



udp

## Tarea N°5

Ramo: Taller de redes y servicios

Sección: 1

Profesor: Nicolás Boettcher

Ayudante: Juan Pablo Briones

Integrantes: Felipe Ulloa, Marcos Valderrama

### 1. Resumen

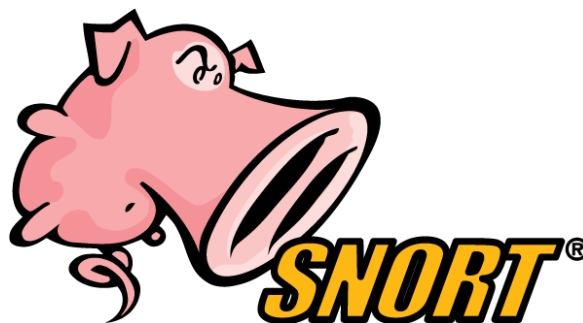
En este trabajo, se tiene como objetivo interceptar, modificar el tráfico generando el escenario de un tráfico anómalo, para finalmente alertar con una aplicación, que funciona a partir de ciertas reglas de tráfico señaladas. Todo lo anterior en el contexto de la dupla Cliente-Servidor que se comunican con cierto protocolo, trabajados en todo el semestre.

### 2. Introducción

Alertar las ocurrencias de tráfico anómalo es una gran herramienta, de suma importancia a la hora de mantener la seguridad del servicio a auditar, ya que al recibir esta alerta, se podrá reaccionar y solucionar a la brevedad posible el tráfico anómalo. El servicio que se está auditando es la dupla cliente-Servidor que se comunica por tráfico MDNS.

La aplicación utilizada para alertar es un tipo IDS, un IDS es un modelo de seguridad aplicable tanto a ordenadores como redes. Un sistema IDS recolecta y analiza información procedente de distintas áreas con el objetivo de identificar posibles vulnerabilidades de seguridad.

El IDS utilizado para alertar fue SNORT, es un IDS basado en red (NIDS). Implementa un motor de detección de ataques y barrido de puertos que permite registrar, alertar y responder ante cualquier anomalía en tiempo real, previamente definida.



(Imagen 1: Logo Snort)

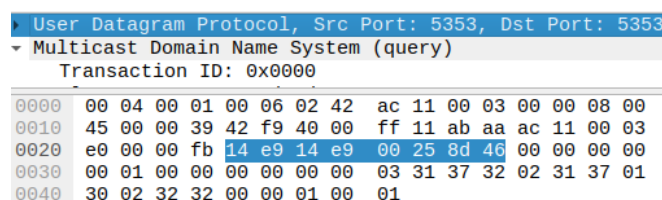
### 3.Desarrollo

#### 3.1. Primera regla/alerta con content, offset y depth, para una consulta con count queries=0.

##### 3.1.1 Explicación primera regla.

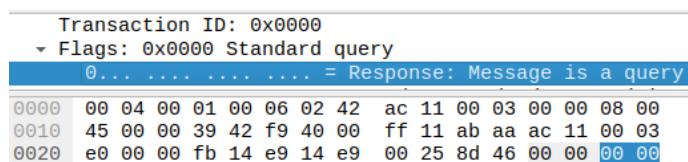
La primera regla busca alertar cuando un paquete es consulta y además tenga su contador de queries en 0, ya que es anómalo tener un paquete de consulta, sin consultas.

Para aquello necesitamos tener 3 posiciones en consideración y el valor que queremos buscar. Primero se generará una alerta con el protocolo udp, por lo tanto snort tomará de referencia el byte final de la capa UDP como el byte 0 (*Imagen 2*). Luego queremos determinar que el paquete sea una consulta, y esto se logra revisando los Flags de MDNS, donde los bytes deben tener el valor de "00 00" en hexadecimal (*Imagen 3*), para indicar que el paquete es una query. Por ultima, el contador de query's del paquete, como se busca alertar cuando sea 0 y MDNS dispone para el contador de query's de 2 bytes, se requiere que tenga el valor "00 00" en hexadecimal (*Imagen 4*).



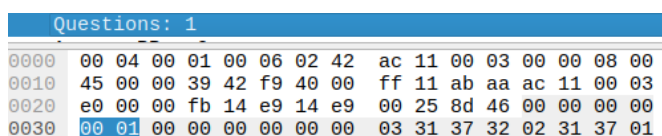
User Datagram Protocol, Src Port: 5353, Dst Port: 5353	
Multicast Domain Name System (query)	
Transaction ID: 0x0000	
0000	00 04 00 01 00 06 02 42 ac 11 00 03 00 00 08 00
0010	45 00 00 39 42 f9 40 00 ff 11 ab aa ac 11 00 03
0020	e0 00 00 fb 14 e9 14 e9 00 25 8d 46 00 00 00 00
0030	00 01 00 00 00 00 00 00 03 31 37 32 02 31 37 01
0040	30 02 32 32 00 00 01 00 01

(Imagen 2: Posición de referencia, encabezado UDP)



Transaction ID: 0x0000	
Flags: 0x0000 Standard query	
0... .. = Response: Message is a query	
0000	00 04 00 01 00 06 02 42 ac 11 00 03 00 00 08 00
0010	45 00 00 39 42 f9 40 00 ff 11 ab aa ac 11 00 03
0020	e0 00 00 fb 14 e9 14 e9 00 25 8d 46 00 00 00 00

(Imagen 3: posición de los Bytes que indica que el paquete es una consulta )



Questions: 1	
0000	00 04 00 01 00 06 02 42 ac 11 00 03 00 00 08 00
0010	45 00 00 39 42 f9 40 00 ff 11 ab aa ac 11 00 03
0020	e0 00 00 fb 14 e9 14 e9 00 25 8d 46 00 00 00 00
0030	00 01 00 00 00 00 00 00 03 31 37 32 02 31 37 01

(Imagen 4: Posición de los Bytes, que indica la cantidad de consultas que contiene el paquete)

Uniendo lo anterior, con la posición de los byte's mostrada en imagen, necesitamos buscar el contenido "|00 00 00 00 |", que los primeros 2 bytes indican el flags, de que el paquete es una query, y los últimos 2 indican que el contador de query's está en 0. Lo anterior sucede a una distancia de 2 byte del último byte de la capa UDP, por lo que necesitamos un offset = 2, y además como revisamos sólo 4 byte's, necesitamos una profundidad de 4 byte's (depth=4), Resultando en la siguiente regla:

##### Regla snort:

```
alert udp $HOME_NET any -> 224.0.0.251 5353 (content: "|00 00 00 00|"; msg:"Consulta con query count 0 ";offset:2;depth:4 ;sid:1;)
```

(Imagen 5: Primera regla Snort)

### 3.1.2 Generando el escenario de count queries = 0.

Para poder generar el escenario de tener consultas con count queries=0 se hizo uso de la herramienta polymorph, que intercepta paquetes en el vuelo y permite modificarlos en tiempo real.

Los contenedores del Cliente-Servidor se encuentran en la interfaz de docker con dirección 172.17.0.0/16, el servidor dispone la 172.17.0.2 y el cliente la 172.17.0.3.

