

# Universidad de Alcalá Escuela Politécnica Superior

Máster Universitario en  
Ciberseguridad



ESCUELA POLITÉCNICA  
Autores:  
Aryan Rezaeian Hernández  
SUPERIOR

14 de Octubre de 2024



# Índice general

<b>1. Configuración de Snort con Reglas de Detección</b>	<b>2</b>
1.1. Instalación de Snort . . . . .	2
1.2. Configuración de snort . . . . .	3
1.3. Reglas Predeterminadas . . . . .	5
1.4. Reglas Personalizadas . . . . .	9
1.4.1. Primer contacto con Snort . . . . .	9
1.4.2. Definir 5 reglas personalizadas . . . . .	11
1.4.2.1. Primera regla creada . . . . .	12
1.4.2.2. Pruebas Realizadas Regla 1 . . . . .	13
1.4.2.3. Segunda regla creada . . . . .	14
1.4.2.4. Pruebas Realizadas Regla 2 . . . . .	15
1.4.2.5. Tercera regla creada parte 1 . . . . .	15
1.4.2.6. Pruebas Realizadas Regla 3.1 . . . . .	16
1.4.2.7. Tercera regla creada parte 2 . . . . .	17
1.4.2.8. Pruebas Realizadas Regla 3.2 . . . . .	20
1.4.2.9. Cuarta regla creada . . . . .	22
1.4.2.10. Pruebas Realizadas Regla 4 . . . . .	23
1.4.2.11. Quinta regla creada . . . . .	23
1.4.2.12. Pruebas Realizadas Regla 5 . . . . .	25
1.4.2.13. Preprocesadores . . . . .	26
1.4.2.14. Pruebas preprocesadores . . . . .	27
<b>2. Configuración de Suricata o Zeek</b>	<b>29</b>
2.1. Instalación de Suricata o Zeek . . . . .	29
2.2. Confiuración de Suricata o Zeek . . . . .	29
2.3. Reglas predeterminadas . . . . .	30
2.4. Creación de Reglas . . . . .	30
2.5. Funcionamiento . . . . .	32
<b>3. Conclusiones</b>	<b>36</b>
<b>Bibliografía</b>	<b>37</b>



# Capítulo 1

## Configuración de Snort con Reglas de Detección

Una vez instalado *VirtualBox* y creado una máquina virtual *ubuntu 24.04.1* lo primero que hacemos es actualizar el sistema:

```
sudo apt update && sudo apt upgrade -y
```

Una vez actualizado podemos comenzar con la instalación de *snort v2* en nuestra máquina con nombre **snort-aryan**.

### 1.1. Instalación de Snort

Lo primero que vamos a hacer va a ser instalar las dependencias necesarias, para evitar posibles errores. Comando utilizado:

```
sudo apt-get install build-essential autotools-dev  
libdumbnet-dev libluajit-5.1-dev libpcap-dev libpcre3-dev  
zlib1g-dev pkg-config libhwloc-dev cmake liblzma-dev  
openssl libssl-dev cpputest libsqlite3-dev libntirpc-dev  
rpcbind flex bison libtirpc-dev
```

```
aryan@snort-aryan:~$ sudo apt-get install build-essential autotools-dev libdumbnet-dev libluajit-5.1-dev libpcap-dev libpcre3-dev zlib1g-dev pkg-config libhwloc-dev cmake liblzma-dev openssl libssl-dev cpputest libsqlite3-dev libntirpc-dev  
rpcbind flex bison libtirpc-dev  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias... Hecho  
Leyendo la información de estado... Hecho  
build-essential ya está en su versión más reciente (12.10ubuntu1).  
autotools-dev ya está en su versión más reciente (20220109.1).  
libdumbnet-dev ya está en su versión más reciente (1.17.0-1ubuntu2).  
libluajit-5.1-dev ya está en su versión más reciente (2.1.0+git20231223.c525bcb+dfsg-1).  
libpcap-dev ya está en su versión más reciente (1.10.4-4.1ubuntu3).  
libpcre3-dev ya está en su versión más reciente (2:8.39-15build1).  
zlib1g-dev ya está en su versión más reciente (1:1.3.dfsg-3.1ubuntu2.1).  
pkg-config ya está en su versión más reciente (1.8.1-2build1).  
libhwloc-dev ya está en su versión más reciente (2.10.0-1build1).  
cmake ya está en su versión más reciente (3.28.3-1build7).  
liblzma-dev ya está en su versión más reciente (5.6.1+really5.4.5-1build0.1).  
openssl ya está en su versión más reciente (3.0.13-0ubuntu3.4).  
libssl-dev ya está en su versión más reciente (3.0.13-0ubuntu3.4).  
cpputest ya está en su versión más reciente (4.0-2).  
libsqlite3-dev ya está en su versión más reciente (3.45.1-1ubuntu2).  
libntirpc-dev ya está en su versión más reciente (4.3-3.1build2).  
rpcbind ya está en su versión más reciente (1.2.6-7ubuntu2).  
flex ya está en su versión más reciente (2.6.4-8.2build1).  
bison ya está en su versión más reciente (2:3.8.2+dfsg-1build2).  
libtirpc-dev ya está en su versión más reciente (1.3.4+ds-1.1build1).  
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 3 no actualizados.
```

Figura 1.1: Dependencias Instaladas

Una vez instaladas las dependencias necesarias volvemos a instalar snort. Para ello usamos el siguiente comando:



```
sudo apt install snort
```

Y una vez el comando anterior se completa probamos la instalación de **snort** comprobando su versión [1.2](#).

```
snort --version
```

```
aryan@snort-aryan:~$ snort --version
,,_ -*> Snort! <-
o" )~ Version 2.9.20 GRE (Build 82)
'``' By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2022 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.10.4 (with TPACKET_V3)
Using PCRE version: 8.39 2016-06-14
Using ZLIB version: 1.3
```

Figura 1.2: Versión de Snort

## 1.2. Configuración de snort

Una vez instalado **snort** vamos a comenzar con la configuración.

Vamos a editar el archivo de configuración **snort.conf** ubicado en */etc/snort/snort.conf*.

```
sudo nano /etc/snort/snort.conf
```

En el vamos a configurar la red e interfaz de red que queremos monitorizar.

Una vez hecho esto vamos a activar el modo promiscuo para capturar todo el tráfico en la red (no solo tráfico dirigido a nuestra máquina)

```
sudo ip link set enp0s3 promisc on
```

Para finalizar con la configuración de snort vamos a crear un directorio. Ya que el resto de directorios estan creados: log, rules etcétera.

```
sudo mkdir /etc/snort/preproc_rules
```

Antes de continuar vamos a definir la arquitectura de nuestro entorno, que se va a utilizar en la práctica. Vamos a tener una máquina externa atacante Kali Linux, un sistema ubuntu 24.04 con snort instalado analizando el trafico además existirá una máquina víctima metasploitable. La arquitectura se puede visualizar en la imagen [1.4](#)



```
aryan@snort-aryan: /etc/snort
GNU nano 7.2                                     snort.debian.conf
# snort.debian.config (Debian Snort configuration file)
#
# This file was generated by the post-installation script of the snort
# package using values from the debconf database.
#
# It is used for options that are changed by Debian to leave
# the original configuration files untouched.
#
# This file is automatically updated on upgrades of the snort package
# *only* if it has not been modified since the last upgrade of that package.
#
# If you have edited this file but would like it to be automatically updated
# again, run the following command as root:
#   dpkg-reconfigure snort

DEBIAN_SNORT_STARTUP="boot"
DEBIAN_SNORT_HOME_NET="192.168.1.0/24"
DEBIAN_SNORT_OPTIONS=
DEBIAN_SNORT_INTERFACE="enp0s3"
DEBIAN_SNORT_SEND_STATS="true"
DEBIAN_SNORT_STATS_RCPT="root"
DEBIAN_SNORT_STATS_THRESHOLD="1"
```

Figura 1.3: Editar archivo de configuración snort.debian.conf

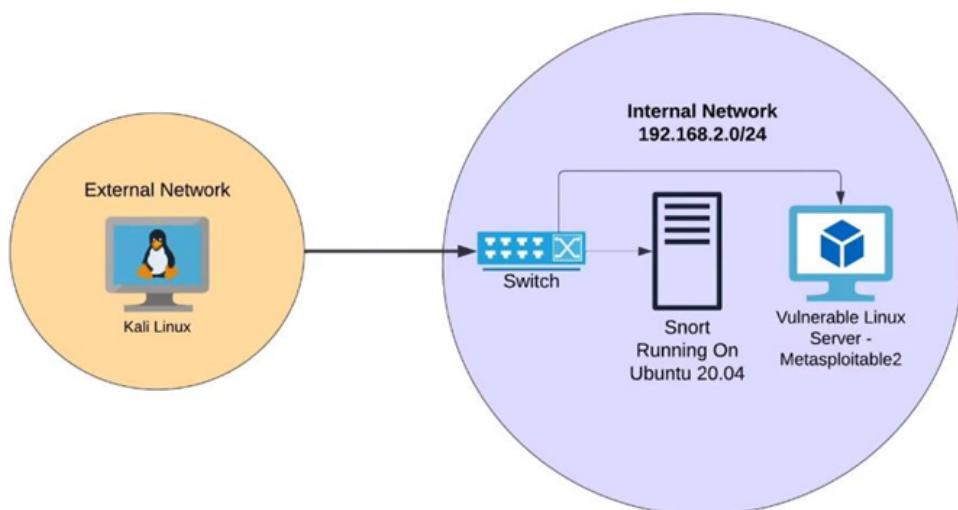


Figura 1.4: Arquitectura del entorno



### 1.3. Reglas Predeterminadas

Como podemos ver en la imagen 1.5 ya tenemos las reglas predefinidas en nuestro directorio rules, en caso de no tenerlas las tendríamos que descargar desde el sitio oficial.

```
wget https://www.snort.org/downloads/community/community-rules.tar.gz  
tar -xvzf community-rules.tar.gz
```

```
aryan@snort-aryan:/etc/snort/rules$ ls  
attack-responses.rules      community-nntp.rules      deleted.rules      netbios.rules      sql.rules  
backdoor.rules               community-oracle.rules    dns.rules        nntp.rules       telnet.rules  
bad-traffic.rules            community-policy.rules   dos.rules        oracle.rules     tftp.rules  
chat.rules                  community-sip.rules      experimental.rules other-ids.rules  virus.rules  
community-bot.rules          community-smtp.rules     exploit.rules    p2p.rules        web-attacks.rules  
community-deleted.rules     community-sql-injection.rules finger.rules    policy.rules     web-cgi.rules  
community-dos.rules          community-virus.rules    ftp.rules       pop2.rules      web-client.rules  
community-exploit.rules     community-web-attacks.rules icmp-info.rules  pop3.rules      web-coldfusion.rules  
community-ftp.rules          community-web-cgi.rules   icmp.rules      porn.rules      web-frontpage.rules  
community-game.rules         community-web-client.rules  imap.rules      rpc.rules      web-iis.rules  
community-icmp.rules         community-web-dos.rules   info.rules      rservices.rules  web-misc.rules  
community-imap.rules         community-web-iis.rules  local.rules     scan.rules      web-php.rules  
community-inappropriate.rules community-web-misc.rules misc.rules      shellcode.rules x11.rules  
community-mail-client.rules  community-web-php.rules  multimedia.rules smtp.rules  
community-misc.rules         ddos.rules           mysql.rules    snmp.rules
```

