



Universidad
de Alcalá

Ejercicio IDS

Máster en ciberseguridad

Asignatura: Sistemas de información para la
ciberseguridad

Kevin van Liebergen



2020

1 Abstract

En esta práctica se va a aprender a configurar dos tipos de IDS, adquiriendo un conocimiento sobre preprocesadores, reglas, codificadores. La primera parte se va a enfocar en la instalación, configuración, generación de reglas e implementación de preprocesadore para el IDS Snort, mientras que en la segunda parte se va a proceder a realizar los mismos pasos pero para el IDS Suricata, para finalizar se van a exponer las principales diferencias entre ambos IDS junto con unas conclusiones.

2 Parte 1

El alumno deberá elegir un preprocesador distinto al comentado en la transparencia anterior que contenga:

- Una memoria con explicación detallada del mismo.
- Diferentes opciones de configuración
- Imágenes de captura.
- Con ejemplos demostrativos de un entorno configurado por el alumno
- Las capturas de imágenes debe aparecer nombre del alumno en el prompt del sistema, para comprobar la autenticidad del mismo

2.1. Introducción

Un IDS (*Intrusion Detection System*) es un sistema que actúa como un *sniffer* para analizar el tráfico de la red y poder descubrir posibles peligros y amenazas. Cualquier posible amenaza será reportada por un IDS hacia un operario que podrá decidir con dicha amenaza.

Existen dos tipos de IDS:

1. Network Intrusion Detection System (NIDS)
2. Host Intrusion Detecion System (HIDS)

Sin embargo un IDS necesita de un preprocesador para funcionar más eficientemente, un preprocesador es un módulo que usa Snort para modificar paquetes que recibe del decodificador para posteriormente añadirlo al motor de reglas, o incluso evitarlo sin que salten las alarmas, analizan datos de la capa de aplicación de los paquetesque recibe y desensamblarlos y analizar capas inferiores. De todos los que existen en nuestro caso se ha elegido trabajar con el preprocesador ArpSpoof, para ello, es necesario comprender unos cuantos aspectos.

2.2. ¿Qué es ARP?

ARP es un protocolo de la capa de enlace [1], en esta capa del modelo OSI se establece el concepto de direcciones MAC o Ethernet, asociando una dirección IP con su dirección MAC. Este protocolo de resolución de direcciones necesita de una tabla de correspondencia entre direcciones MAC - direcciones IP, para que pueda redirigir el tráfico de un dispositivo origen a destino.

Este protocolo no está libre de ataques, y por ello se han implementado distintas medidas para mitigarlos. Un ataque muy conocido se denomina **ARP Spoofing** en la que como su nombre indica, un ordenador es capaz de ‘hacerse pasar por otro’ envenenando la tabla de resolución de direcciones en las máquinas de alrededor, haciendo creer al router que mi dirección MAC (atacante) se corresponde con la dirección IP que posee el usuario legítimo, y viceversa con el router, de esta manera es posible realizar un Man-in-the-middle.

2.3. Desarrollo

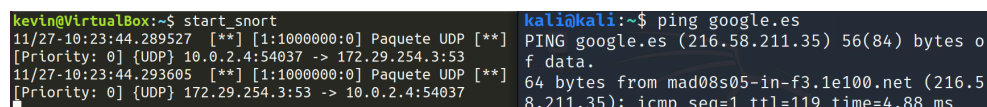
Para el desarrollo de esta práctica se va a exponer cómo se ha implementado el preprocesador junto a un intento de ataque de ARPSpoofing en donde se detecta la amenaza.

En primer lugar comentar que para no tener que tirar cada dos por tres el comando de inicio de snort con sus argumentos se ha creado un alias y añadido en el fichero `.bashrc`, concretamente esta línea:

```
alias start_snort="sudo snort -A console -q -i enp0s8 -c /etc/snort/snort.conf --daq-dir /usr/local/lib/daq/"
```

Se han implementado las reglas como demostración de la efectividad del IDS.