La función utilizada en polymorph fue la siguiente:

```
def funcion1(packet):
    try:
        if(packet['IP']['src'] == '172.17.0.3'):
            packet['MDNS']['count.queries']=0
            print("\n contador modificado a :",packet['MDNS']['count.queries'])
            return packet
    except:
        return packet
```

(Imagen 6: funcion1, primer escenario)

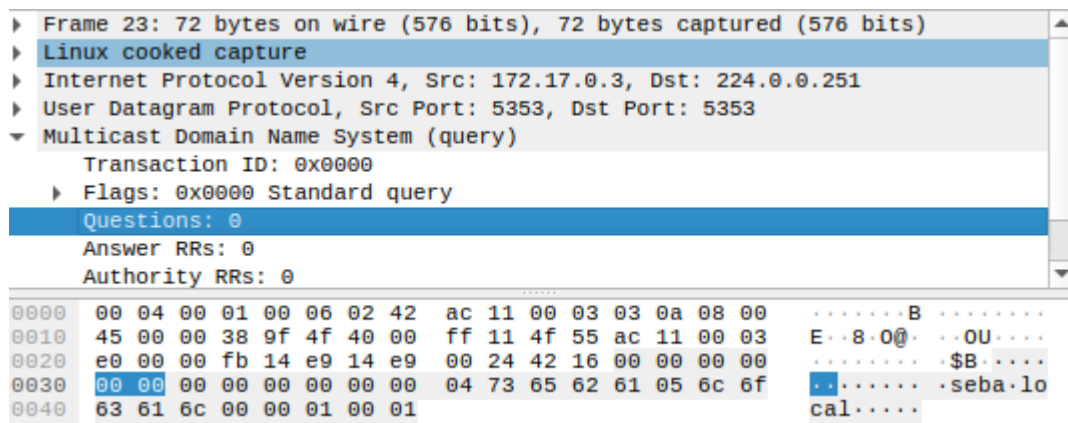
```
^CPH:cap/t1 > intercept
[*] Waiting for packets...
```

(Press Ctrl-C to exit)

```
contador modificado a : 0
```

```
contador modificado a : 0
```

(Imagen 7: Resultados Polymorph funcion1)



The image shows a Wireshark packet capture of a DNS query. The packet is 72 bytes on wire and 72 bytes captured. It is a Multicast Domain Name System (query) packet. The Transaction ID is 0x0000. The Flags are 0x0000, indicating a Standard query. The Questions field is 0. The Answer RRs field is 0. The Authority RRs field is 0. The packet data is shown in hexadecimal and ASCII format.

Offset	Hex	ASCII
0000	00 04 00 01 00 06 02 42 ac 11 00 03 03 0a 08 00	.....B.....
0010	45 00 00 38 9f 4f 40 00 ff 11 4f 55 ac 11 00 03	E..8.0@..OU....
0020	e0 00 00 fb 14 e9 14 e9 00 24 42 16 00 00 00 00	.....\$B.....
0030	00 00 00 00 00 00 00 00 04 73 65 62 61 05 6c 6f	.....seba.lo
0040	63 61 6c 00 00 01 00 01	cal.....

(Imagen 8: Count.queries modificado en wireshark)

```
12:57:53.839 [error] GenServer Mdns.Server terminating
** (MatchError) no match of right hand side value: {:error, :fmt}
(dns) lib/dns/record.ex:57: DNS.Record.decode/1
(mdns) lib/mdns/server.ex:90: Mdns.Server.handle_packet/4
(mdns) lib/mdns/server.ex:86: Mdns.Server.handle_info/2
(stdlib) gen_server.erl:637: :gen_server.try_dispatch/4
(stdlib) gen_server.erl:711: :gen_server.handle_msg/6
(stdlib) proc_lib.erl:249: :proc_lib.init_p_do_apply/3
Last message: {:udp, #Port<0.6>, {172, 17, 0, 3}, 5353, <<0, 0, 0,
```

(Imagen 9: Caída del servidor)

Las modificaciones quedaron capturadas en el siguiente pcap, donde las modificaciones se realizaron desde el paquete 22, para poder ver el contraste con las consultas normales, con las modificadas resultando en 10 paquetes modificados y la caída del servidor en el primer modificada.

20	3.3499...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6...
21	3.7029...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A www.icarito.cl, "QM...
22	3.7032...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6...
23	7.0582...	172.17.0.3	224.0.0.251	MDNS	72	Standard query 0x0000
24	10.410...	172.17.0.3	224.0.0.251	MDNS	74	Standard query 0x0000
25	13.787...	172.17.0.3	224.0.0.251	MDNS	73	Standard query 0x0000
26	17.146...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000
27	20.499...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000
28	23.852...	172.17.0.3	224.0.0.251	MDNS	73	Standard query 0x0000
29	27.203...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000
30	30.568...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000
31	33.922...	172.17.0.3	224.0.0.251	MDNS	73	Standard query 0x0000
32	37.278...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000

(Imagen 10: Primer escenario, ModCountQueries0.pcap)

### 3.1.3 Resultados primera alerta Snort.

El comando utilizado para poder revisar la captura del escenario generado, y generar las alertas correspondientes, guardandolo en un archivo denominado CountQueries, es el siguiente:

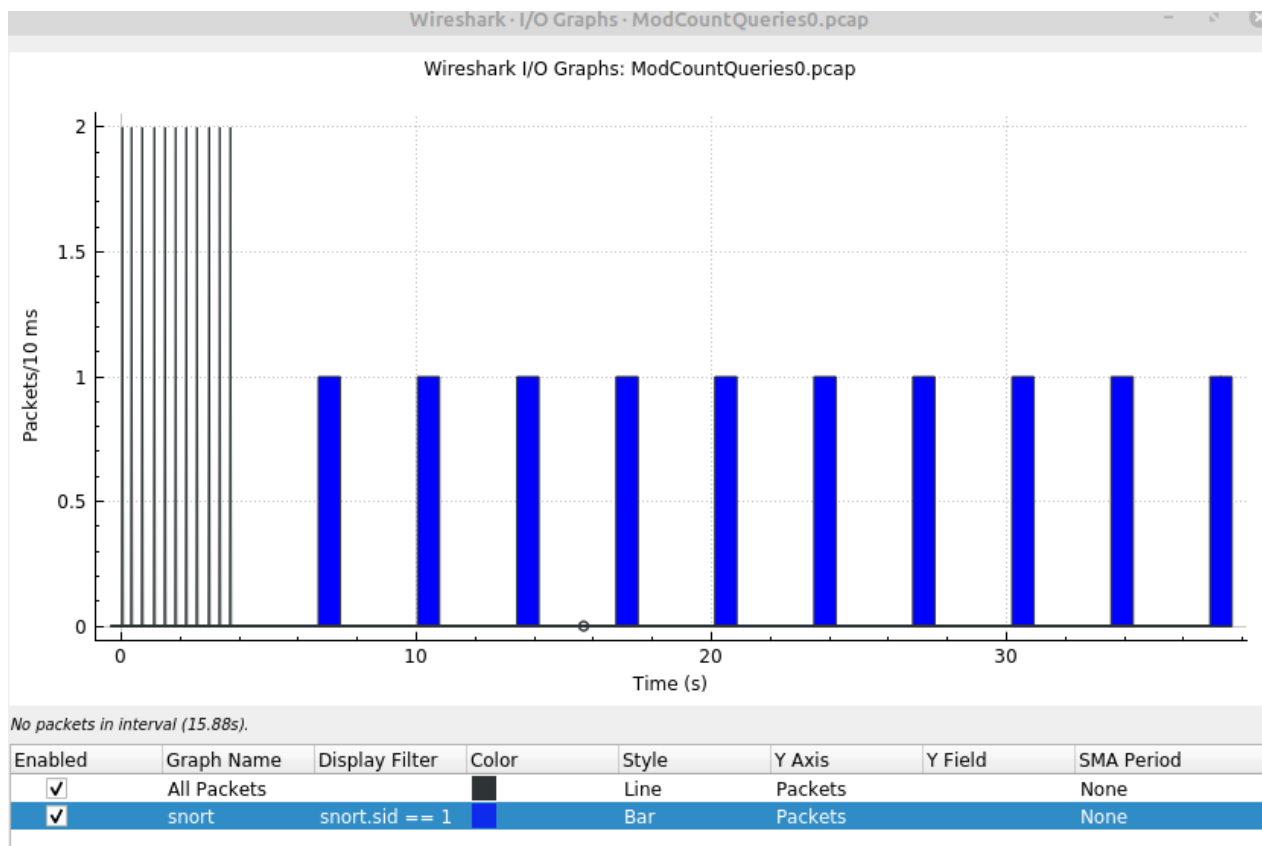
```
sudo snort -A console -c /etc/snort/snort.conf -k none
-r ModCountQueries0.pcap &> CountQueries
```