Figura 1.5: Reglas predefinidas de snort

Para que snort pueda utilizarlas tenemos que incluirlas en el archivo de configuración de snort, *snort.conf*

```
GNU nano 7.2                                         snort.conf  
include $RULE_PATH/snmp.rules  
#include $RULE_PATH/specific-threats.rules  
#include $RULE_PATH/spyware-put.rules  
include $RULE_PATH/sql.rules  
include $RULE_PATH/telnet.rules  
include $RULE_PATH/tftp.rules  
include $RULE_PATH/virus.rules  
#include $RULE_PATH/voip.rules  
#include $RULE_PATH/web-activex.rules  
include $RULE_PATH/web-attacks.rules  
include $RULE_PATH/web-cgi.rules  
include $RULE_PATH/web-client.rules  
include $RULE_PATH/web-coldfusion.rules  
include $RULE_PATH/web-frontpage.rules  
include $RULE_PATH/web-iis.rules  
include $RULE_PATH/web-misc.rules  
include $RULE_PATH/web-php.rules  
include $RULE_PATH/x11.rules  
include $RULE_PATH/community-sql-injection.rules  
include $RULE_PATH/community-web-client.rules  
include $RULE_PATH/community-web-dos.rules  
include $RULE_PATH/community-web-iis.rules  
include $RULE_PATH/community-web-misc.rules  
include $RULE_PATH/community-web-php.rules  
include $RULE_PATH/community-sql-injection.rules  
include $RULE_PATH/community-web-client.rules  
include $RULE_PATH/community-web-dos.rules  
include $RULE_PATH/community-web-iis.rules
```

Figura 1.6: Snort.conf reglas predefinidas



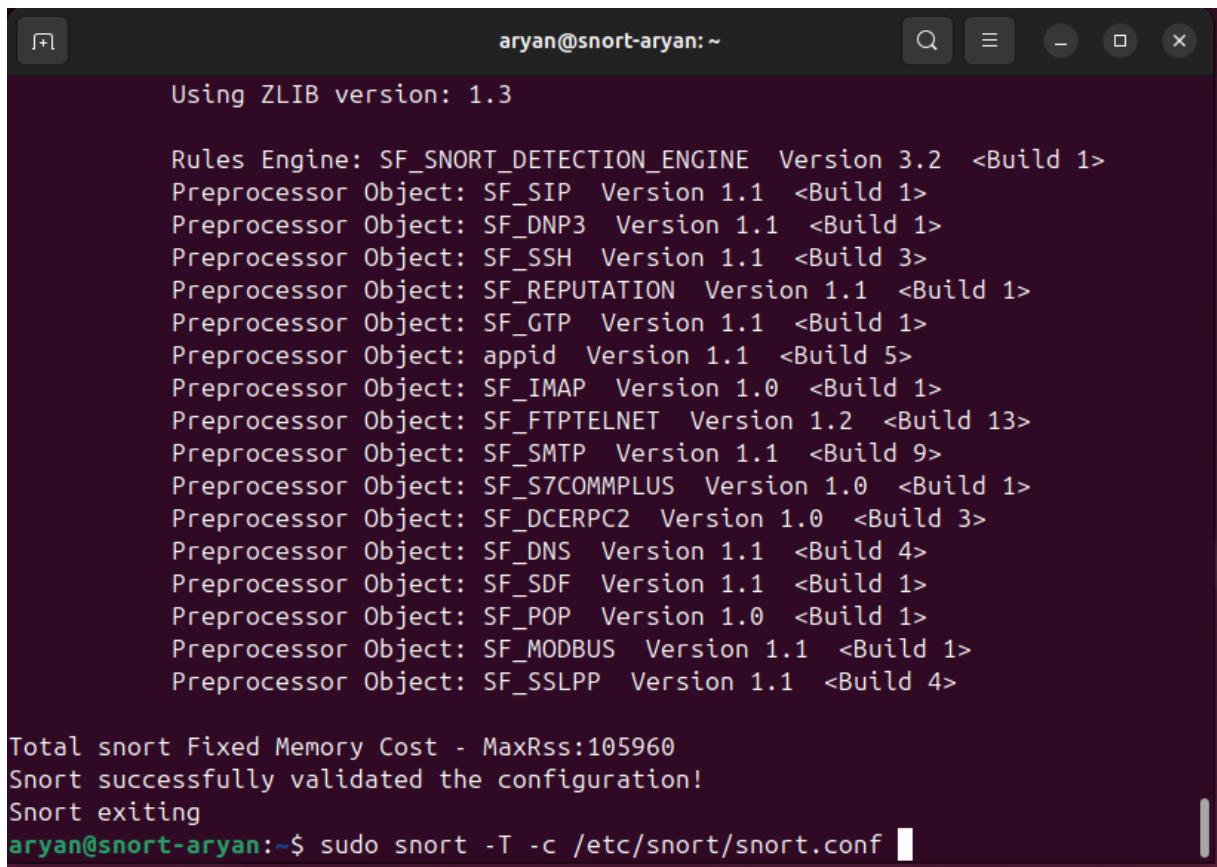
---

La sintaxis para incluir las reglas en el archivo de configuración, por ejemplo, para reglas sql es la siguiente:

```
include $RULE_PATH/sql.rules
```

Para comprobar que todo funciona correctamente vamos a lanzar el siguiente comando:

```
sudo snort -T -c /etc/snort/snort.conf
```



```
aryan@snort-aryan:~ Using ZLIB version: 1.3 Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.2 <Build 1> Preprocessor Object: SF_SIP Version 1.1 <Build 1> Preprocessor Object: SF_DNP3 Version 1.1 <Build 1> Preprocessor Object: SF_SSH Version 1.1 <Build 3> Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1> Preprocessor Object: SF_GTP Version 1.1 <Build 1> Preprocessor Object: appid Version 1.1 <Build 5> Preprocessor Object: SF_IMAP Version 1.0 <Build 1> Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13> Preprocessor Object: SF_SMTP Version 1.1 <Build 9> Preprocessor Object: SF_S7COMMPLUS Version 1.0 <Build 1> Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3> Preprocessor Object: SF_DNS Version 1.1 <Build 4> Preprocessor Object: SF_SDF Version 1.1 <Build 1> Preprocessor Object: SF_POP Version 1.0 <Build 1> Preprocessor Object: SF_MODBUS Version 1.1 <Build 1> Preprocessor Object: SF_SSLPP Version 1.1 <Build 4> Total snort Fixed Memory Cost - MaxRss:105960 Snort successfully validated the configuration! Snort exiting aryang@snort-aryan:~$ sudo snort -T -c /etc/snort/snort.conf
```

Figura 1.7: Validación de la configuración con las reglas predeterminadas

Una de las reglas comunes que se puede probar es detectar tráfico ICMP (como un ping), ya que es fácil de generar y las reglas predeterminadas suelen incluir detección de este tipo de tráfico.

Revisamos el archivo de configuración de snort y vemos que la regla predeterminada esta incluido ??.

Vamos a comentar el resto de reglas para poder asegurarnos que solo exista la regla predeterminada de ICMP. Comentamos la línea dynamicdetection, el procesador reputation con todas sus Líneas y todas las reglas restantes.



```
aryan@snort-aryan: ~
GNU nano 7.2          /etc/snort/snort.conf *
#include $RULE_PATH/dos.rules
#include $RULE_PATH/experimental.rules
#include $RULE_PATH/exploit-kit.rules
#include $RULE_PATH/exploit.rules
#include $RULE_PATH/file-executable.rules
#include $RULE_PATH/file-flash.rules
#include $RULE_PATH/file-identify.rules
#include $RULE_PATH/file-image.rules
#include $RULE_PATH/file-multimedia.rules
#include $RULE_PATH/file-office.rules
#include $RULE_PATH/file-other.rules
#include $RULE_PATH/file-pdf.rules
#include $RULE_PATH/finger.rules
#include $RULE_PATH/ftp.rules
# ICMP standard information queries will trigger these rules, they are very
# chatty, only enable if you need them
#include $RULE_PATH/icmp-info.rules
include $RULE_PATH/icmp.rules
#include $RULE_PATH/imap.rules
#include $RULE_PATH/indicator-compromise.rules
```

Figura 1.8: ICMP Rules

```
sudo snort -i enp0s3 -c /etc/snort/snort.conf -A console
```

- -i enp0s3: La interfaz de red que estás monitoreando.
- -c /etc/snort/snort.conf: Archivo de configuración principal.
- -A console: Muestra las alertas en tiempo real en la consola.

Para poder generar la alerta de una de las reglas predefinidas que existen en icmp.rules, como por ejemplo:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Source Quench"; icode:0; itype:
```

Primero tenemos que añadir unas líneas al archivo de configuración de snort para que no nos salte error:

Todo esto se podría solucionar descomentando la linea **include classification.config**

Lanzamos desde nuestro kali generamos tráfico ICMP con hping3. El tipo 4 de ICMP (Source Quench) es más difícil de encontrar en el tráfico estándar.

Y lanzamos snort con el siguiente comando:



```
# Event thresholding or suppression commands. See threshold.conf
include threshold.conf
config classification: attempted-recon, Attempted Information Leak, 2
config classification: attempted-dos, Attempted Denial of Service, 2
config classification: bad-unknown, Potentially Bad Traffic, 2
```

Figura 1.9: Error corregido

```
[kali㉿kali)-[~]
$ sudo hping3 --icmp --icmptype 4 192.168.1.168

HPING 192.168.1.168 (eth0 192.168.1.168): icmp mode set, 28 headers + 0 data
bytes
```

Figura 1.10: Hping3 icmp tipo 4

```
sudo snort -i enp0s3 -c /etc/snort/snort.conf -A console
```

Con el parametro **-i** especificamos la interfaz de red por la que snort va a estar escuchando.

Con **-c** especificamos la ruta del archivo de configuración de snort.

Y con **-A** indicamos donde queremos que se muestren las alertas, en las terminal.

Y vemos como nos salta la alerta de una de las reglas predeterminadas de snort:

```
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>
Commencing packet processing (pid=6143)