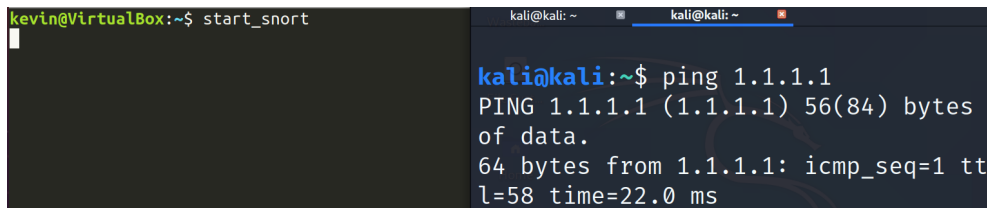
Es interesante conocer que al implementar una alerta sobre paquetes UDP nos avisa de un envío de un paquete UDP cuando realizamos un ping a un dominio, que se realiza sobre el protocolo de red ICMP, y no en la capa de transporte, como no entendía por qué snort detectaba esto, realicé un ping sobre una dirección IP en lugar de un dominio y Snort no capturaba ningún paquete UDP. Finalmente se descubrió que el paquete UDP que se captura cuando se realiza un ping hacia un dominio es la consulta DNS, que opera sobre UDP. En las imágenes 1 y 2 se puede observar cuando Snort detecta y cuando no detecta los paquetes que se envían y reciben.



```
kevin@VirtualBox:~$ start_snort
11/27-10:23:44.289527  [**] [1:1000000:0] Paquete UDP [**]
[Priority: 0] {UDP} 10.0.2.4:54037 -> 172.29.254.3:53
11/27-10:23:44.293605  [**] [1:1000000:0] Paquete UDP [**]
[Priority: 0] {UDP} 172.29.254.3:53 -> 10.0.2.4:54037

kali@kali:~$ ping google.es
PING google.es (216.58.211.35) 56(84) bytes of data:
64 bytes from mad08s05-in-f3.1e100.net (216.58.211.35): icmp_seq=1 ttl=119 time=4.88 ms
```

Figura 1: Ping a un dominio detectado



```
kevin@VirtualBox:~$ start_snort
kali@kali: ~
kali@kali: ~$ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes
of data.
64 bytes from 1.1.1.1: icmp_seq=1 tt
l=58 time=22.0 ms
```

Figura 2: Ping a una dirección IP no detectada

Como se puede observar se han implementado algunas reglas que demuestran la instalación y funcionamiento del IDS basado en red Snort.

2.4. Ejemplo práctico con el preprocesador

Para continuar con la práctica vamos a elegir un preprocesador y explicar como funciona, para la elección de este se ha elegido ARP spoof debido a la dificultad de controlar la capa de enlace, muchos preprocesadores se basan en análisis del contenido de paquetes, además muchos de estos ataques se pueden notificar o parar mediante firewalls o herramientas más complejas, sin embargo, existen menos herramientas que nos permiten realizar un análisis sobre este protocolo.

El preprocesador ARP spoof [2] captura paquetes ARP y detecta posibles amenazas y ataques hacia el protocolo ARP.

Para entender el preprocesador primero debemos atender a cómo funciona y a sus características, el preprocesador cuando no se añade ningún argumento solamente inspecciona direcciones Ethernet de los paquetes ARP. Además si se añade el argumento `-unicast` a la línea `preprocessor arpspoof` además se buscarán los paquetes unicast, la taxonomía, se debe añadir su configuración en el fichero `/etc/snort/snort.conf` siendo la siguiente:

```
preprocessor arpspoof[: <-unicast>]
preprocessor arpspoof_detect_host: <ip> <mac>
```

Existen 3 formas para configurar el preprocesador arpsspoof:

1. Normal

Esta configuración no realiza una búsqueda unicast ni monitoriza paquetes ARP, únicamente busca inconsistencias en las direcciones.

```
preprocessor arpspoof
```

2. Mapeo ARP

Una configuración un poco más avanzada en la que se sigue sin monitorizar paquete unicast pero sí monitoriza las direcciones IP con MAC para las máquinas que definimos.

```
preprocessor arpspoof
preprocessor arpspoof_detect_host: <ip> <mac>
```

3. Detección de unicast

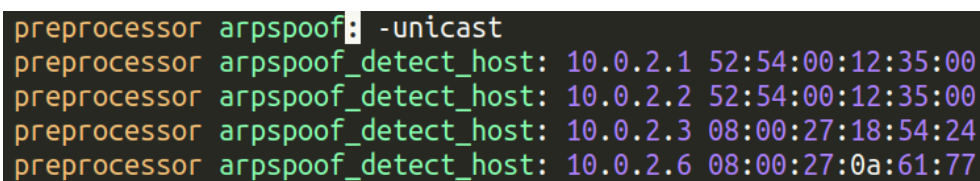
En este caso además de realizar un monitoreo de las máquinas de la red ofrece una monitorización sobre los paquetes unicast.

```
preprocessor arpspoof -unicast
preprocessor arpspoof_detect_host: <ip> <mac>
```