(Imagen 11: Comando snort. regla 1)

Como se mencionó en la captura .pcap, tenemos 10 paquetes modificados y que son query's

```
felipe@felipe-notebook:~/workspace/tallerRedes$ grep "Alerts\|Total:" CountQueries
Total: 32
Alerts: 10 ( 31.250%)
```

(Imagen 12: Resultados primera regla Snort)



(Imagen 13: Gui primer alerta snort en Wireshark)

22	0.000354	172.17.0.2	224.0.0.251	MDNS	157	Standard query response
23	3.354997	172.17.0.3	224.0.0.251	MDNS	72	Standard query 0x0000

<ul style="list-style-type: none"> <li>Frame 23: 72 bytes on wire (576 bits), 72 bytes captured (576 bits)</li> <li>Linux cooked capture</li> <li>Internet Protocol Version 4, Src: 172.17.0.3, Dst: 224.0.0.251</li> <li>User Datagram Protocol, Src Port: 5353, Dst Port: 5353</li> <li>Multicast Domain Name System (query)</li> <li>Snort: (msg: "Consulta con query count 0 " sid: 1 rev: 0) [from Running Snort]</li> </ul>
---

(Imagen 14: Expert info, paquete con alerta de primera regla)

### 3.2. Segunda regla /alerta con pcre, para una consulta o respuesta con una dirección no local.

#### 3.2.1 Explicación segunda regla.

La segunda regla busca alertar cuando un paquete de consulta o respuesta contenga una dirección no local por medio de una regla con expresiones regulares. Si bien no es una vulnerabilidad, es un tráfico anómalo, ya que se puede realizar consultas o respuestas por consultas no locales, pero solo si no hay un servidor dns funcionando correctamente, por lo cual alerta una anomalía respecto a un falla por parte del servicio DNS convencional.

```
alert udp any any -> 224.0.0.251 5353 (msg:"Se consulta o responde por una direccion no local" ;pcre:"/(http[s]?://){0,1}(w{3,3}.)*[-a-z0-9+&@#/%?=_!:.]*[-a-z0-9+&@#/%?=_!:.]"/ ;sid:2;)
```

(Imagen 15: Segunda regla Snort)

La expresión regular, realiza match, con palabras que tengan o no http[s]://, luego debe seguir de un "www.", luego pueden haber caracteres de la "a" a la "z" de 0 a 9 veces, luego unos caracteres especiales terminando con un punto, para luego del segundo punto, puedan nuevamente ir caracteres de 0 a 9 veces, para terminar con unos caracteres especiales, pero ahora no el punto.

#### 3.2.2 Generando el escenario de consulta y respuestas con direcciones no locales.

Para este escenario no se tuvo que modificar tráfico con polymorph, como es un escenario anómalo que puede ocurrir, se determinó por modificar el cliente y servidor para que el cliente consulte por una dirección externa "www.icarito.cl" y el servidor responda, por una consulta inversa, "www.iotuve.cl".

```
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
▼ Queries
  ▼ www.icarito.cl: type A, class IN, "QM" question
    Name: www.icarito.cl
```

(Imagen 16: Dirección externa en consulta)

```
▼ Answers
  ▶ seba.local: type A, class IN, addr 172.17.0.6
  ▶ example1.local: type A, class IN, addr 172.17.0.9
  ▶ example4.local: type A, class IN, addr 172.17.0.10
  ▼ 172.17.0.36: type PTR, class IN, www.iotuve.cl
    Name: 172.17.0.36
    Type: PTR (domain name PointeR) (12)
    .0000 0000 0000 0001 = Class: IN (0x0001)
    0... .... = Cache flush: False
    Time to live: 64 (1 minute, 4 seconds)
    Data length: 15
    Domain Name: www.iotuve.cl
```

(Imagen 17: Dirección externa en respuesta)

Se puede apreciar que se realizaron 2 consultas y 2 respuestas con direcciones externas en la siguiente captura:

13	2.1994...	172.17.0.3	224.0.0.251	MDNS	73	Standard query 0x0000 A 172.17.0.36, "QM" question
14	2.1997...	172.17.0.2	224.0.0.251	MDNS	195	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
15	2.5645...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example7.local, "QM" question
16	2.5647...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
17	2.9388...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A www.icarito.cl, "QM" question
18	2.9391...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
19	6.3039...	192.168.1.4	224.0.0.251	MDNS	72	Standard query 0x0000 A seba.local, "QM" question
20	6.3043...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
21	6.6796...	192.168.1.4	224.0.0.251	MDNS	74	Standard query 0x0000 A felipe.local, "QM" question
22	6.6800...	172.17.0.2	224.0.0.251	MDNS	201	Standard query response 0x0000 A 172.17.0.6 TXT A 172.17.0.9 ...
23	7.0478...	192.168.1.4	224.0.0.251	MDNS	73	Standard query 0x0000 A 172.17.0.22, "QM" question
24	7.0482...	172.17.0.2	224.0.0.251	MDNS	196	Standard query response 0x0000 A 172.17.0.6 PTR _rosetta._tcp...
25	7.4070...	192.168.1.4	224.0.0.251	MDNS	76	Standard query 0x0000 A example1.local, "QM" question
26	7.4073...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
27	7.7637...	192.168.1.4	224.0.0.251	MDNS	76	Standard query 0x0000 A example2.local, "QM" question
28	7.7640...	172.17.0.2	224.0.0.251	MDNS	204	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 TXT ...
29	8.1375...	192.168.1.4	224.0.0.251	MDNS	73	Standard query 0x0000 A 172.17.0.33, "QM" question
30	8.1378...	172.17.0.2	224.0.0.251	MDNS	194	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 PTR ...
31	8.5054...	192.168.1.4	224.0.0.251	MDNS	76	Standard query 0x0000 A example4.local, "QM" question
32	8.5057...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
33	8.8760...	192.168.1.4	224.0.0.251	MDNS	76	Standard query 0x0000 A example5.local, "QM" question
34	8.8763...	172.17.0.2	224.0.0.251	MDNS	204	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
35	9.2431...	192.168.1.4	224.0.0.251	MDNS	73	Standard query 0x0000 A 172.17.0.36, "QM" question
36	9.2434...	172.17.0.2	224.0.0.251	MDNS	195	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
37	9.6129...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example7.local, "QM" question
38	9.6132...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
39	9.9770...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A www.icarito.cl, "QM" question
40	9.9773...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...