11/21-01:56:21.200971  [**] [1:477:2] ICMP Source Quench [**] [Classification: P
otentially Bad Traffic] [Priority: 2] {ICMP} 192.168.1.169 -> 192.168.1.168
11/21-01:56:22.204481  [**] [1:477:2] ICMP Source Quench [**] [Classification: P
otentially Bad Traffic] [Priority: 2] {ICMP} 192.168.1.169 -> 192.168.1.168
11/21-01:56:23.207661  [**] [1:477:2] ICMP Source Quench [**] [Classification: P
```

Figura 1.11: Alerta en snort de ICMP tipo 4

Una vez comprbado que funcionan las reglas predeterminadas de snort vamos a comentar todas para realizar nuestras propias reglas personalizadas.



## 1.4. Reglas Personalizadas

### 1.4.1. Primer contacto con Snort

Una vez configurado el entorno [1.4](#) vamos a hacer nuestro primer contacto con **snort**. Vamos a implementar una regla personalizada sencilla en **snort** para comprobar su funcionamiento.

Primero tenemos que saber cual es la sintaxis de las reglas [1.12](#).

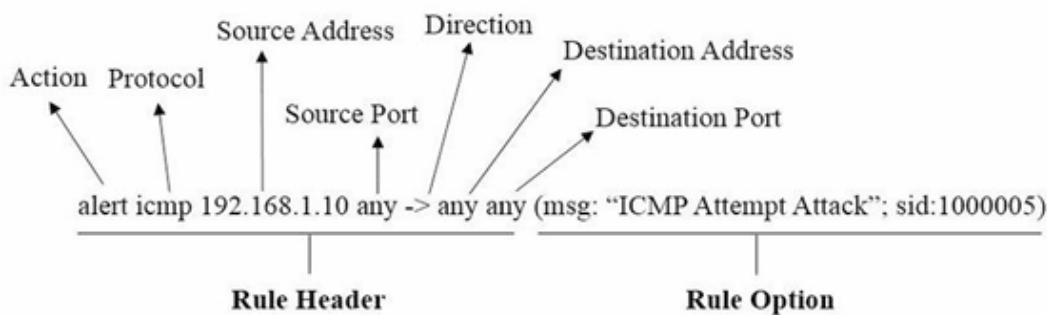


Figura 1.12: Sintaxis de las Reglas de Snort

Los tipos de **action** que existen en **snort** son:

- **alert**: Genera una alerta cuando se detecta una coincidencia en la regla.
- **log**: Guarda el tráfico que coincide con la regla en un archivo de registro específico.
- **pass**: ignorar el paquete.
- **drop**: Descarta el paquete que coincide con la regla y lo elimina del tráfico de red sin enviar ninguna respuesta. bloquear y registrar el paquete.
- **reject**: bloquear el paquete, registrarlo y luego enviar un reinicio de TCP si el protocolo es TCP o un mensaje de puerto ICMP inalcanzable si el protocolo es UDP.
- **sdrop**: Similar a drop, esta acción descarta el paquete sin generar ningún registro.

Para configurar **snort** tenemos que entrar en el archivo de configuración. Usamos el siguiente comando para editarlo:

```
nano /etc/snort/snort.conf
```

Una vez dentro creamos una regla siguiendo la sintaxis que se refleja en la imagen [1.12](#)

```
alert icmp any any -> 192.168.1.0/24 any  
(msg:"Alerta: Ping detectado hacia red interna"; itype:8; sid:1000001;)
```

El campo **itype** especifica el tipo de mensaje ICMP. El valor 8 corresponde a un ping de



---

solicitud (Echo Request). En otras palabras, esta regla está buscando paquetes ICMP que sean pings enviados hacia la red interna. El tipo 8 es utilizado cuando una máquina envía un ping para verificar la conectividad.

Insertamos la regla al final del archivo y guardamos como se muestra en la imagen 1.13

```
alert icmp any any -> 192.168.1.0/24 any (msg: "Alerta: Ping detectado hacia la >
```

Figura 1.13: Regla de alerta de pings

Para que las configuraciones que realizamos en snort se realicen hay que reiniciar el servicio.

```
sudo systemctl restart snort
```

Después de reiniciar el servicio de snort, lo activamos y probamos el funcionamiento de esta regla sencilla para ello vamos a lanzar un ping desde nuestra kali linux a la victima metasploitable.

Lanzamos snort como hemos explicado anteriormente.

```
sudo snort -i enp0s3 -c /etc/snort/snort.conf -A console
```

Lanzamos el ping desde nuestro kali.

```
ping 192.168.1.168
```

```
(kali㉿kali)-[~]
$ ping 192.168.1.168
PING 192.168.1.168 (192.168.1.168) 56(84) bytes of data.
64 bytes from 192.168.1.168: icmp_seq=1 ttl=64 time=3.59 ms
64 bytes from 192.168.1.168: icmp_seq=2 ttl=64 time=3.27 ms
64 bytes from 192.168.1.168: icmp_seq=3 ttl=64 time=2.50 ms
64 bytes from 192.168.1.168: icmp_seq=4 ttl=64 time=2.24 ms
64 bytes from 192.168.1.168: icmp_seq=5 ttl=64 time=3.87 ms
64 bytes from 192.168.1.168: icmp_seq=6 ttl=64 time=2.68 ms
64 bytes from 192.168.1.168: icmp_seq=7 ttl=64 time=3.64 ms
^C
--- 192.168.1.168 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6016ms
rtt min/avg/max/mdev = 2.244/3.114/3.871/0.588 ms
```

Figura 1.14: Ping desde el kali

Y detectamos el ping en nuestro snort.[1.15](#).



```
aryan@snort-aryan:~  
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>  
Preprocessor Object: SF_S7COMMPLUS Version 1.0 <Build 1>  
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>  
Preprocessor Object: SF_DNS Version 1.1 <Build 4>  
Preprocessor Object: SF_SDF Version 1.1 <Build 1>  
Preprocessor Object: SF_POP Version 1.0 <Build 1>  
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>  
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>  
Commencing packet processing (pid=6542)  
11/21-02:28:02.207758 [**] [1:1000001:0] Alerta: Ping detectado hacia la red interna [**] [Priority: 0] {ICMP} 192.168.1.169 -> 192.168.1.168  
11/21-02:28:03.208651 [**] [1:1000001:0] Alerta: Ping detectado hacia la red interna [**] [Priority: 0] {ICMP} 192.168.1.169 -> 192.168.1.168  
11/21-02:28:04.210672 [**] [1:1000001:0] Alerta: Ping detectado hacia la red interna [**] [Priority: 0] {ICMP} 192.168.1.169 -> 192.168.1.168  
11/21-02:28:05.218074 [**] [1:1000001:0] Alerta: Ping detectado hacia la red interna [**] [Priority: 0] {ICMP} 192.168.1.169 -> 192.168.1.168  
11/21-02:28:06.215692 [**] [1:1000001:0] Alerta: Ping detectado hacia la red interna [**] [Priority: 0] {ICMP} 192.168.1.169 -> 192.168.1.168  
11/21-02:28:07.218780 [**] [1:1000001:0] Alerta: Ping detectado hacia la red interna [**] [Priority: 0] {ICMP} 192.168.1.169 -> 192.168.1.168  
11/21-02:28:08.222907 [**] [1:1000001:0] Alerta: Ping detectado hacia la red interna [**] [Priority: 0] {ICMP} 192.168.1.169 -> 192.168.1.168
```

Figura 1.15: Detección de la relga simple personalizada

#### 1.4.2. Definir 5 reglas personalizadas

Lo primero que vamos a hacer va a ser crear un archivo local.rules y lo incluiremos en nuestro archivo de configuración.

```
sudo touch /etc/snort/rules/local.rules
```

```
aryan@snort-aryan:/etc/snort/rules$ cat local.rules  
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $  
# -----  
# LOCAL RULES  
# -----  
# This file intentionally does not come with signatures. Put your local  
# additions here.  
aryan@snort-aryan:/etc/snort/rules$
```

Figura 1.16: Archivo local.rules

Y entramos al archivo de configuración de snort y descomentamos esta línea [1.17](#):

Una vez hecho ya podemos añadir nuestras reglas personalizadas en el archivo de local.rules.



```
aryan@snort-aryan: /etc/snort
GNU nano 7.2          snort.conf *
# rules files that are available in your system (in the /etc/snort/rules
# directory)

# site specific rules
include $RULE_PATH/local.rules

# The include files commented below have been disabled
# because they are not available in the stock Debian
# rules. If you install the Sourcefire VRT please make
# sure you re-enable them again:

#include $RULE_PATH/app-detect.rules
#include $RULE_PATH/attack-responses.rules
#include $RULE_PATH/backdoor.rules
#include $RULE_PATH/bad-traffic.rules
#include $RULE_PATH/blacklist.rules
#include $RULE_PATH/botnet-cnc.rules
#include $RULE_PATH/browser-chrome.rules
#include $RULE_PATH/browser-firefox.rules
#include $RULE_PATH/browser-ie.rules
```

Figura 1.17: Incluir local.rules en snort.conf

#### 1.4.2.1. Primera regla creada

La primera regla que vamos a crear va a ser una regla para detectar escaneos de puertos (Port Scanning)

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
  (msg:"Alerta: Escaneo de puertos detectado";
   flags:S; threshold:type both, track by_src,
   count 20, seconds 60; classtype:attempted-recon; sid:1000002;)
```

El parametro **flags:S** Monitorea paquetes con el flag SYN activado.

**type both:** Actúa como un rate limiter y un detector de patrones. Lanza la alerta una vez al superar el umbral y sigue registrando eventos.

**track by\_src:** Rastrea la actividad por dirección IP de origen (por ejemplo, un atacante externo).

**count 20, seconds 60:** Lanza la alerta si detecta al menos 20 eventos dentro de un periodo de 60 segundos.



**classtype:attempted-recon** Clasifica la alerta en una categoría específica. Aquí, se clasifica como un intento de reconocimiento.

En resumen la regla detecta escaneos de puertos hacia la red interna. Utiliza el flag S (SYN) para identificar un intento de conexión, lo que es típico en un escaneo de puertos. Si se detectan 20 intentos de conexión en 60 segundos desde la misma dirección IP de origen, Snort generará una alerta.

```
GNU nano 7.2                                     local.rules
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# -----
# LOCAL RULES
# -----
# This file intentionally does not come with signatures. Put your local
# additions here.

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "Alerta: Escaneo de puertos >
```

Figura 1.18: Regla de escaneo de puertos

#### 1.4.2.2. Pruebas Realizadas Regla 1

Hemos lanzado un nmap desde nuestro kali-linux a nuestra máquina metasploitable.

A screenshot of a Kali Linux desktop environment within Oracle VM VirtualBox. The terminal window at the bottom left shows the output of a Nmap scan of the IP address 192.168.1.168, listing numerous open ports and their corresponding services. The desktop interface includes a file manager, a terminal window, and a central application center.

Figura 1.19: Prueba realizada nmap



---

Como podemos apreciar en nuestra imagen [1.19](#) se alerta el nmap.

#### 1.4.2.3. Segunda regla creada

La segunda regla que vamos a crear es una regla para detectar tráfico HTTP con parámetros sospechosos (SQL Injection).