Para la inserción de reglas en el fichero `/etc/snort/snort.conf` tendrán el `gid: 112` [3], y además tienen cuatro `sid` [4]:

- 1. Unicast ARP request)
- 2. Etherframe ARP mismatch (src)
- 3. Etherframe ARP mismatch (dst)
- 4. ARP cache overwrite attack

El preprocesador se encuentra configurado tal y como se muestra en la imagen 3.

A screenshot of a terminal window showing the configuration of the 'arpspoof' preprocessor in Snort. The configuration includes the 'arpspoof' preprocessor itself and four instances of 'arpspoof_detect_host' with specific IP and MAC addresses.

```
preprocessor arpspoof -unicast
preprocessor arpspoof_detect_host: 10.0.2.1 52:54:00:12:35:00
preprocessor arpspoof_detect_host: 10.0.2.2 52:54:00:12:35:00
preprocessor arpspoof_detect_host: 10.0.2.3 08:00:27:18:54:24
preprocessor arpspoof_detect_host: 10.0.2.6 08:00:27:0a:61:77
```

Figura 3: Configuración preprocesador

Como se ha comentado anteriormente existe 4 `sid`, por lo que vamos a generar cuatro reglas de tipo alert como se muestra en la figura 4, activando la opción unicast y monitorizando las máquinas de la red que aparecen definidas.

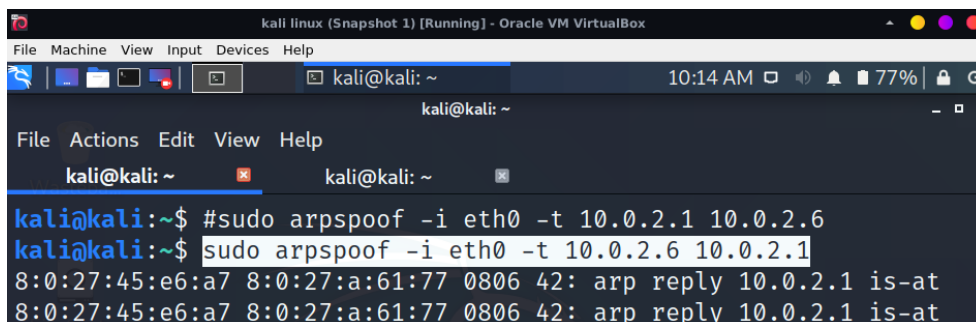
```

alert (msg:"Unicast ARP request"; sid:1; gid:112;)
alert (msg:"Etherframe ARP mismatch (src)"; sid:2; gid:112;)
alert (msg:"Etherframe ARP mismatch (dst)"; sid:3; gid:112;)
alert (msg:"ARP cache overwrite attack"; sid:4; gid:112;)

```

Figura 4: Reglas de ARP Spoof

A continuación se va a mostrar un ejemplo utilizando la herramienta arpspoof (como se aprendió en la asignatura optativa Seguridad de cuarto curso), se va a engañar al cliente como se muestra en la figura 5 al cliente indicando que la dirección 10.0.2.1 (router) corresponde con su MAC, y viceversa con el cliente, figura 6.

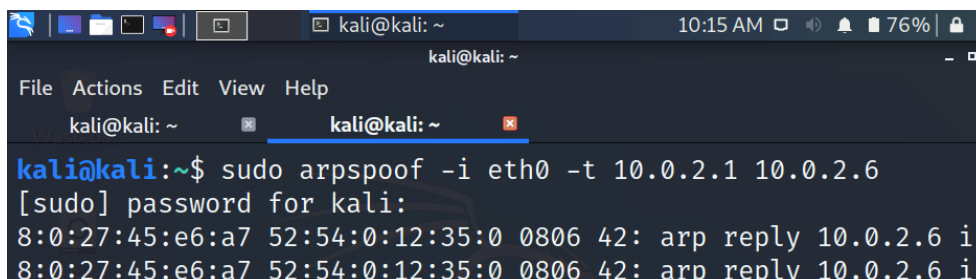


```

kali linux (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
kali@kali: ~ 10:14 AM 77%
File Actions Edit View Help
kali@kali: ~ kali@kali: ~
kali@kali:~$ #sudo arpspoof -i eth0 -t 10.0.2.1 10.0.2.6
kali@kali:~$ sudo arpspoof -i eth0 -t 10.0.2.6 10.0.2.1
8:0:27:45:e6:a7 8:0:27:a:61:77 0806 42: arp reply 10.0.2.1 is-at
8:0:27:45:e6:a7 8:0:27:a:61:77 0806 42: arp reply 10.0.2.1 is-at