(Imagen 18: Escenario 2, URLmdns.pcap)

### 3.2.3 Resultados segunda alerta Snort.

El comando utilizado para poder revisar la captura del escenario generado, y generar las alertas correspondientes, guardandolo en un archivo denominado url, es el siguiente:

```
sudo snort -A console -c /etc/snort/snort.conf -k none
-r URLmdns.pcap &> url
```

(Imagen 19: Comando snort. regla 2)

Como se mencionó en la captura URLmdns.pcap, tenemos 2 paquetes de consulta y 2 respuestas con dirección no local, resultando en 4 alertas.

```
felipe@felipe-notebook:~/workspace/tallerRedes$ grep "Alerts\\|Total:" url
Total: 44
Alerts: 4 ( 9.091%)
```

(Imagen 20: Resultados segunda regla Snort)

Wireshark no quiso alertar la segunda regla, con expresiones regulares, siendo que snort las alerta sin ningún problema, se deja imagen donde se puede apreciar que wireshark no muestra alerta y donde snort si, se puede ver que son el mismo paquete por el "Arrival time":

18	0	172.17.0.2	224.0.0.251	MDNS	1	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 172.17.0.10 PTR www.10tuve.cl A 172.17.0.11
19	0	172.17.0.3	224.0.0.251	MDNS	74	Standard query
20	0	172.17.0.2	224.0.0.251	MDNS	1	Standard query
21	0	172.17.0.3	224.0.0.251	MDNS	74	Standard query
22	0	172.17.0.2	224.0.0.251	MDNS	1	Standard query
23	3	172.17.0.3	224.0.0.251	MDNS	70	Standard query
24	0	172.17.0.2	224.0.0.251	MDNS	1	Standard query
25	0	172.17.0.3	224.0.0.251	MDNS	72	Standard query

Frame 18:	193 bytes on wire (1544 bits), 193 bytes captured on interface
Encapsulation type:	Ethernet (I)
Arrival time:	Dec 5, 2021 10:50:45.402481000 -03
[Time shift for this packet:	0.000000000 seconds]
Epoch Time:	1638712245.402481000 seconds
[Time delta from previous captured frame:	0.000351000 seconds]

300	01 00 5e 00 00 fb 02 42 ac 11 00 02 08 00 4b 00 00	12/05-10:50:45.402481	[**] [1:2:0] Se consulta o responde por una direccion no local
310	00 b3 e7 9e 40 00 ff 11 06 8c ac 11 00 02 e0 00 00	12/05-10:50:46.130117	[**] [1:2:0] Se consulta o responde por una direccion no local
320	00 fb 14 e9 14 e9 00 9f 8d bf 00 00 84 00 00 00 00	12/05-10:50:53.198333	[**] [1:2:0] Se consulta o responde por una direccion no local
330	00 05 00 00 00 00 04 73 65 62 61 05 6c 6f 63 61	12/05-10:50:52.473995	[**] [1:2:0] Se consulta o responde por una direccion no local
340	6c 00 00 01 00 01 00 00 00 40 00 04 ac 11 00 06 00		
350	08 65 78 61 6d 70 6c 65 31 c0 11 00 01 00 01 00		
360	00 00 00 00 04 ac 11 00 09 08 65 78 61 6d 70 6c		
370	65 34 c0 11 00 01 00 01 00 00 00 ff 00 04 ac 11		
380	00 0a 03 31 37 32 02 31 37 01 30 02 33 36 00 00		
390	0c 00 01 00 00 00 40 00 0f 03 77 77 77 06 69 6f		
3a0	74 75 76 65 02 63 6c 00 08 65 78 61 6d 70 6c 65		
3b0	37 c0 11 00 01 00 01 00 00 00 40 00 04 ac 11 00		

(Imagen 21: Comando snort. regla 2)



### 3.3. Tercera regla /alerta con flow, para una consulta proveniente de una red externa.

#### 3.3.1 Explicación tercera regla.

La tercera regla busca alertar una posible vulnerabilidad que sucede cuando existe el caso de que un host externo a la red, logre consultar a nuestro servicio MDNS, obteniendo información o alterando la existente, y es una vulneración además, por que el protocolo MDNS por definición funciona de manera local.

Con la opción de flow indicamos que se aplique la regla solo en uno de los sentidos del flujo de comunicación. De esta manera se puede aplicar la regla solo a los clientes o servidores. Las opciones según los flujos son:

- **to\_client** respuesta de servidores A hacia clientes B
- **to\_server** peticiones de servidores A hacia B
- **from\_client** peticiones de clientes A hacia B
- **from\_server** respuesta de servidores de A hacia B
- **established** conexiones TCP establecidas
- **stateless** no tiene en cuenta la inspección de estados

Por lo tanto, para la tercera regla, se debe implementar flow con **from\_client**, para alertar de peticiones provenientes de tráfico no local.

```
alert udp !$HOME_NET any -> 224.0.0.251 5353 (msg:"Tráfico anómalo proveniente fuera de local";  
flow:from_client; sid:3;)
```

*(Imagen 22: Tercera regla Snort)*

#### 3.3.2 Generando el escenario para tráfico proveniente de otra red.

Para poder generar tráfico el escenario de tener consultas provenientes de otra red, se hizo uso de la herramienta polymorph, modificando el ip.src de los paquetes del cliente, por una dirección ip que no corresponde a la red.

```
def funcion3(packet):  
    try:  
        if(packet['IP']['src'] == '172.17.0.3'):  
            packet['IP']['src'] == '192.168.1.4'  
            return packet  
    except:  
        return packet
```

*(Imagen 23: funcion3, tercer escenario)*



Las modificaciones se quedaron capturadas en el siguiente pcap, donde las modificaciones se realizaron desde el paquete 18 hasta el 35, resultando en 9 paquetes provenientes de la red 192.168.1.4.

17	2.9388...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A www.icarito.cl, "QM" question
18	2.9391...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
19	6.3039...	192.168.1.4	224.0.0.251	MDNS	72	Standard query 0x0000 A seba.local, "QM" question
20	6.3043...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
21	6.6796...	192.168.1.4	224.0.0.251	MDNS	74	Standard query 0x0000 A felipe.local, "QM" question
22	6.6800...	172.17.0.2	224.0.0.251	MDNS	201	Standard query response 0x0000 A 172.17.0.6 TXT A 172.17.0.9 ...
23	7.0478...	192.168.1.4	224.0.0.251	MDNS	73	Standard query 0x0000 A 172.17.0.22, "QM" question
24	7.0482...	172.17.0.2	224.0.0.251	MDNS	196	Standard query response 0x0000 A 172.17.0.6 PTR _rosetta._tcp...
25	7.4070...	192.168.1.4	224.0.0.251	MDNS	76	Standard query 0x0000 A example1.local, "QM" question
26	7.4073...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
27	7.7637...	192.168.1.4	224.0.0.251	MDNS	76	Standard query 0x0000 A example2.local, "QM" question
28	7.7640...	172.17.0.2	224.0.0.251	MDNS	204	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 TXT ...
29	8.1375...	192.168.1.4	224.0.0.251	MDNS	73	Standard query 0x0000 A 172.17.0.33, "QM" question
30	8.1378...	172.17.0.2	224.0.0.251	MDNS	194	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 PTR ...
31	8.5054...	192.168.1.4	224.0.0.251	MDNS	76	Standard query 0x0000 A example4.local, "QM" question
32	8.5057...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
33	8.8760...	192.168.1.4	224.0.0.251	MDNS	76	Standard query 0x0000 A example5.local, "QM" question
34	8.8763...	172.17.0.2	224.0.0.251	MDNS	204	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
35	9.2431...	192.168.1.4	224.0.0.251	MDNS	73	Standard query 0x0000 A 172.17.0.36, "QM" question
36	9.2434...	172.17.0.2	224.0.0.251	MDNS	195	Standard query response 0x0000 A 172.17.0.6 A 172.17.0.9 A 17...
37	9.6129...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example7.local, "QM" question