Esta regla detecta patrones comunes en intentos de inyección SQL en solicitudes HTTP. La inyección SQL es una técnica utilizada por los atacantes para insertar código malicioso en bases de datos a través de formularios web o parámetros de URL.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80
(msg:"Alerta: Intento de inyección SQL detectado en HTTP";
content:'' OR 1=1--'; http_uri; classtype:attempted-user; sid:1000004;)
```

La opción **content: '' OR 1=1--''** Busca en el contenido del paquete HTTP la cadena exacta.

Con **http\_uri** Indica que la regla debe buscar la cadena de contenido en la URI de la solicitud HTTP. Esto significa que se examina específicamente la parte de la solicitud que contiene los parámetros de consulta.

Con **classtype:attempted-user** Clasifica la alerta como un intento de compromiso de una cuenta de usuario.

En resumen la regla detecta intentos de inyección SQL utilizando la cadena especificada, que es un patrón común utilizado para evadir autenticaciones o acceder a datos no autorizados. La cadena se busca en el URI de la solicitud HTTP.

Para que no nos salte un error vamos a añadir esta regla a nuestro archivo de configuración:

```
# Event thresholding or suppression commands. See threshold.conf
include threshold.conf
config classification: attempted-recon, Attempted Information Leak, 2
config classification: attempted-dos, Attempted Denial of Service, 2
config classification: bad-unknown, Potentially Bad Traffic, 2
config classification: misc-activity, Miscellaneous Activity, 3
config classification: attempted-user, Attempted User Privilege Gain, 2
```

Figura 1.20: Arreglar error Unknown ClassType: attempted-user



#### 1.4.2.4. Pruebas Realizadas Regla 2

Para poder probar esta regla vamos a enviar una solicitud con un SQLi a nuestra máquina víctima. Para ello vamos a usar DVWA que tiene abierto nuestro metasploitable.

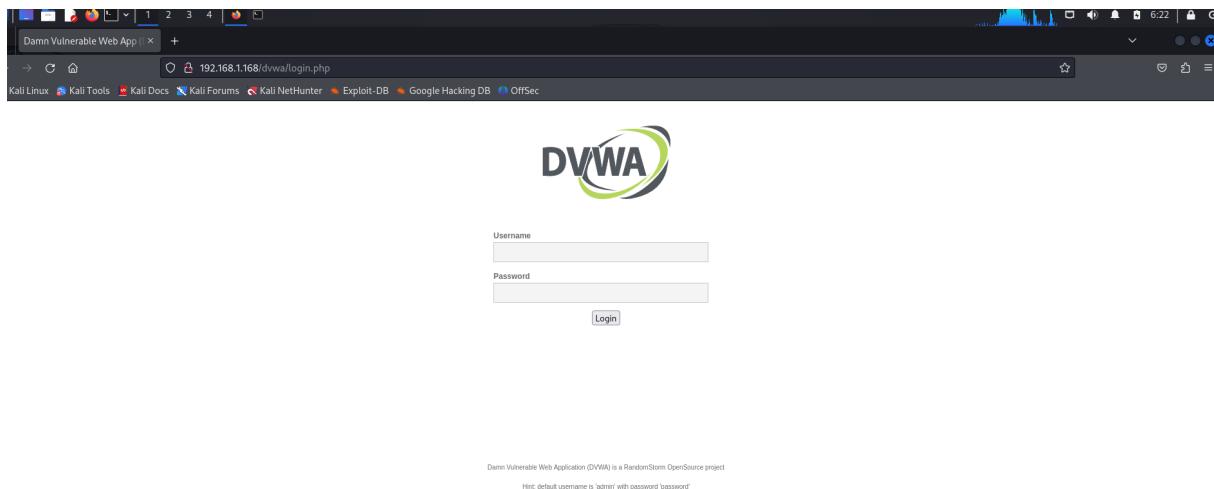


Figura 1.21: Login DVWA

Accedemos con **admin:password** [1.21](#) y una vez dentro nos adentramos a la pestaña de SQL injection e introducimos el SQLi, como podemos ver en la imagen ??

Como podemos ver en [1.23](#) snort nos ha detectado el intento de SQL injection.

#### 1.4.2.5. Tercera regla creada parte 1

Esta regla detecta un ataque de denegación de servicio distribuido (DDoS) basado en la sobrecarga de pings (ICMP Echo Requests). Este tipo de ataque puede generar una gran cantidad de tráfico ICMP que afecta la disponibilidad de un sistema.

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any
(msg:"Alerta: Ataque DDoS ICMP detectado";
itype:8; threshold:type both, track by_src, count 50,
seconds 5; classtype:denial-of-service; sid:1000005;)
```

La opción **itype: 8** especifica el tipo de mensaje ICMP que se está monitoreando

Con **threshold:type both** Define un umbral para generar alertas y lanza una alerta cuando se alcanza el umbral y continúa registrando eventos.

**track by\_src:** Rastrea la actividad por dirección IP de origen (por ejemplo, un atacante externo).



8/dvwa/vulnerabilities/sql\_injection/?id='admin+OR+1%3D1--&Submit=Submit#

Kali NetHunter Exploit-DB Google Hacking DB OffSec

DVWA

Vulnerability: SQL Injection (Blind)

User ID: ' OR 1=1--

More info

<http://www.secureteam.com/securityreviews/5DP0N1P76E.html>  
[http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)  
<http://www.unixwiz.net/t echtips/sql-injection.html>

Home Instructions Setup Brute Force Command Execution CSRF File Inclusion SQL Injection SQL Injection (Blind) Upload XSS reflected XSS stored DVWA Security PHP Info About Logout

Username: admin  
Security Level: high  
PHPIDS: disabled

View Source View Help

Damn Vulnerable Web Application (DVWA) v1.0.7

Figura 1.22: SQL injection

**count 50, seconds 5:** Lanza la alerta si detecta al menos 50 eventos dentro de un periodo de 5 segundos.

Con **classtype:denial-of-service** Clasifica la alerta como un intento de DoS.

En resumen esta regla detecta un posible ataque de denegación de servicio (DoS) basado en ICMP. Si se reciben más de 50 pings de tipo 8 en 5 segundos desde la misma IP de origen, se genera una alerta.

Primer vamos a probar esta alerta que funciona y despues vamos a añadir la acción de *drop* para bloquear el ataque.

#### 1.4.2.6. Pruebas Realizadas Regla 3.1

Para realizar esta prueba vamos a hacer uso de la herramienta **hping3**

```
sudo hping3 -i u1 -c 1000 --icmp 192.168.1.168
```

- **-i u1:** Envía paquetes con un intervalo de 1 microsegundo.
- **-c 1000:** Número de paquetes a enviar.



```
aryan@snort-aryan: /etc/snort/rules
Using ZLIB version: 1.3

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.2 <Build 1>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: appid Version 1.1 <Build 5>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_S7COMMPLUS Version 1.0 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>
Commencing packet processing (pid=4281)
11/21-12:47:45.527551 [**] [1:1000004:0] Intento de SQL detectado en HTTP [**]
[Classification: Attempted User Privilege Gain] [Priority: 2] {TCP} 192.168.1.16
9:60702 -> 192.168.1.168:80
```

Figura 1.23: Detección de SQLi en snort

```
# Se detallan aqui.
#Alerta de nmap
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "Alerta: Escaneo de puertos >
#Alerta de SQLi
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "Intento de SQL detectado en >
#Alerta y bloqueo de Dos
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg: "Alerta: Ataque DoS ICMP de >
#drop icmp $EXTERNAL_NET any -> $HOME_NET any (msg: "Bloqueo: Ataque DoS ICMP d>
```

Figura 1.24: Regla personalizada DoS

- **-icmp:** Envía paquetes ICMP.

Lanzamos el ataque desde nuestro Kali a nuestra máquina víctima.

y vemos como snort nos avisa con la alerta [1.25](#)

#### 1.4.2.7. Tercera regla creada parte 2

Snort puede funcionar en tres modos diferentes, passive, inline y en inline-test. Para poder conseguir que Snort nos bloquee tráfico tenemos que pasarlo al modo inline, para ello tenemos



```
(kali㉿kali)-[~]
$ sudo hping3 --icmp --icmptype 4 192.168.1.168

HPING 192.168.1.168 (eth0 192.168.1.168): icmp mode set, 28 headers + 0 data
bytes
```

Figura 1.25: Herramienta hping3

```
aryan@snort-aryan:/etc/snort
Using ZLIB version: 1.3

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.2 <Build 1>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: appid Version 1.1 <Build 5>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_S7COMMPLUS Version 1.0 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>

Commencing packet processing (pid=4635)
11/21-13:25:53.342119 [**] [1:1000005:0] Alerta: Ataque DoS ICMP detectado [**]
[Classification: Attempted Denial of Service Attack] [Priority: 2] {ICMP} 192.168.1.169 -> 192.168.1.168
```

Figura 1.26: Detección de alerta DoS

que instalarnos la dependencia de daq que aparece en la página oficial de snort.

```
wget https://www.snort.org/downloads/snort/daq-2.0.7.tar.gz
tar xvzf daq-2.0.7.tar.gz

cd daq-2.0.7
./configure && make && sudo make install
```

Una vez hecho eso vamos a editar el archivo de configuración de snort, para ello entramos en snort.conf y descomentamos las siguientes líneas y le ponemos lo siguiente [1.27](#):



```
aryan@snort-aryan:~
```

```
GNU nano 7.2          /etc/snort/snort.conf *
```

```
# Configure ports to ignore
# config ignore_ports: tcp 21 6667:6671 1356
# config ignore_ports: udp 1:17 53

# Configure active response for non inline operation. For more information, see>
# config response: eth0 attempts 2

# Configure DAQ related options for inline operation. For more information, see>
#
config daq: afpacket
config daq_dir: /usr/local/lib/daq
config daq_mode: inline
# config daq_var: <var>
#
# <type> ::= pcap | afpacket | dump | nfq | ipq | ipfw
# <mode> ::= read-file | passive | inline
# <var> ::= arbitrary <name>=<value passed to DAQ>
# <dir> ::= path as to where to look for DAQ module so's

# Configure specific UID and GID to run snort as after dropping privs. For more>

^G Ayuda      ^O Guardar    ^W Buscar     ^K Cortar     ^T Ejecutar   ^C Ubicación
^X Salir      ^R Leer fich. ^\ Reemplazar ^U Pegar      ^J Justificar ^/ Ir a línea
```

Figura 1.27: Configuración snort.cong para daq

**config daq:** Especificamos el tipo de captura que queremos usar. Por ejemplo, si deseamos usar el módulo afpacket para la captura de paquetes en modo inline

**config daq\_mode:** Define el modo en que Snort interactúa con DAQ. Los valores comunes son:

- inline: Modo de captura y prevención (Snort puede bloquear los paquetes).
- passive: Modo de solo detección (Snort solo detecta, no bloquea).

Snort en modo inline necesita una interfaz para recibir y procesar los paquetes (interfaz de captura) y otra interfaz para enviar los paquetes modificados o bloqueados (interfaz de salida). Esto se debe a que en modo inline, Snort está en el medio de la comunicación de red, monitoreando y potencialmente bloqueando paquetes.