```

Figura 5: ARP Spoofing engañando al cliente



```

kali@kali: ~ 10:15 AM 76%
File Actions Edit View Help
kali@kali: ~ kali@kali: ~
kali@kali:~$ sudo arpspoof -i eth0 -t 10.0.2.1 10.0.2.6
[sudo] password for kali:
8:0:27:45:e6:a7 52:54:0:12:35:0 0806 42: arp reply 10.0.2.6 i
8:0:27:45:e6:a7 52:54:0:12:35:0 0806 42: arp reply 10.0.2.6 i

```

Figura 6: ARP Spoofing engañando al router

Seguidamente se va a proceder a iniciar el IDS Snort y como podemos observar en la figura 7, nos aparecen las siguientes alertas:

- ARP cache overflow attack

Debido a un continuo envío de paquetes ARP inusual.

- Etherframe ARP mismatch (src)
Debido a que la trama ARP no coincide con el que se encuentra en el fichero de configuración.
- Unicast ARP request
Debido al envío de paquetes unicast cuando deben de enviarse de manera multicast.

```

kevin@VirtualBox:~$ start_snort
11/27-11:55:18.468127  [**] [112:4:0] ARP cache overwrite attack [**]
11/27-11:56:18.567933  [**] [112:4:0] ARP cache overwrite attack [**]
11/27-11:56:23.326421  [**] [112:2:0] Etherframe ARP mismatch (src) [**]

11/27-11:56:54.940928  [**] [1:1000000:0] Paquete UDP [**] [Priority: 0]
{UDP} 10.0.2.6:40516 -> 172.29.254.3:53
11/27-11:56:54.944208  [**] [1:1000000:0] Paquete UDP [**] [Priority: 0]
{UDP} 172.29.254.3:53 -> 10.0.2.6:40516
11/27-11:56:59.966597  [**] [112:1:0] Unicast ARP request [**]
^C*** Caught Int-Signal
11/27-11:57:24.050549  [**] [1:1000000:0] Paquete UDP [**] [Priority: 0]
{UDP} 10.0.2.4:68 -> 10.0.2.3:67

```

Figura 7: Detección del IDS Snort

Podemos concluir que el preprocesador funciona correctamente contra este tipo de ataques debido a que inspecciona y localiza de manera exitosa las posibles amenazas que definamos en el fichero de configuración `/etc/snort/snort.conf`.

3 Parte 2 - Suricata

El alumno debe elegir un IDS diferente a Snort:

- Una memoria con explicación detallada:
- Instalación del IDS elegido
- Configuración de los elementos principales
- Generación de reglas
- ¿Qué diferencias encuentras entre Snort?

3.1. Introducción

Una vez se ha comentado como funciona Snort, se ha elegido como IDS adicional para completar esta práctica el IDS Suricata [5], este sistema además se puede configurar como IPS (Intrusion Prevention System), NSM (Network Security Monitoring) y un procesador pcap sin necesidad de estar conectado a Internet.

La instalación se ha realizado siguiendo los pasos de la página Atlantic [6] y se explicarán a continuación

3.2. Configuración

Tras finalizar con la instalación del IDS de manera compilada podemos mostrar las reglas que ofrece Suricata en la imagen 8.

```
kevin@VirtualBox:~/suricata-5.0.3$ sudo su -
root@VirtualBox:~# cat /var/lib/suricata/rules/suricata.rules | head -5
alert ip any any -> any any (msg:"SURICATA Applayer Mismatch protocol both
directions"; flow:established; app-layer-event:applayer_mismatch_protocol
_both_directions; flowint:applayer.anomaly.count,+,1; classtype:protocol-c
ommand-decode; sid:2260000; rev:1;)
alert ip any any -> any any (msg:"SURICATA Applayer Wrong direction first
Data"; flow:established; app-layer-event:applayer_wrong_direction_first_da
ta; flowint:applayer.anomaly.count,+,1; classtype:protocol-command-decode;
sid:2260001; rev:1;)
```