(Imagen 24: Tercer escenario, NoLocalMdns.pcap)

### 3.3.3 Resultados tercera alerta Snort.

El comando utilizado para poder revisar la captura del escenario generado, y generar las alertas correspondientes, guardandolo en un archivo denominado localNo, es el siguiente:

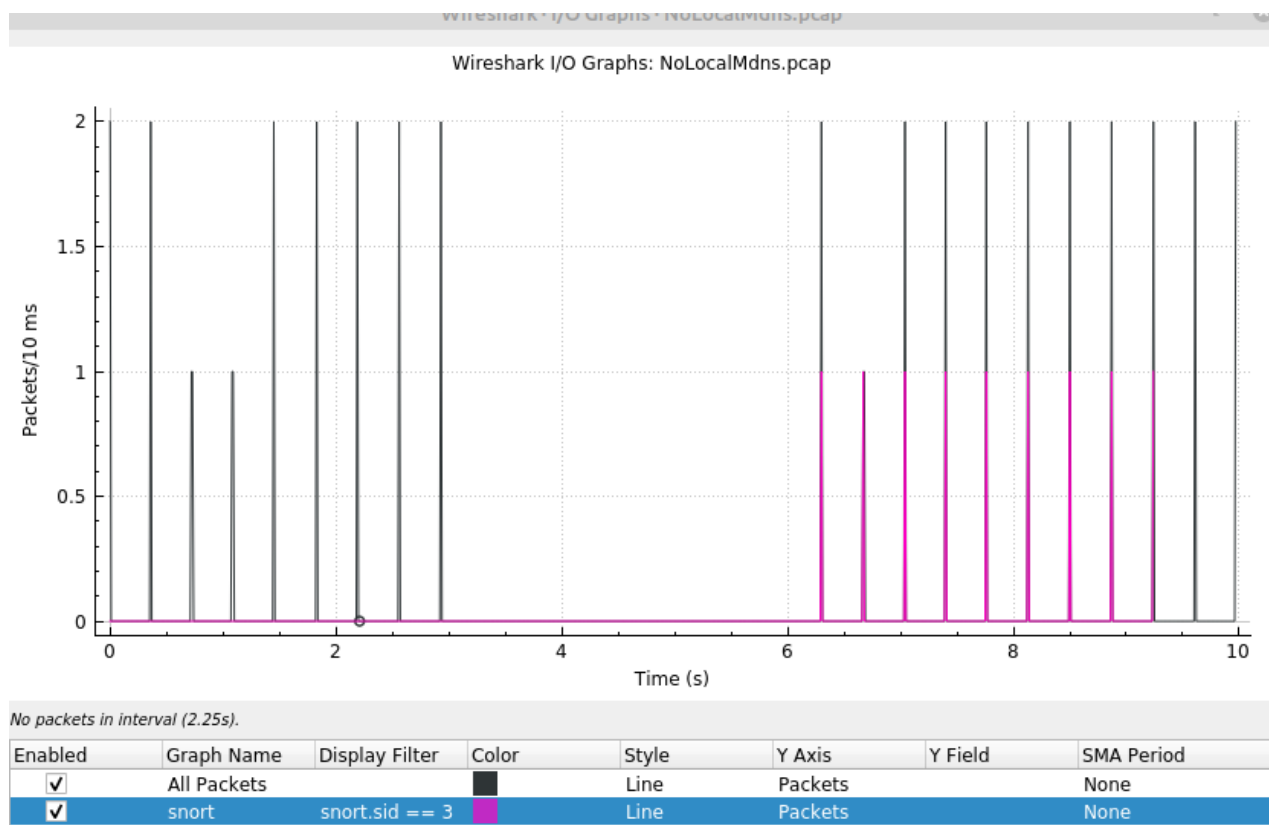
```
sudo snort -A console -c /etc/snort/snort.conf -k none
-r NoLocalMdns.pcap &> localNo
```

(Imagen 25: Comando snort. regla 3)

Como se mencionó en la captura NoLocalMdns.pcap, tenemos 9 paquetes modificados, provenientes de una red que no es la local, lo que resulta en 9 alertas.

```
felipe@felipe-notebook:~/workspace/tallerRedes$ grep "Alerts\|Total:" localNo
Total: 40
Alerts: 9 ( 22.500%)
```

(Imagen 26: Resultados tercera regla Snort)



(Imagen 27: Gui segunda alerta snort en Wireshark)

26	0.000306	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 1
27	0.356352	192.168.1.4	224.0.0.251	MDNS	76	Standard query 0x0000 A example2.1
28	0.000279	172.17.0.2	224.0.0.251	MDNS	204	Standard query response 0x0000 A 1

<ul style="list-style-type: none"> <li>▶ Frame 27: 76 bytes on wire (608 bits), 76 bytes captured (608 bits)</li> <li>▶ Linux cooked capture</li> <li>▶ Internet Protocol Version 4, Src: 192.168.1.4, Dst: 224.0.0.251</li> <li>▶ User Datagram Protocol, Src Port: 5353, Dst Port: 5353</li> <li>▶ Multicast Domain Name System (query)</li> <li>▶ Snort: (msg: "Trafico anomalo proveniente fuera de local" sid: 3 rev: 0) [from Running Snort]</li> </ul>
---

(Imagen 28: Expert info, paquete con alerta de segunda regla)

### 3.4. Cuarta regla /alerta con dsize, para un paquete con una cantidad de bytes distinta al esperado en el payload.

#### 3.4.1 Explicación cuarta regla.

La cuarta regla verifica que se cumpla con la cantidad especificada en bytes desde el payload del paquete, esta cantidad es entregada como parámetro siendo definida mayor o menos, e incluso entre medio de la cantidad que se defina, en caso de cumplir con esta condición, esté alerta sobre el paquete. Si bien esto depende de cada escenario y de cómo se dé uso a esta. Por ejemplo, ya conociendo la cantidad mínima de bytes que se puede esperar de un paquete entonces recibir al menor del esperado se puede alertar como anomalía.

```
alert udp any any -> 224.0.0.251 5353 (msg:"Paquete anomalo, tiene menos del payload minimo";  
dsize:<17; sid:4;)
```

*(Imagen 29: Cuarta regla, Snort)*

En este caso si la cantidad de bytes del payload es menor a 17, entonces se activa la alerta.

#### 3.4.2 Generando el escenario de un paquete con payload menor a 17 bytes.

En este escenario se utilizó polymorph para reducir el tamaño del payload del paquete el cual resultó en un total de 16 bytes, para realizar esto se modificó la query dejándola vacía, de esta manera no se utilizan bytes para ese parámetro, además se tuvo que cambiar el *length* ubicado en la capa UDP e IP para mantener la consistencia del paquete y que por ello no sea omitido al analizarlo con el programa Snort.