Vamos a crear una nueva interfaz de red llamada enp0s8:

```
aryan@snort-aryan:~$ sudo ip link add enp0s8 link enp0s3 type macvlan mode bridge
aryan@snort-aryan:~$ ip a
```

Figura 1.28: Creación de nueva interfaz



```
aryan@snort-ryan:~$ sudo ip link set enp0s8 up
aryan@snort-ryan:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host noprefixroute
            valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:53:38:75 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.163/24 brd 192.168.1.255 scope global dynamic noprefixroute enp0s3
        valid_lft 77113sec preferred_lft 77113sec
    inet6 fd66:a0a4:17b:884c:5f0d:be91:9af9:c36c/64 scope global temporary dynamic
        valid_lft 1757sec preferred_lft 1757sec
    inet6 fd66:a0a4:17b:884c:a00:27ff:fe53:3875/64 scope global dynamic mngtmpaddr
        valid_lft 1757sec preferred_lft 1757sec
    inet6 fe80::a00:27ff:fe53:3875/64 scope link tentative
        valid_lft forever preferred_lft forever
3: enp0s8@enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 8e:70:fd:63:d1:01 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::8c70:fdff:fe63:d101/64 scope link tentative
        valid_lft forever preferred_lft forever
aryan@snort-ryan:~$
```

Figura 1.29: Ip a con dos interfaces

Ahora vamos a descomentar la regla de drop y vamos a probar su funcionamiento.

```
drop icmp $EXTERNAL_NET any -> $HOME_NET any
(msg:"Bloqueo: DDOS ICMP Flood Detected";
itype:8; threshold:type both, track by_src, count 100, seconds 10;
classtype:denial-of-service; sid:1000006; rev:2;)
```

Como hemos explicado antes esta regla detecta un posible ataque de denegación de servicio (DoS) basado en ICMP. Si se reciben más de 100 pings de tipo 8 en 10 segundos desde la misma IP de origen, se genera lo dropea.

#### 1.4.2.8. Pruebas Realizadas Regla 3.2

Para ver como funciona esta prueba vamos usar la herramienta hping3:

```
sudo hping3 --icmp -c 100000 -i u1 --flood 192.168.1.168
```

Y ejecutamos snort de la siguiente manera:

```
sudo snort -A console -Q -c /etc/snort/snort.conf -i enp0s3:enp0s8
```

Para poder hacer que snort funcione en modo inline a parte de crear la nueva interfaz tenemos que añadir el parametro: **-Q** para el modo inline **-i** definimos la interfaz de entrada y la de salida.

Como podemos observar en [1.32](#) los paquete recibido han sido una cantidad de 0 esto se debe a la acción drop de nuestro snort.



```
aryan@snort-aryan:~
```

```
GNU nano 7.2          /etc/snort/rules/local.rules *
```

```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
```

```
# -----
```

```
# LOCAL RULES
```

```
# -----
```

```
# This file intentionally does not come with signatures. Put your local
```

```
# additions here.
```

```
#Alerta de nmap
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "Alerta: Escaneo de puertos >
```

```
#Alerta de SQLi
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "Intento de SQL detectado en >
```

```
#Alerta y bloqueo de Dos
```

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg: "Alerta: Ataque DoS ICMP de >
```

```
drop icmp $EXTERNAL_NET any -> $HOME_NET any (msg: "Bloqueo: Ataque DoS ICMP de >
```

Figura 1.30: Regla personalizada DoS

```
aryan@snort-aryan:~
```

```
Using ZLIB version: 1.3
```

```
Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.2 <Build 1>
```

```
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
```

```
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
```

```
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
```

```
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
```

```
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
```

```
Preprocessor Object: appid Version 1.1 <Build 5>
```

```
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
```

```
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
```

```
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
```

```
Preprocessor Object: SF_S7COMMPLUS Version 1.0 <Build 1>
```

```
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
```

```
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
```

```
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
```

```
Preprocessor Object: SF_POP Version 1.0 <Build 1>
```

```
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
```

```
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>
```

```
Commencing packet processing (pid=10524)
```

```
Decoding Ethernet
```

```
11/21-15:03:53.188050 [Drop] [**] [1:1000006:2] Bloqueo: DoS ICMP Flood detectado
```

```
do [**] [Priority: 0] {ICMP} 192.168.1.169 -> 192.168.1.168
```

Figura 1.31: Acción drop



```
(kali㉿kali)-[~]
$ sudo hping3 --icmp -c 100000 -i u1 --flood 192.168.1.168
HPING 192.168.1.168 (eth0 192.168.1.168): icmp mode set, 28 headers + 0 data
bytes
hping in flood mode, no replies will be shown
^C
— 192.168.1.168 hping statistic —
34922 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
— (kali㉿kali)-[~]
```

Figura 1.32: Hping3 flood DoS

Además añadimos esta regla a nuestro snort.conf para evitar errores.

```
aryan@snoer-aryan:/etc/snort$ sudo cat snort.conf | tail
# include $SO_RULE_PATH/web-misc.rules

# Event thresholding or suppression commands. See threshold.conf
include threshold.conf
config classification: attempted-recon, Attempted Information Leak, 2
config classification: attempted-dos, Attempted Denial of Service, 2
config classification: bad-unknown, Potentially Bad Traffic, 2
config classification: misc-activity, Miscellaneous Activity, 3
config classification: attempted-user, Attempted User Privilege Gain, 2
config classification: denial-of-service, Attempted Denial of Service Attack, 2
aryan@snoer-aryan:/etc/snort$
```

Figura 1.33: Snoer.conf arreglo de error

Todo esto se podria solucionar decomentando la linea **include classification.config**

#### 1.4.2.9. Cuarta regla creada

Vamos a controlar el tráfico al que se puede acceder por ejemplo si alguien de la red interna accede a buscar alcohol o armas o contenido para adulto.

```
reject tcp $HOME_NET any -> $EXTERNAL_NET anywww.com
(msg:"Alerta: Acceso a contenido sensible detectado";
content:"alcohol"; http_uri; nocase; content:"armas";
http_uri; nocase; content:"porno"; http_uri; nocase;
classtype:web-application-activity; sid:1000020;)
```

La opción **content: ".alcohol"** Busca en el contenido del paquete HTTP la cadena exacta: alcohol, armas, porno.

Con **http\_uri** Indica que la regla debe buscar la cadena de contenido en la URI de la solicitud HTTP. Esto significa que se examina específicamente la parte de la solicitud que contiene los



---

parámetros de consulta.

Con **classtype:web-application-activity** Clasifica el evento como una actividad relacionada con aplicaciones web.

En resumen esta regla bloquea el acceso a contenido sensible que incluya palabras como ".alcohol", ".armas", o "porno.en las URIs de solicitudes HTTP desde la red interna hacia Internet.

```
aryan@snort-aryan:~$ sudo cat /etc/snort/rules/local.rules
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# -----
# LOCAL RULES
#
# This file intentionally does not come with signatures. Put your local
# additions here.
#ALerta de nmap
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "Alerta: Escaneo de puertos detectado"; flags: S; threshold:type both
, track_by_src, count 20, seconds 60; classtype:attempted-recon; sid:1000002;)
#ALerta de SQLi
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "Intento de SQL detectado en HTTP"; content: "' OR 1=1--"; http_uri;
classtype:attempted-user; sid:1000004;)
#Alerta y bloqueo de Dos
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg: "Alerta: Ataque DoS ICMP detectado"; itype: 8; threshold: type both,
track_by_src, count 50, seconds 5; classtype:denial-of-service; sid:1000005;)
drop icmp $EXTERNAL_NET any -> $HOME_NET any (msg: "Bloqueo: DoS ICMP Flood detectado"; itype: 8; threshold:type both, t
rack_by_src, count 100, seconds 10; sid:1000006; rev:2;)
#Control de acceso a internet
alert tcp any any -> any any (msg: "ALerta: Acceso a contenido sensible detectado"; content: "alcohol"; classtype:web-ap
plication-activity; sid: 1000030;)
```

Figura 1.34: Regla acceso a contenido sensible

#### 1.4.2.10. Pruebas Realizadas Regla 4

Para probar le funcionamiento de esta regla vamos a utilizar la herramienta curl:

```
curl -v http://www.alcohol.com/
```

Y como podemos ver nuestro snort nos detecta el acceso a la página.

#### 1.4.2.11. Quinta regla creada

Nuestra última regla va a consistir en detectar ataques de fuerza bruta en el protocolo SSH:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 22
(msg:"Alerta: Posible ataque de fuerza bruta SSH detectado";
flow:to_server,established; content:"SSH";
detection_filter:track by_src, count 5, seconds 60;
classtype:suspicious-login; sid:1000012; rev:2;)

drop tcp $EXTERNAL_NET any -> $HOME_NET 22
(msg:"Alerta: Posible ataque de fuerza bruta SSH detectado";
```



```
(kali㉿kali)-[~]
$ curl -v http://www.alcohol.com
* Host www.alcohol.com:80 was resolved.
* IPv6: 2600:1f14:ea2:fa00:d16c:e086:3be1:e745, 2600:1f14:ea2:fa01:5f63:807e:6c26:65db
* IPv4: 34.214.136.245, 35.82.214.86
* Trying 34.214.136.245:80 ...
* Connected to www.alcohol.com (34.214.136.245) port 80
> GET / HTTP/1.1
> Host: www.alcohol.com
> User-Agent: curl/8.8.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 302 Moved Temporarily
< Date: Thu, 21 Nov 2024 17:19:49 GMT
< Content-Type: text/html
< Content-Length: 154
< Connection: keep-alive
< Server: nginx/1.18.0 (Ubuntu)
< Location: https://venture.com/domains/alcohol.com/
<
<html>
<head><title>302 Found</title></head>
<body>
<center><h1>302 Found</h1></center>
<hr><center>nginx/1.18.0 (Ubuntu)</center>
</body>
</html>
* Connection #0 to host www.alcohol.com left intact
```

Figura 1.35: Curl

```
Preprocessor Object: SF_HOOBOS Version 1.1 <Build 1>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>
Commencing packet processing (pid=14795)
Decoding Ethernet
11/21-18:19:49.432802 [**] [1:1000030:0] Alerta: Acceso a contenido sensible detectado [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 192.168.1.169:48394 -> 34.214.136.245:80
11/21-18:19:49.596997 [**] [1:1000030:0] Alerta: Acceso a contenido sensible detectado [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 34.214.136.245:80 -> 192.168.1.169:48394
11/21-18:19:49.694545 [**] [1:1000030:0] Alerta: Acceso a contenido sensible detectado [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 34.214.136.245:80 -> 192.168.1.169:48394
```

Figura 1.36: Aleta acceso a contenido sensible

```
flow:to_server,established; content:"SSH";
detection_filter:track_by_src, count 10, seconds 60;
classtype:suspicious-login; sid:1000013; rev:2;)
```

Las opciones son **flow:to\_server,established**. El paquete debe estar dirigido hacia el servidor y la conexión debe estar establecida (es decir, debe haber completado el handshake TCP).

Con **content:"SSH"** Busca la cadena SSH en el contenido del paquete para confirmar que se trata de tráfico relacionado con el protocolo SSH.