Figura 8: Reglas instaladas en el Suricata

Continuando con la configuración se ha creado como en el punto 1 una red NAT desde virtualbox cuyo rango de IPs lo hemos establecido como variable global de HOME_NET como se muestra en la configuración.

```
vars:
# more specific is better for a
address-groups:
#HOME_NET: "[192.168.0.0/16,10.0.0.0/8]"
HOME_NET: "[10.0.2.0/24]"
#HOME_NET: "[192.168.0.0/16]"
#HOME_NET: "[10.0.0.0/8]"
#HOME_NET: "[192.168.0.0/16]"
```

Figura 9: Configuración variable HOME NET

Para finalizar he capturado el nombre de mi tarjeta de red para crear un alias y facilitar la ejecución de la herramienta añadiendo el comando en el fichero `.bashrc`, el contenido que he añadido es el siguiente:

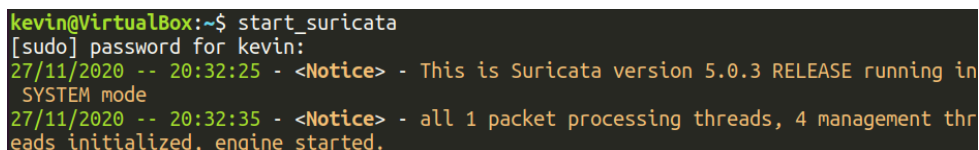

```
alias start_suricata="sudo suricata -c /etc/suricata/suricata.yaml  
-i enp0s8"
```

3.3. Configuración de reglas

La configuración de las reglas se encuentran en el fichero `/etc/suricata/suricata.yaml`, en él podemos añadir distintos ficheros creados o descargados para que los incluya a la hora de inspeccionar paquetes.

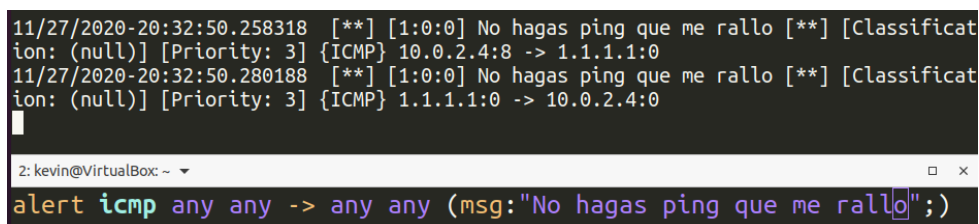
La creación de las reglas de Suricata es igual que las de Snort debido a que su taxonomía es la misma, por lo que podremos intercambiar distintas reglas entre ellas, como ejemplo se va a mostrar cómo crear reglas que filtren por contenido.

Como ejemplo base se ha creado una alerta para detectar protocolos ICMP (ping) como observamos en la imagen 11, lanzamos suricata mediante el comando `$ start_suricata` (figura 10) y observamos en la terminal de arriba de la imagen 11 que procesa los pings cuando se lo enviamos desde la otra máquina virtual.



```
kevin@VirtualBox:~$ start_suricata  
[sudo] password for kevin:  
27/11/2020 -- 20:32:25 - <Notice> - This is Suricata version 5.0.3 RELEASE running in  
SYSTEM mode  
27/11/2020 -- 20:32:35 - <Notice> - all 1 packet processing threads, 4 management thr  
eads initialized, engine started.
```

Figura 10: Configuración variable HOME NET



```
11/27/2020-20:32:50.258318  [**] [1:0:0] No hasas ping que me rallo [**] [Classificat  
ion: (null)] [Priority: 3] {ICMP} 10.0.2.4:8 -> 1.1.1.1:0  
11/27/2020-20:32:50.280188  [**] [1:0:0] No hasas ping que me rallo [**] [Classificat  
ion: (null)] [Priority: 3] {ICMP} 1.1.1.1:0 -> 10.0.2.4:0  
[  
2: kevin@VirtualBox: ~  
alert icmp any any -> any any (msg:"No hasas ping que me rallo";)
```