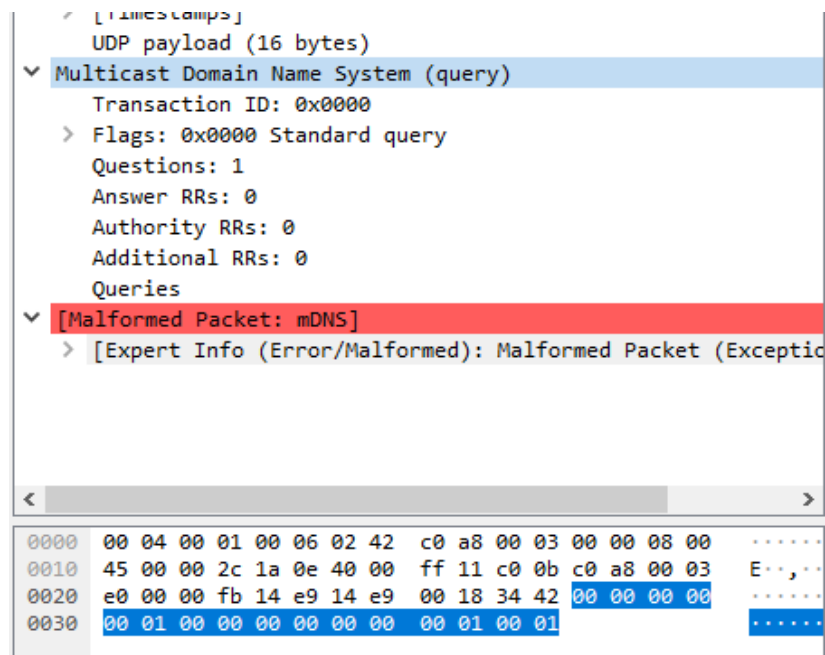
```
def funcion4(packet):  
    try:  
        if(packet['IP']['src'] == '192.168.0.3'):  
            if(packet['UDP']['length'] == 36 ):  
                packet['MDNS']['qry.name']=''  
                packet['UDP']['length'] = 24  
                packet['IP']['len'] = 44  
                return packet  
    except:  
        return packet
```

*(Imagen 30: funcion4, cuarto escenario)*

De aquí se generó solo un paquete con estas características desde el cliente, el cual provocó la caída del servidor debido a que la query de esta se encontraba vacía. En este escenario se capturaron en el archivo *dSize.cap* un total de 16 paquetes incluyendo el modificado.

1	0.000000	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example7.local, "QM" question
2	0.819061	192.168.0.3	224.0.0.251	MDNS	72	Standard query 0x0000 A seba.local, "QM" question
3	0.826643	192.168.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 A felipe.local, "QM" question
4	0.830311	192.168.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 A marcos.local, "QM" question
5	0.832869	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example1.local, "QM" question
6	0.824574	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example2.local, "QM" question
7	0.834153	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example4.local, "QM" question
8	0.828016	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example5.local, "QM" question
9	0.835671	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example6.local, "QM" question
10	0.830824	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example7.local, "QM" question
11	0.835329	192.168.0.3	224.0.0.251	MDNS	60	[Standard query 0x0000 [Malformed Packet]
12	3.817716	192.168.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 A felipe.local, "QM" question
13	3.818497	192.168.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 A marcos.local, "QM" question
14	4.033804	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example1.local, "QM" question
15	3.804708	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example2.local, "QM" question
16	3.820997	192.168.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example4.local, "QM" question

(Imagen 31: Cuarto escenario, dSize.cap)



(Imagen 32: Cuarto escenario, paquete modificado)

### 3.4.3 Resultados cuarta alerta Snort.

El comando utilizado para poder revisar la captura del escenario generado, y generar las alertas correspondientes, guardandolo en un archivo denominado dsize, es el siguiente:

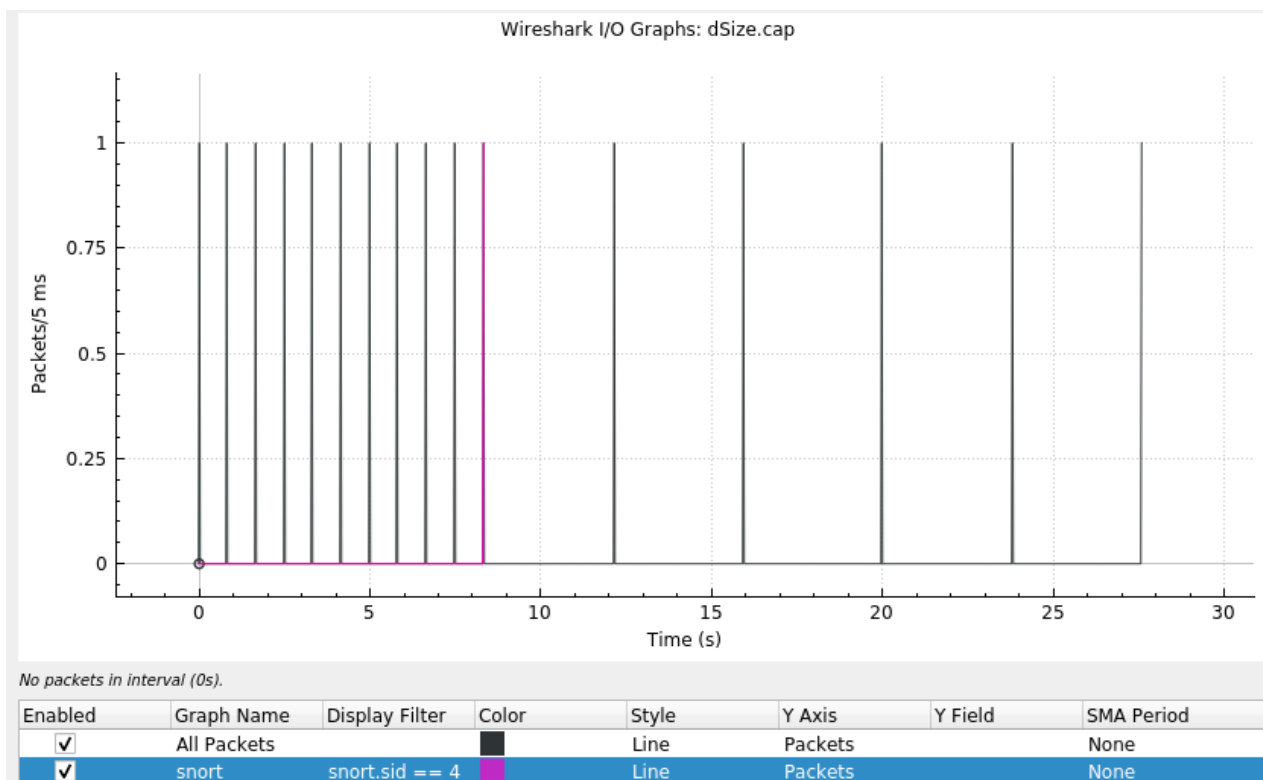
```
sudo snort -A console -c /etc/snort/snort.conf -k none
-r dSize.cap &> dsize
```

(Imagen 33: Comando snort. regla 4)

Como se mencionó en la captura dSize.cap, tenemos 1 paquete modificado de un total de 16 paquetes, el cual resultaría generando una alerta:

```
felipe@felipe-pc:~/workspace/tallerredes2$ grep "Alerts:\ \|Total:" dsize
Total: 16
Alerts: 1 ( 6.250%)
```

(Imagen 34: Resultados cuarta regla Snort)



(Imagen 34: Gui cuarta alerta snort en Wireshark)

11	0.835329	192.168.0.3	224.0.0.251	MDNS	60	Standard query 0x0000[Malformed Packet]
10	0.843746	192.168.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 [Malformed Packet]

▶ Frame 11: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 192.168.0.3, Dst: 224.0.0.251
▶ User Datagram Protocol, Src Port: 5353, Dst Port: 5353
▶ Multicast Domain Name System (query)
▶ [Malformed Packet: mDNS]
▶ Snort: (msg: "Paquete anomalo, tiene menos del payload minimo" sid: 4 rev: 0) [from Running Snort]

(Imagen 35: Expert info, paquete con alerta de cuarta regla)

### 3.5. Quinta regla /alerta con offset, content y depth para un paquete con cantidad de preguntas y respuestas en la query distintos a cero.