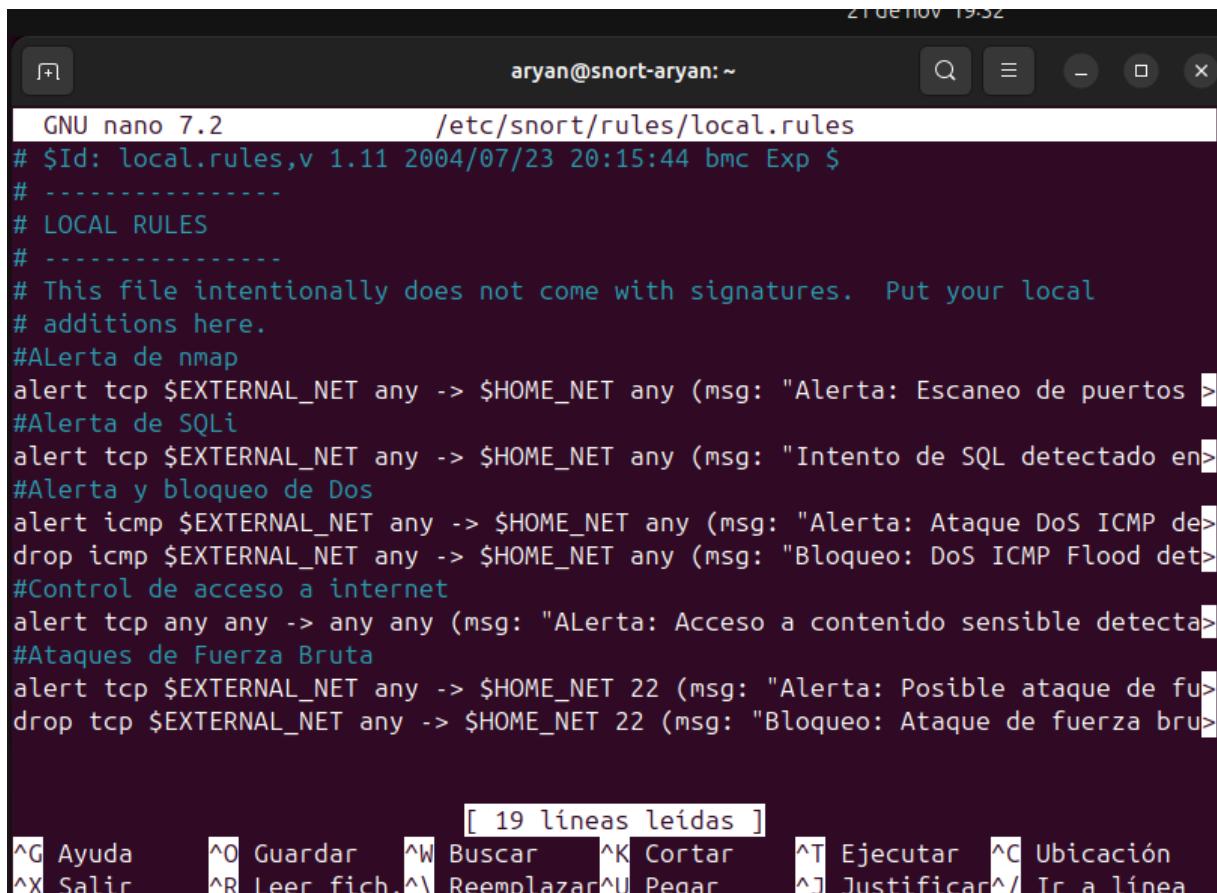
Con **detection\_filter : trackby\_src, count5, seconds60; Rastrea en modo de eventos por dirección IP de origen**

**classtype:suspicious-login** Clasifica el evento como un intento sospechoso de inicio de sesión. Estas dos reglas tienen el mismo cometido una de ellas alerta si ha habido



---

cinco intentos en 60 segundos y la otra dropa si hay ya 10 intentos en menos de 60 segundos.



```
GNU nano 7.2          /etc/snort/rules/local.rules
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# -----
# LOCAL RULES
# -----
# This file intentionally does not come with signatures. Put your local
# additions here.
#ALerta de nmap
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "Alerta: Escaneo de puertos >
#Alerta de SQLi
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "Intento de SQL detectado en >
#Alerta y bloqueo de Dos
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg: "Alerta: Ataque DoS ICMP de >
drop icmp $EXTERNAL_NET any -> $HOME_NET any (msg: "Bloqueo: DoS ICMP Flood det >
#Control de acceso a internet
alert tcp any any -> any any (msg: "ALerta: Acceso a contenido sensible detecta >
#Ataques de Fuerza Bruta
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg: "Alerta: Posible ataque de fu >
drop tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg: "Bloqueo: Ataque de fuerza bru >

[ 19 líneas leídas ]
^G Ayuda      ^O Guardar    ^W Buscar     ^K Cortar     ^T Ejecutar   ^C Ubicación
^X Salir      ^R Leer fich. ^\ Reemplazar^U Pegar      ^J Justificar^/ Ir a línea
```

Figura 1.37: Regla para detectar ataque de fuerza bruta

#### 1.4.2.12. Pruebas Realizadas Regla 5

Para esta parte vamos a utilizar la herramienta hydra: (Para esta parte vamos a utilizar otra víctima porque metasploitable tiene la versión muy antigua de ssh y no son compatibles los algoritmos)

```
hydra -L /usr/share/wordlists/top-usernames-shortlist.txt
-P /usr/share/wordlists/500-worst-passwords.txt ssh://192.168.1.172/
```

como podemos ver el intento de ataque de fuerza bruta, nos lo alerta y bloquea snort: [1.39](#)



```
(kali㉿kali)-[~]
└─$ hydra -L /usr/share/wordlists/top-usernames-shortlist.txt -P /usr/share/wordlists/500-worst-passwords.txt ssh://192.168.1.172/
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in
military or secret service organizations, or for illegal purposes (this is n
on-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-11-21 18:
57:57
[WARNING] Many SSH configurations limit the number of parallel tasks, it is r
ecommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 5142048 login tries (l:27
84/p:1847), ~321378 tries per task
[DATA] attacking ssh://192.168.1.172:22/
^CThe session file ./hydra.restore was written. Type "hydra -R" to resume ses
sion.

(aryan㉿kali)-[~]
```

Figura 1.38: Realización de fuerza bruta con hydra

```
aryan@snort-aryan:~
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: appid Version 1.1 <Build 5>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_S7COMMPLUS Version 1.0 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>
Commencing packet processing (pid=5006)
Decoding Ethernet
11/22-01:04:24.137196 [**] [1:1000002:0] Alerta: Escaneo de puertos detectado [**] [Classification: Attempted Informati
on Leak] [Priority: 2] {TCP} 192.168.1.160:53657 -> 192.168.1.1:53
11/22-01:04:37.384072 [**] [1:1000012:2] Alerta: Posible ataque de fuerza bruta en SSH detectado [**] [Classification:
An attempted login using a suspicious username was detected] [Priority: 2] {TCP} 192.168.1.177:40876 -> 192.168.1.172:22
11/22-01:04:37.384072 [**] [1:1000012:2] Alerta: Posible ataque de fuerza bruta en SSH detectado [**] [Classification:
An attempted login using a suspicious username was detected] [Priority: 2] {TCP} 192.168.1.177:40768 -> 192.168.1.172:22
11/22-01:04:37.382838 [**] [1:1000012:2] Alerta: Posible ataque de fuerza bruta en SSH detectado [**] [Classification:
An attempted login using a suspicious username was detected] [Priority: 2] {TCP} 192.168.1.177:40820 -> 192.168.1.172:22
11/22-01:04:37.384691 [**] [1:1000012:2] Alerta: Posible ataque de fuerza bruta en SSH detectado [**] [Classification:
An attempted login using a suspicious username was detected] [Priority: 2] {TCP} 192.168.1.177:40796 -> 192.168.1.172:22
11/22-01:04:37.384070 [**] [1:1000012:2] Alerta: Posible ataque de fuerza bruta en SSH detectado [**] [Classification:
An attempted login using a suspicious username was detected] [Priority: 2] {TCP} 192.168.1.177:40760 -> 192.168.1.172:22
11/22-01:04:37.384693 [Drop] [**] [1:1000013:2] Bloqueo: Ataque de fuerza bruta bloqueado [**] [Classification: An atte
mpted login using a suspicious username was detected] [Priority: 2] {TCP} 192.168.1.177:40776 -> 192.168.1.172:22
```

Figura 1.39: Detección y bloqueo de hydra

#### 1.4.2.13. Preprocesadores

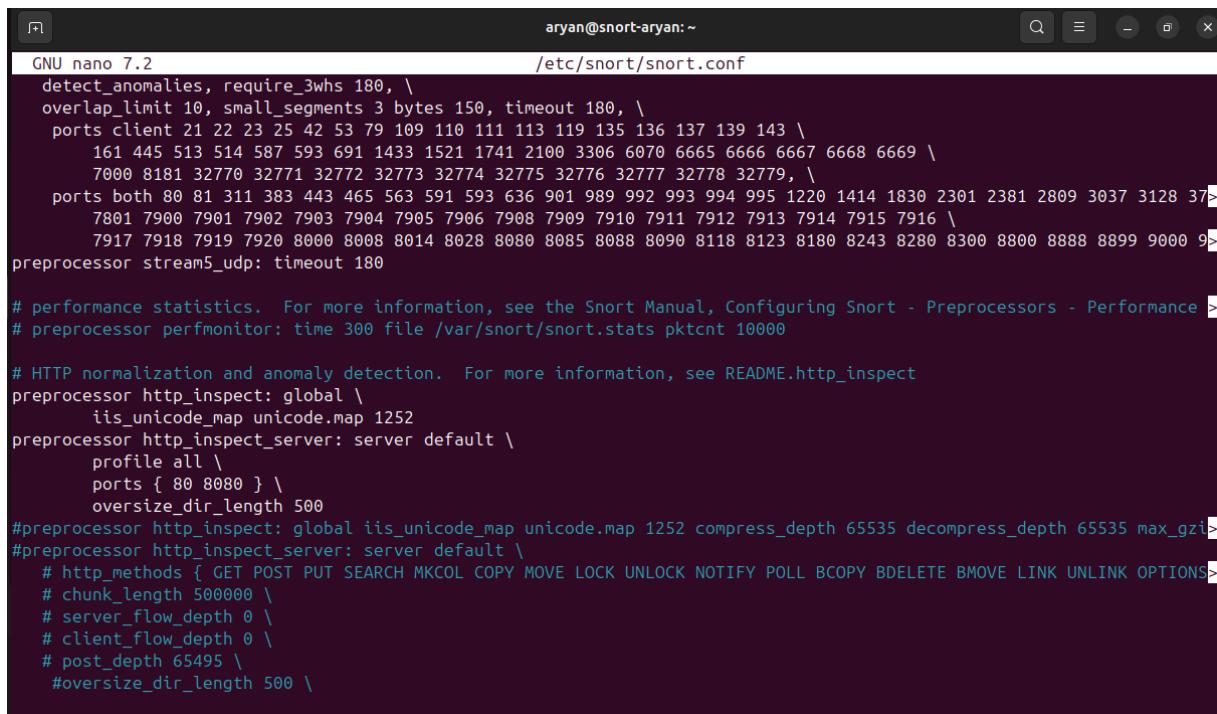
Para esta quinta regla vamos a hacer uso de los preprocesadores, los preprocesadores son módulos que analizan y manipulan el tráfico antes de que las reglas sean evaluadas. Son útiles para tareas como reensamblar flujos de paquetes, detectar



---

anomalías de protocolo, y manejar datos específicos como DNS o HTTP.