Figura 11: Configuración variable HOME NET

Además se ha añadido un fichero de reglas ¹ para evitar salir a tor, descargando el fichero y añadiendolo a la localización `/var/lib/suricata/rules`

¹<https://github.com/jpalanco/alienvault-ossim/blob/master/snort-rules-default-open/rules/2.9.2/emerging.rules/emerging-tor.rules>

```
default-rule-path: /var/lib/suricata/rules

rule-files:
- suricata.rules
- custom.rules
- emerging-tor.rules
```

Figura 12: Rutas de ficheros de reglas donde se va a buscar

Como se muestra en la figura 13, a la izquierda se obtiene una parte de la máquina virtual Kali Linux donde se ve que estamos abriendo Tor, a la derecha con el Ubuntu abierto y mediante el fichero de log leyendo (/var/log/suricata/fast.log) se observa que se detectan las IPs que en este caso con Onion Proxys y Guard Nodes a los que nos conectamos.

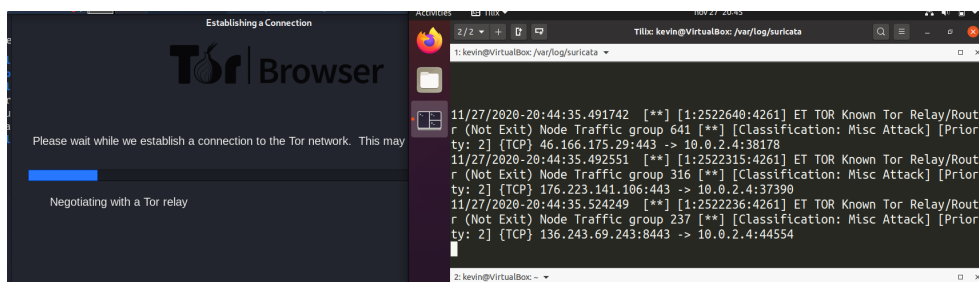


Figura 13: Detección por parte de Suricata cuando abro Tor

4 Diferencias entre IDS y conclusiones

En este apartado se va a proceder a comentar las características y diferencias más reseñables que se han encontrado en esta práctica con la utilización de los dos IDS.

En primer lugar cabe destacar que Snort es monohilo, lo que provoca es que su rendimiento se vea limitado a menos que se cree un cluster, sin embargo Suricata es multihilo, una de las ventajas que presenta este tipo de IDS es la ausencia de pérdida de paquetes, al poder balancear la carga entre los procesadores soporta realizar más acciones sin encolar ni descartar paquetes, característica que no tiene Snort.

Ambos NIDS poseen una gran comunidad de usuarios, es por ello que se van actualizando más a menudo el software, reglas, etc. Suricata además ofrece dos características importantes: un panel de administración y un módulo de detección de anomalías.

Una de las características reseñables que posee Snort es que cuando inspecciona un paquete que coincide con algún filtro o regla que aparece en su archivo de configuración se notificará añadiendo información importante, como el cuándo se ha inspeccionado el paquete, el dónde y el cómo. Además nos ofrece la posibilidad de elegir donde guardar la información registrada.

Ambos se basan en firmas de conocimiento, sin embargo, Snort permite el uso de preprocesadores mientras que Suricata no posee esa característica, sin embargo Suricata permite utilizar las mismas reglas que Snort y permite diferenciar protocolos a nivel de aplicación.

Con todo ello sumado a que Suricata permite la geolocalización por IP, mientras que Snort no, y la posibilidad de extraer archivos interceptados bajo mi punto de vista Suricata es un software más potente de cara a la implementación en un ambiente laboral y personal.

Bibliografía

- [1] D. Plummer. An ethernet address resolution protocol: Or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware. [Online]. Available: <https://tools.ietf.org/html/rfc826>
- [2] Decoder and preprocessor rules. [Online]. Available: <http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node18.html>
- [3] 2.3 decoder and preprocessor rules. [Online]. Available: <http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node18.html>
- [4] eldondev/snort. [Online]. Available: <https://github.com/eldondev/Snort>
- [5] Suricata. [Online]. Available: <https://suricata-ids.org/>
- [6] How to install and setup suricata IDS on ubuntu 20.04. [Online]. Available: <https://www.atlantic.net/vps-hosting/how-to-install-and-setup-suricata-ids-on-ubuntu-20-04/>