#### 3.5.1 Explicación quinta regla.

La quinta regla verifica los bytes de una posición específica del payload del paquete, para esto se utiliza el parámetro *offset*, el cual indica la cantidad de bytes que debe ignorar a contar desde el comienzo del payload, luego de esto se usa el parámetro *depth*, el cual indica la cantidad de bytes que se considerarán a continuación para ser analizados en esta regla. En este escenario utilizamos la regla *content* como tercer parámetro, la cual compara la posición mencionada previamente con los bytes que ingresemos en esta opción. En este caso le ingresamos que los bytes deban ser distinto a '00 00' por agrupación para el antes mencionado *depth* de 4, esto indica que si ambas agrupaciones de bytes son distintas al indicado, entonces se activará la alerta, en caso contrario si al menos uno de estos dos cumple ser igual a la agrupación '00 00', entonces la alerta no se activará.

```
alert udp any any -> 224.0.0.251 5353 (content:!"|00 00|";offset:4;depth:4;sid:5;  
msg:"Paquete con ambos contadores, de preguntas y respuestas, mayor a 0");
```

(Imagen 36: Quinta regla, Snort)

#### 3.5.2 Generando el escenario de un paquete con cantidad de preguntas y respuestas distintas a cero.

Para este escenario se utilizó polymorph para modificar tanto los paquetes generados por el cliente y del servidor, en el cual se cambió el campo count.queries y count.answers de tal manera que fueran mayores a 0, así el paquete registraría ambos campos de bytes distintos a '00 00', tanto el de preguntas como respuestas, los cuales en este caso fueron de 69 y 33 respectivamente, registrando en hexadecimal de '00 45' y '00 21'.

```
def funcion5(packet):  
    try:  
        if(packet['IP']['src'] == '172.17.0.2'):  
            packet['MDNS']['count.queries'] = 69  
            print("Count queries modificada a 69")  
            packet['MDNS']['count.answers'] = 33  
            print("\nCount answers modificada a 33")  
        return packet  
    except:  
        return packet
```

(Imagen 37: función5, quinto escenario)

En este escenario el servidor registra la ip 172.17.0.2, por lo que primero se modificaron los paquetes provenientes de este y luego las del cliente, ya que estas consultas al estar mal generadas provocaron la caída del servicio.

De esta manera, para un total de 40 paquetes, se modificaron 16, los cuales quedaron registrados en el archivo countQueriesResponses.pcap de manera aislada.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000...	172.17.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 A marcos.local, "QM" question
2	0.000...	172.17.0.2	224.0.0.251	MDNS	192	Standard query response 0x0000 A 172.18.0.6 PTR _rosetta._tcp.local A 172.18.0.9 A 172.18.0.10 A 172.18.0.11
3	0.593...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example1.local, "QM" question
4	0.000...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.18.0.6 A 172.18.0.9 A 172.18.0.10 A 172.18.0.11
5	0.587...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example2.local, "QM" question
6	0.000...	172.17.0.2	224.0.0.251	MDNS	204	Standard query response 0x0000 A 172.18.0.6 A 172.18.0.9 TXT A 172.18.0.10 A 172.18.0.11
7	0.589...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example4.local, "QM" question
8	0.000...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.18.0.6 A 172.18.0.9 A 172.18.0.10 A 172.18.0.11
9	0.600...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example5.local, "QM" question
10	0.000...	172.17.0.2	224.0.0.251	MDNS	204	Standard query response 0x0000 A 172.18.0.6 A 172.18.0.9 A 172.18.0.10 TXT A 172.18.0.11
11	0.593...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example6.local, "QM" question
12	0.000...	172.17.0.2	224.0.0.251	MDNS	192	Standard query response 0x0000 A 172.18.0.6 A 172.18.0.9 A 172.18.0.10 PTR _pipo2._tcp.local A 172.18.0.11
13	0.592...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example7.local, "QM" question
14	0.000...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.18.0.6 A 172.18.0.9 A 172.18.0.10 A 172.18.0.11
15	0.590...	172.17.0.3	224.0.0.251	MDNS	72	Standard query 0x0000 A seba.local, "QM" question
16	0.000...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A 172.18.0.6 A 172.18.0.9 A 172.18.0.10 A 172.18.0.11
17	0.596...	172.17.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 A felipe.local, "QM" question
18	0.000...	172.17.0.2	224.0.0.251	MDNS	201	Standard query response 0x0000 A 172.18.0.6 TXT A 172.18.0.9 A 172.18.0.10 A 172.18.0.11
19	11.33...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example4.local, "QM" question
20	0.000...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A seba.local, "QM" question Unused <Root>, "QM" question A *{022\000\006.example1.local,
21	3.581...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example5.local, "QM" question
22	0.000...	172.17.0.2	224.0.0.251	MDNS	204	Standard query response 0x0000 A seba.local, "QM" question Unused <Root>, "QM" question A *{022\000\006.example1.local,
23	3.582...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example6.local, "QM" question
24	0.000...	172.17.0.2	224.0.0.251	MDNS	192	Standard query response 0x0000 A seba.local, "QM" question Unused <Root>, "QM" question A *{022\000\006.example1.local,
25	3.578...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example7.local, "QM" question
26	0.000...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A seba.local, "QM" question Unused <Root>, "QM" question A *{022\000\006.example1.local,
27	3.600...	172.17.0.3	224.0.0.251	MDNS	72	Standard query 0x0000 A seba.local, "QM" question
28	0.000...	172.17.0.2	224.0.0.251	MDNS	157	Standard query response 0x0000 A seba.local, "QM" question Unused <Root>, "QM" question A *{022\000\006.example1.local,
29	3.626...	172.17.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 A felipe.local, "QM" question
30	0.000...	172.17.0.2	224.0.0.251	MDNS	201	Standard query response 0x0000 A seba.local, "QM" question Unused <Root>, "QM" question TXT *{022\000\006.felipe.local,
31	21.50...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example6.local, "QM" question[Malformed Packet]
32	3.582...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example7.local, "QM" question[Malformed Packet]
33	3.576...	172.17.0.3	224.0.0.251	MDNS	72	Standard query 0x0000 A seba.local, "QM" question[Malformed Packet]
34	3.585...	172.17.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 A felipe.local, "QM" question[Malformed Packet]
35	3.580...	172.17.0.3	224.0.0.251	MDNS	74	Standard query 0x0000 A marcos.local, "QM" question[Malformed Packet]
36	3.579...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example1.local, "QM" question[Malformed Packet]
37	3.583...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example2.local, "QM" question[Malformed Packet]
38	3.577...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example4.local, "QM" question[Malformed Packet]
39	3.582...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example5.local, "QM" question[Malformed Packet]
40	3.579...	172.17.0.3	224.0.0.251	MDNS	76	Standard query 0x0000 A example6.local, "QM" question[Malformed Packet]

(Imagen 38: Quinto escenario, countQueriesResponses.pcap)

```

UDP payload (52 bytes)
▼ Multicast Domain Name System (query)
  Transaction ID: 0x0000
  > Flags: 0x0000 Standard query
    Questions: 69
    Answer RRs: 33
    Authority RRs: 0
    Additional RRs: 0
  > Queries
▼ [Malformed Packet: mDNS]
  > [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]

0000 00 04 00 01 00 06 02 42 ac 11 00 03 07 44 08 00 .....B.....D..
0010 45 00 00 3c 66 09 40 00 ff 11 88 97 ac 11 00 03 E..<f.@.....
0020 00 00 00 fb 14 e9 14 e9 00 28 7b e1 00 00 00 00 .....({.....
0030 00 45 00 21 00 00 00 00 08 65 78 61 6d 70 6c 65 .E!.....example
0040 36 05 6c 6f 63 61 6c 00 00 01 00 01 .....6.local....