Para poder incluir los procesadores tenemos que editar el archivo de configuración de snort e introducir estas líneas [1.40](#):



```
GNU nano 7.2                               /etc/snort/snort.conf
detect_anomalies, require_3whs 180, \
overlap_limit 10, small_segments 3 bytes 150, timeout 180, \
ports client 21 22 23 25 42 53 79 109 110 111 113 119 135 136 137 139 143 \
161 445 513 514 587 593 691 1433 1521 1741 2100 3306 6070 6665 6666 6667 6668 6669 \
7000 8181 32770 32771 32772 32773 32774 32775 32776 32777 32778 32779, \
ports both 80 81 311 383 443 465 563 591 593 636 901 989 992 993 994 995 1220 1414 1830 2301 2381 2809 3037 3128 37>
7801 7900 7901 7902 7903 7904 7905 7906 7908 7909 7910 7911 7912 7913 7914 7915 7916 \
7917 7918 7919 7920 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180 8243 8280 8300 8800 8888 8899 9000 9>
preprocessor stream5_udp: timeout 180

# performance statistics. For more information, see the Snort Manual, Configuring Snort - Preprocessors - Performance >
# preprocessor perfmonitor: time 300 file /var/snort/snort.stats ptkcnt 10000

# HTTP normalization and anomaly detection. For more information, see README.http_inspect
preprocessor http_inspect: global \
    iis_unicode_map unicode.map 1252
preprocessor http_inspect_server: server default \
    profile all \
    ports { 80 8080 } \
    oversize_dir_length 500
#preprocessor http_inspect: global iis_unicode_map unicode.map 1252 compress_depth 65535 decompress_depth 65535 max_gzi>
#preprocessor http_inspect_server: server default \
    # http_methods { GET POST PUT SEARCH MKCOL COPY MOVE LOCK UNLOCK NOTIFY POLL BCOPY BDELETE BMOVE LINK UNLINK OPTIONS>
    # chunk_length 500000 \
    # server_flow_depth 0 \
    # client_flow_depth 0 \
    # post_depth 65495 \
    #oversize_dir_length 500 \
```

Figura 1.40: Líneas para incluir preprocesadores

Este preprocesador analiza el tráfico HTTP en los puertos 80 y 8080 para detectar anomalías. Vamos a crear una regla para probar el preprocesador, esta regla detecta intentos de inyección de comandos en solicitudes HTTP (por ejemplo, /etc/passwd o ; ls).

Vamos a editar el archivo local.rules

```
alert tcp any any -> any 80
(msg:"Intento de inyección detectado";
content:"/etc/passwd"; nocase; http_uri; sid:1000015;)
```

nocase Hace que la búsqueda del contenido no distinga entre mayúsculas y minúsculas.

#### 1.4.2.14. Pruebas preprocesadores

Vamos a hacer uso de la herramienta curl:



---

```
curl -X GET http://192.168.1.168/etc/passwd
```



# Capítulo 2

## Configuración de Suricata o Zeek

Una vez instalado *VirtualBox* y creado una máquina virtual *ubuntu 24.04.1* lo primero que hacemos es actualizar el sistema:

```
sudo apt update && sudo apt upgrade -y
```

Una vez actualizado podemos comenzar con la instalación de *suricata* en nuestra máquina con nombre *suricata-aryan*.

### 2.1. Instalación de Suricata o Zeek

Vamos a instalar dependencias como con snort [1.1](#).

Una vez instaladas ejecutamos el comando:

```
sudo apt install suricata
```

Y una vez el comando anterior se completa probamos la instalacion de suricata comprobando la versión ??

```
suricata -V
```

```
aryan@suricata-aryan:~$ suricata -V
This is Suricata version 7.0.3 RELEASE
aryan@suricata-aryan:~$
```

Figura 2.1: Versión de Suricata

### 2.2. Confiuración de Suricata o Zeek

Una vez que Suricata esté instalado, vamos a empezar a configurar el archivo principal de configuración de Suricata, que se encuentra en */etc/suricata/suricata.yaml*.

Modificamos la ruta de las reglas para poner */etc/suricata/rules* Y añadimos el archivo de *local.rules* que hemos creado para poner nuestras reglas. Y por ultimo modificamos la interfaz. Y para que los cambios se apliquen tenemos que reiniciar el servicio.



```
aryan@suricata-aryan: /etc/suricata
GNU nano 7.2 suricata.yaml *
# See Napatech NTPL documentation other hashmodes and details on their use.
#
# This parameter has no effect if auto-config is disabled.
#
hashmode: hash5tuplesorted

##
## Configure Suricata to load Suricata-Update managed rules.
##

default-rule-path: /etc/suricata/rules

rule-files:
- suricata.rules
- local.rules
##
## Auxiliary configuration files.
##

classification-file: /etc/suricata/classification.config

^G Ayuda      ^O Guardar    ^W Buscar     ^K Cortar     ^T Ejecutar   ^C Ubicación
^X Salir      ^R Leer fich. ^\ Reemplazar ^U Pegar      ^J Justificar ^/ Ir a linea
```

Figura 2.2: Suricata-yaml

```
sudo systemctl restart suricata.service
```

### 2.3. Reglas predeterminadas

Para añadirlas simplemente modificamos le archivo de configuración y elegimos unas de las que tenemos dentro del directorio /etc/suricata/rules, como por ejemplo: ssh-events.rules

Después hay que reiniciar el servicio y ya estaria actualizado.

### 2.4. Creación de Reglas

El formato de las reglas de suricata es:

Las reglas se crean igual que en snort [1.12](#) Las reglas de Suricata son las que definen cómo detectar ciertos patrones de tráfico en la red. Estas reglas se almacenan en archivos con extensión .rules, y se encuentran típicamente en /etc/suricata/rules/ como en snort



```
aryan@suricata-aryan:/etc/suricata
```

```
GNU nano 7.2 suricata.yaml *
```

```
# See Napatech NTPL documentation other hashmodes and details on their use.
#
# This parameter has no effect if auto-config is disabled.
#
hashmode: hash5tuplesorted

##
## Configure Suricata to load Suricata-Update managed rules.
##

default-rule-path: /etc/suricata/rules

rule-files:
- suricata.rules
- local.rules
- ssh-events.rules

##
## Auxiliary configuration files.
##
```

Figura 2.3: Añadir regla predeterminada suricata-yaml

- **Action:** Determina que sucede cuando una firma hace match.
- **Header:** Define el protocolo, dirección IP, puertos y la dirección de la regla.
- **Rule options:** Define las especificaciones de la regla.
- El siguiente es un ejemplo:
  - `drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)"; flow:established,to_server; flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK .*USA.*[0-9]{3,}/i"; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124; rev:2;)`

Figura 2.4: Sintaxis firmas

En esa carpeta creamos un archivo llamado local.rules Hemos creado tres reglas para detectar un ping, un nmap y un DoS.

```
alert icmp any any -> any any (msg:"Ping ICMP detectado";
itype:8; sid:1000002; rev:1;)

alert tcp any any -> any any
```



```
(msg:"Possible escaneo de puertos TCP detectado";
flags:S; threshold:type both, track by_src, count 20, seconds 10;
sid:1000001; rev:1;)

alert tcp any any -> $HOME_NET 80 (msg: "Possible DDoS attack"; flags: S;
flow: stateless; threshold: type both, track by_dst, count 200, seconds 1;
sid:1000001; rev:1;)
```

Todas estas reglas se explicaron anteriormente.

```
aryan@suricata-aryan:~$ sudo cat /etc/suricata/rules/local.rules
#reglas
alert tcp any any -> $HOME_NET any (msg: "Alerta: posible nmap detectado"; flags
:S; threshold: type both, track by_src, count 20, seconds 10; sid: 1000001; rev:
1;)
alert icmp any any -> $HOME_NET any (msg: "Ping ICMP detectado"; itype:8; sid:10
00002; rev:1;)
alert tcp any any -> $HOME_NET 80 (msg: "Posible ataque DDoS"; flags: S; flow: s
tateless; threshold: type both, track by_dst, count 200, seconds 1; sid: 1000004
; rev:1;)
aryan@suricata-aryan:~$
```

Figura 2.5: Reglas Suricata

## 2.5. Funcionamiento

Para lanzar suricata tenemos que usar este comando:

```
sudo suricata -c /etc/suricata/suricata.yaml -i enp0s3
```

Al lanzar suricata nos sale este error, vamos a modificar el archivo de configuración:

```
aryan@suricata-aryan:/etc/suricata$ sudo suricata -c /etc/suricata/suricata.yaml
-i enp0s3
i: suricata: This is Suricata version 7.0.3 RELEASE running in SYSTEM mode
E: af-packet: fanout not supported by kernel: Kernel too old or cluster-id 99 al
ready in use.
i: threads: Threads created -> W: 1 FM: 1 FR: 1   Engine started.
^C: suricata: Signal Received. Stopping engine.
i: device: enp0s3: packets: 48, drops: 0 (0.00%), invalid checksum: 0
aryan@suricata-aryan:/etc/suricata$
```

Figura 2.6: Error al lanzar suricata

Vamos a modificar el `cluster_id` para evitar este error poniéndole el valor 100 por ejemplo. [2.7](#)



```
aryan@suricata-aryan:/etc/suricata
```

```
GNU nano 7.2 suricata.yaml *
```

```
# Linux high speed capture support
af-packet:
  - interface: enp0s3
    # Number of receive threads. "auto" uses the number of cores
    #threads: auto
    # Default clusterid. AF_PACKET will load balance packets based on flow.
    cluster-id: 100
    # Default AF_PACKET cluster type. AF_PACKET can load balance per flow or per hash.
    # This is only supported for Linux kernel > 3.1
    # possible value are:
    # * cluster_flow: all packets of a given flow are sent to the same socket
    # * cluster_cpu: all packets treated in kernel by a CPU are sent to the same socket
    # * cluster_qm: all packets linked by network card to a RSS queue are sent to the same
    # socket. Requires at least Linux 3.14.
    # * cluster_ebpf: eBPF file load balancing. See doc/userguide/capture-hardware/ebpf-xdp.rst for
    # more info.
    # Recommended modes are cluster_flow on most boxes and cluster_cpu or cluster_qm on system
    # with capture card using RSS (requires cpu affinity tuning and system IRQ tuning)
    # cluster_rollover has been deprecated; if used, it'll be replaced with cluster_flow.
    cluster-type: cluster_flow
    # In some fragmentation cases, the hash can not be computed. If "defrag" is set
    # to yes, the kernel will do the needed defragmentation before sending the packets.
    defrag: yes
    # To use the ring feature of AF_PACKET, set 'use-mmap' to yes
    #use-mmap: yes
    # Lock memory map to avoid it being swapped. Be careful that over
    # subscribing could lock your system
    #mmap-locked: yes
```

Figura 2.7: Cluster id

Guardamos y reiniciamos el servicio y volvemos a lanzar suricata. Pero no conseguimos arreglarlo así que lo ignoramos porque parece que funciona todo bien.

