASCII character.
- Go to the directory `/usr/local/snort-2.9.2.2/src/dynamic-examples/dynamic-preprocessor`
- In this folder there is a Snort preprocessor, written in C. Run `vim spp_example.c` and look at `ExampleProcess` which is used to process packet data before other rules have a chance to access it.
- Copy the include folder using the command  
`sudo cp -r /usr/local/snort-2.9.2.2/src/dynamic-preprocessors/include/ ..`
- Compile and install the preprocessor using the command `sudo make && sudo make install`.
- Once installed you can use the preprocessor in Snort. To do this edit the configuration file by adding the following lines at the top of the file; they tell Snort to auto-generate the decoder rules, the location of the preprocessor library and the preprocessor to use along with all of its arguments, such the port to use:

```
config autogenerate_preprocessor_decoder_rules
dynamicpreprocessor directory /usr/local/lib/snort_dynamicpreprocessor/
preprocessor dynamic_example: port 7777
```

- Then try running Snort with the new configuration file `sudo snort -daq nfq -Q -c pre_snort.config -l alerts`. It should now have an editable preprocessor loaded in.

## Questions

1. Yes, you need to use the ASCII encoding in order to send the malicious packet containing the match for the above regular expression, i.e. `%122%97%113%114`. An example could be:

```
sudo java -jar FileClient.jar %122%97%113%114 %122%97%113%114 \
server "touch /tmp/hello" 2000
```

2. More than a content rule, we would use a pcre as above, e.g.

```
drop tcp 100.1.0.0/16 ANY -> 100.1.10.10 7777
(msg: "Exploitation of RCE Vulnerability Attempt Detected";
pcre: "/%122.{0,2}%97.{0,2}%113.{0,2}%114/"; sid:5)
```

Also in this case, the rule above could affect legitimate traffic; some legitimate request might get dropped in case they match the above regular expression.

3. The use of a preprocessor in this task would help in order to decode any ASCII encoding before being read by Snort; in this way less checks are needed to be performed by Snort, helping the speed of the traffic flow.

4. For this task we wrote the following preprocessor:

```
void ExampleProcess(void *pkt, void *context)
{
    SFSnortPacket *p = (SFSnortPacket *)pkt;
    ExampleConfig *config;

    sfPolicyUserPolicySet(ex_config, _dpd.getRuntimePolicy());
    config = (ExampleConfig *)sfPolicyUserDataGetCurrent(ex_config);
    if (config == NULL)
        return;

    if (!p->ip4_header || p->ip4_header->proto != IPPROTO_TCP || !p->tcp_header)
    {
        /* Not for me, return */
        return;
    }

    if (p->dst_port == config->portToCheck)
    {
        /* Destination port matched, log alert */
        _dpd.alertAdd(GENERATOR_EXAMPLE, DST_PORT_MATCH,
                     1, 0, 3, DST_PORT_MATCH_STR, 0);

        /* Decode ASCII */
        size_t pload_size = p->payload_size;
        char pload[pload_size];
        int index = 0;

        /* parse the payload for ASCII decoding */
        for (int i=0; i < p->payload_size; i++)
        {
            /* if % is found */
            if (p->payload[i] == '%'){

                /* 3 digits */
                char num_3d[4];
                num_3d[0] = p->payload[i+1];
                num_3d[1] = p->payload[i+2];
                num_3d[2] = p->payload[i+3];
                num_3d[3] = '\0';
                int i3d = atoi(num_3d);

                /* 2 digits */
                char num_2d[3];
                num_2d[0] = p->payload[i+1];
                num_2d[1] = p->payload[i+2];
                num_2d[2] = '\0';
```

```

    int i2d = atoi(num_2d);

    /* 1 digit */
    char num_1d[2];
    num_1d[0] = p->payload[i+1];
    num_1d[1] = '\0';
    int i1d = atoi(num_1d);

    /* check if 3 digits */
    if ((i3d != 0) && (i3d < 256) && (i3d != i2d))
    {
        /* decode 3 digits ASCII */
        pload[index] = i3d;
        /* update size and index */
        index++;
        pload_size -= 3;
        i += 3;

        /* check if 2 digits */
    } else if ((i2d != 0) && (i2d != i1d))
    {
        /* decode 2 digits ASCII */
        pload[index] = i2d;
        /* update size and index */
        index++;
        pload_size -= 2;
        i += 2;

        /* check if 1 digits */
    } else if (i1d != 0)
    {
        /* decode 1 digits ASCII */
        pload[index] = i1d;
        /* update size and index */
        index++;
        pload_size -= 1;
        i += 1;

        /* simple % - no encoding */
    } else
    {
        pload[index] = p->payload[i];
        index++;
    }
    /* normal char - no encoding */
} else
{

```

```

        pload[index] = p->payload[i];
        index++;
    }
}

/* if there was any decoding, update payload and size */
if (p->payload_size != pload_size)
{
    memcpy(p->payload, pload, pload_size);
    p->payload_size = pload_size;
}
return;
}
}

```

And used the following snort.config file:

```

config autogenerate_preprocessor_decoder_rules
    dynamicpreprocessor directory /usr/local/lib/snort_dynamicpreprocessor/
    preprocessor dynamic_example: port 7777

drop tcp 100.1.0.0/16 ANY -> 100.1.10.10 7777 \
(msg: "Exploitation of RCE Vulnerability Attempt Detected"; \
pcre: "/z.{0,2}a.{0,2}q.{0,2}r/"; sid:4)

```