```

(Imagen 39: Quinto escenario, Paquete modificado)



### 3.5.3 Resultados quinta alerta Snort.

El comando utilizado para poder revisar la captura del escenario generado, y generar las alertas correspondientes, guardandolo en un archivo denominado offset, es el siguiente:

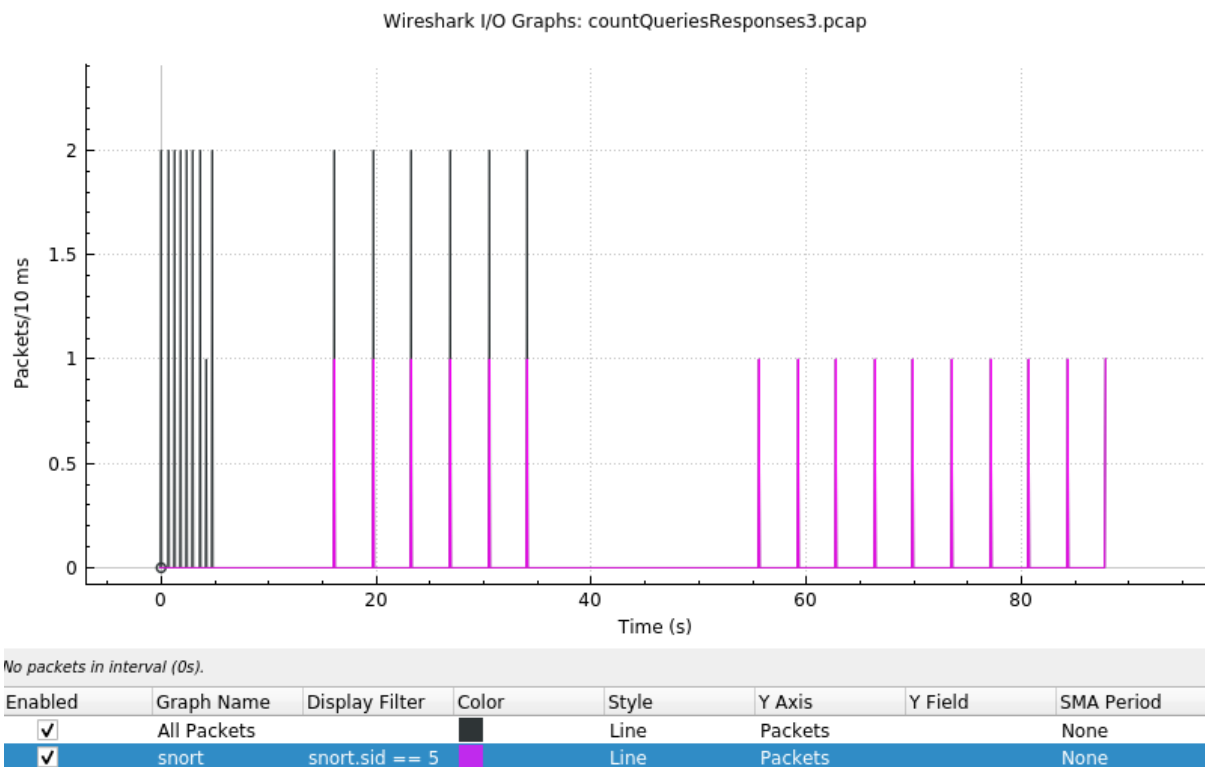
```
sudo snort -A console -c /etc/snort/snort.conf -k none  
-r countQueriesResponses.pcap &> offset
```

(Imagen 40: Comando snort. regla 4)

Como se mencionó en la captura countQueriesResponses.pcap, tenemos 16 paquete modificados de un total de 40, el cual resultaría generando las 16 alertas correspondientes.

```
felipe@felipe-pc:~/workspace/tallerredes2$ grep "Alerts:\ \|Total:" CountQueries  
Total: 32  
Alerts: 10 ( 31.250%)
```

(Imagen 41: Resultados quinta regla Snort)



(Imagen 42: Gui quinta alerta snort en Wireshark)

```
28 0.. 172.17.0.2 224.0.0.251 MDNS 1.. Standard query response 0x0000 A seba.local, "QM" question Unused <Root>, "QM" qu  
Frame 28: 157 bytes on wire (1256 bits), 157 bytes captured (1256 bits)  
Linux cooked capture  
Internet Protocol Version 4, Src: 172.17.0.2, Dst: 224.0.0.251  
User Datagram Protocol, Src Port: 5353, Dst Port: 5353  
Multicast Domain Name System (response)  
Transaction ID: 0x0000  
Flags: 0x8400 Standard query response, No error  
Questions: 69  
Answer RRs: 33  
Authority RRs: 0  
Additional RRs: 0  
Queries  
[Malformed Packet: mDNS]  
Snort: (msg: "Paquete con ambos contadores, de preguntas y respuestas, mayor a 0" sid: 5 rev: 0) [from Running Snort]
```

(Imagen 43: Expert info, paquete con alerta de quinta regla)

#### 4.Conclusión

A través de este informe se pudo observar que la experiencia se desarrolló de forma exitosa, cumpliendo con los objetivos expuestos, ya que, se logró interceptar, modificar y alertar correctamente el tráfico generado en cada uno de los escenarios. Destacando que con esta experiencia se aprendió sobre el uso del IDS Snort junto con diferentes reglas que permiten notificar según lo requerido.

Como se mencionó en el informe, alertar las ocurrencias de tráfico anómalo es una gran herramienta, ya que nos permitirá reaccionar y solucionar a la brevedad la anomalía o posible vulneración, cabe destacar que el IDS utilizado fue SNORT, el cual inicialmente era un software libre mantenido por la comunidad hasta que fuera comprado por CISCO en el 2013 y sacara un version de paga, esto repercute en que la versión de snort usada en esta tarea fue la 2.9.6.2, qué es la última versión en github y esta no se actualiza hace 7 años, a pesar de todo, sigue siendo una increíble herramienta para la seguridad informática en la actualidad, por lo tanto sigue siendo una gran aplicación para aprender a utilizar.

Haciendo un análisis del curso, se puede entender la materia que se enseñó y los trabajos, en su respectivo momento, ya que en esta tarea de cierre primero se utiliza wireshark para capturar el tráfico (*primera tarea*), los contenedores en docker para generar el tráfico entre el cliente y servidor (*segunda tarea*), se modificaron los paquetes con polymorph para generar los escenarios anómalos (*tercera tarea*), y alertar tráfico anómalo, para reaccionar y mantener la seguridad en nuestra red con SNORT (*quinta tarea*), resultando en una gran torre de bloques de conocimientos, que se pueden utilizar de forma individual, como en un conjunto para futuros proyectos.

Link video demostrativo:

[https://youtu.be/qBR\\_M2snXeg](https://youtu.be/qBR_M2snXeg)

Link del GitLab:

[https://gitlab.com/tareas-taller-de-redes-y-servicios-2021-1/2021-2/grupo\\_19](https://gitlab.com/tareas-taller-de-redes-y-servicios-2021-1/2021-2/grupo_19)