Recordamos habilitar el modo promiscuo para esta máquina, porque si no no podremos realizar las pruebas.

Desde nuestra máquina kali lanzamos un nmap, un ping y un hping3 para que nos salte la alerta en suricata.

Para poder ver las alertas nos metemos en el archivo /var/log/suricata/fast.log

Como podemos ver en [2.11](#) las tres reglas que hemos creado se han activado, por lo que podemos dar por finalizada la prueba de funcionamiento de reglas.



```
aryan@suricata-aryan:~$ sudo suricata -c /etc/suricata/suricata.yaml -i enp0s3 -v
Notice: suricata: This is Suricata version 7.0.3 RELEASE running in SYSTEM mode
Info: cpu: CPUs/cores online: 2
Info: suricata: Setting engine mode to IDS mode by default
Info: exception-policy: master exception-policy set to: auto
Info: ioctl: enp0s3: MTU 1500
Info: logopenfile: fast output device (regular) initialized: fast.log
Info: logopenfile: eve-log output device (regular) initialized: eve.json
Info: logopenfile: stats output device (regular) initialized: stats.log
Info: detect: 3 rule files processed. 5 rules successfully loaded, 0 rules failed, 0
Info: threshold-config: Threshold config parsed: 0 rule(s) found
Info: detect: 5 signatures processed. 0 are IP-only rules, 0 are inspecting packet payload, 3 inspect application layer, 0 are decoder event only
Error: af-packet: fanout not supported by kernel: Kernel too old or cluster-id 98 already in use.
Info: runmodes: enp0s3: creating 1 thread
Info: unix-manager: unix socket '/var/run/suricata-command.socket'
Notice: threads: Threads created -> W: 1 FM: 1 FR: 1 Engine started.
```

Figura 2.8: Suricata lanzado

The image shows two terminal windows side-by-side. The left window displays the output of an Nmap scan for host 192.168.1.168, showing various open ports and services. The right window shows a continuous stream of Suricata log entries, primarily ICMP traffic, indicating the engine is active and processing network packets.

```
$ nmap 192.168.1.168
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-11-21 20:31 EST
Nmap scan report for 192.168.1.168 (192.168.1.168)
Host is up (0.0050s latency).
Not shown: 977 closed tcp ports (conn-refused)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
5432/tcp  open  postgresql
5900/tcp  open  vnc
```

```
64 bytes from 192.168.1.168: icmp_seq=169 ttl=64 time=1.20 ms
64 bytes from 192.168.1.168: icmp_seq=170 ttl=64 time=1.00 ms
64 bytes from 192.168.1.168: icmp_seq=171 ttl=64 time=1.00 ms
64 bytes from 192.168.1.168: icmp_seq=172 ttl=64 time=1.58 ms
64 bytes from 192.168.1.168: icmp_seq=173 ttl=64 time=0.765 ms
64 bytes from 192.168.1.168: icmp_seq=174 ttl=64 time=0.902 ms
64 bytes from 192.168.1.168: icmp_seq=175 ttl=64 time=2.56 ms
64 bytes from 192.168.1.168: icmp_seq=176 ttl=64 time=0.914 ms
64 bytes from 192.168.1.168: icmp_seq=177 ttl=64 time=1.10 ms
64 bytes from 192.168.1.168: icmp_seq=178 ttl=64 time=1.07 ms
64 bytes from 192.168.1.168: icmp_seq=179 ttl=64 time=1.38 ms
64 bytes from 192.168.1.168: icmp_seq=180 ttl=64 time=1.44 ms
64 bytes from 192.168.1.168: icmp_seq=181 ttl=64 time=0.936 ms
64 bytes from 192.168.1.168: icmp_seq=182 ttl=64 time=1.27 ms
64 bytes from 192.168.1.168: icmp_seq=183 ttl=64 time=1.06 ms
64 bytes from 192.168.1.168: icmp_seq=184 ttl=64 time=1.30 ms
64 bytes from 192.168.1.168: icmp_seq=185 ttl=64 time=0.991 ms
64 bytes from 192.168.1.168: icmp_seq=186 ttl=64 time=1.28 ms
64 bytes from 192.168.1.168: icmp_seq=187 ttl=64 time=0.830 ms
64 bytes from 192.168.1.168: icmp_seq=188 ttl=64 time=0.721 ms
64 bytes from 192.168.1.168: icmp_seq=189 ttl=64 time=0.960 ms
64 bytes from 192.168.1.168: icmp_seq=190 ttl=64 time=1.34 ms
64 bytes from 192.168.1.168: icmp_seq=191 ttl=64 time=1.40 ms
64 bytes from 192.168.1.168: icmp_seq=192 ttl=64 time=0.802 ms
64 bytes from 192.168.1.168: icmp_seq=193 ttl=64 time=1.03 ms
64 bytes from 192.168.1.168: icmp_seq=194 ttl=64 time=1.19 ms
```

Figura 2.9: Pruebas Suricata

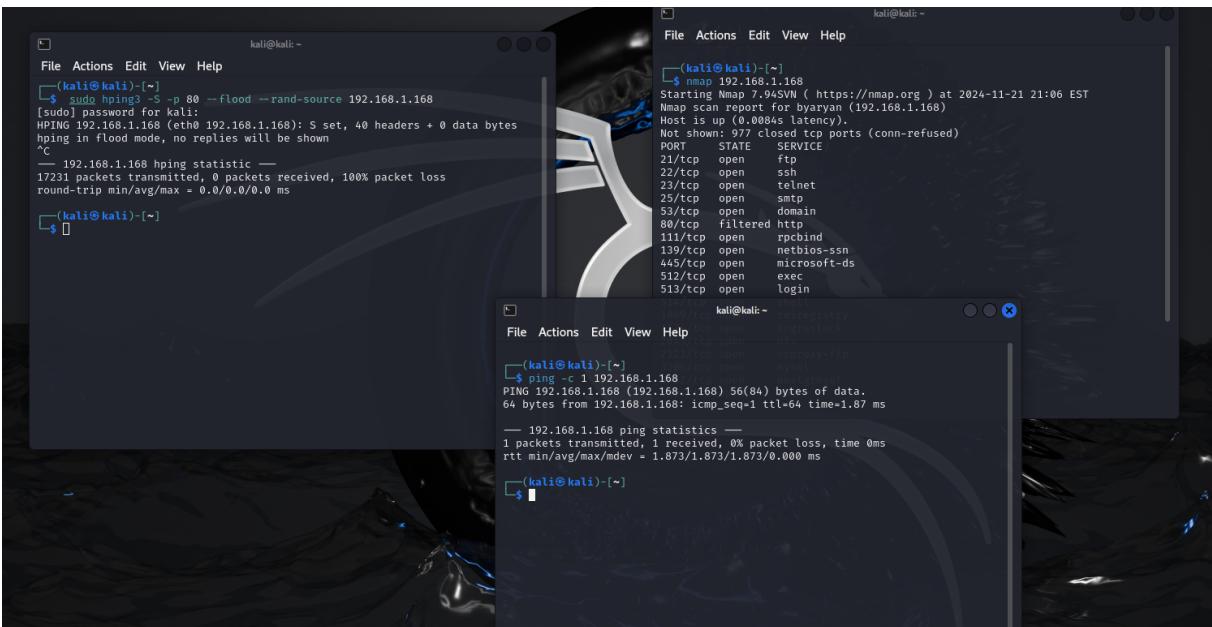


Figura 2.10: Pruebas realizadas con kali

A terminal window titled "aryan@suricata-aryan:/var/log/suricata" displaying log entries from the Suricata IDS/IPS system. The logs show various detection events, primarily related to ICMP and TCP traffic, indicating possible DDoS attacks and nmap scans. Key entries include:

```
ication: (null)] [Priority: 3] {ICMP} 192.168.1.177:8 -> 192.168.1.168:0
11/22/2024-02:42:53.173111 [**] [1:1000002:1] Ping ICMP detectado [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.177:8 -> 192.168.1.168:0
11/22/2024-02:42:53.173096 [**] [1:1000002:1] Ping ICMP detectado [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.177:8 -> 192.168.1.168:0
11/22/2024-03:06:52.263476 [**] [1:1000002:1] Ping ICMP detectado [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.177:8 -> 192.168.1.168:0
11/22/2024-03:06:52.263533 [**] [1:1000002:1] Ping ICMP detectado [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.1.177:8 -> 192.168.1.168:0
11/22/2024-03:06:59.746799 [**] [1:1000004:1] Posible ataque DDoS [**] [Classification: (null)] [Priority: 3] {TCP} 106.103.16.39:2814 -> 192.168.1.168:80
11/22/2024-03:06:59.940249 [**] [1:1000001:1] Alerta: posible nmap detectado [*] [Classification: (null)] [Priority: 3] {TCP} 192.168.1.177:45766 -> 192.168.1.168:8080
11/22/2024-03:06:59.940174 [**] [1:1000001:1] Alerta: posible nmap detectado [*] [Classification: (null)] [Priority: 3] {TCP} 192.168.1.177:47030 -> 192.168.1.168:8888
11/22/2024-03:07:00.756679 [**] [1:1000004:1] Posible ataque DDoS [**] [Classification: (null)] [Priority: 3] {TCP} 87.249.115.251:5184 -> 192.168.1.168:80
11/22/2024-03:07:01.754032 [**] [1:1000004:1] Posible ataque DDoS [**] [Classification: (null)] [Priority: 3] {TCP} 85.191.100.112:7580 -> 192.168.1.168:80
11/22/2024-03:07:02.769343 [**] [1:1000004:1] Posible ataque DDoS [**] [Classification: (null)] [Priority: 3] {TCP} 116.166.32.226:10095 -> 192.168.1.168:80
11/22/2024-03:07:03.771847 [**] [1:1000004:1] Posible ataque DDoS [**] [Classification: (null)] [Priority: 3] {TCP} 116.166.32.226:10095 -> 192.168.1.168:80
```

Figura 2.11: logs suricata



# Capítulo 3

## Conclusiones

Suricata tiene un funcionamiento diferente en cuanto a la detección de aplicaciones, además puede detectar tráfico HTTP o SSH en puertos no estándar basado en protocolos, para después aplicar configuraciones específicas al protocolo en esas direcciones. El principal beneficio de Suricata es multihilo, debido a que fue desarrollado recientemente, Suricata cuenta con todas las herramientas para tomar ventaja de multithreading, Snort a diferencia siempre usará un solo núcleo, no importa cuantos haya presentes. Suricata permite extracción de archivos, cuando la regla “filestore” es disparada, Suricata puede extraer todos los contenidos a un directorio para ser analizado después.



# Bibliografía

[1] Snort, “Snort 2,” 2024. Versión 2.9.20.