Escalamiento de seguridad en redes SDN para generación de reglas en NIDS empleando J48 y TENSORFLOW para analizar ataques basados en tiempo

Toshiro Nagata Bolivar
Universidad Católica de Santa María, Perú, tnagata@ucsm.edu.pe

Resumen– En la actualidad los problemas de seguridad en redes de datos tradicionales van aumentando de forma exponencial, existe un concepto emergente que pretende reemplazar las redes tradicionales separando la lógica de control de la red, de los switches y enrutadores subyacentes, sugiriendo la centralización lógica del control de red, y permitiendo programar la misma. SDN ofrece una solución atractiva para la seguridad de la red. Sin embargo, la predicción de ataques y prevención de los mismos aún se encuentra en constante investigación. El presente artículo propone una arquitectura para mejorar la seguridad en redes SDN mediante la generación automática de reglas para Suricata-IDS (sistema de detección de intrusos), utilizando el algoritmo J48 para el análisis del conjunto de datos de Kyoto2009+, tomando como atributo principal el tiempo de conexión para el posterior entrenamiento de tensorflow para poder realizar auditorías a conexiones sospechosas. Para dicho análisis se utilizará la metodología de seguridad informática según Benson En los resultados se observa la autogeneración de reglas para Suricata-IDS según los patrones generados, así como seguimiento de conexiones sospechosas detectadas por tensorflow. Como conclusión se puede generar reglas de seguridad automáticamente diferenciando el comportamiento de tráfico malicioso para evaluar futuras conexiones, así como realizar seguimiento de las mismas para evitar conexiones a sitios de alto riesgo.

Palabras Clave—Seguridad, Software Defined Network(SDN), Network Iintrusion Detection System(NIDS), J48, TENSORFLOW.

I. INTRODUCCIÓN

A la actualidad los delitos informáticos que se dan dentro de una organización se han visto incrementados de forma exponencial en su faceta de sustracción de información confidencial y de secretos industriales. En vista de la importancia que han tomado las redes de datos en las distintas organizaciones, todas buscan tener la mayor seguridad en los sistemas de comunicaciones, para de esta manera poder evitar poner en riesgo los pilares de seguridad informática tales como son la confidencialidad, disponibilidad e integridad (CIA), los cuales podrían desencadenar riesgos inherentes teniendo como consecuencia posibles pérdidas económicas, de imagen, etc. Debido a que las redes tradicionales IPV4 fueron diseñadas hace muchos años estas no fueron pensadas en la seguridad, sino que tienen un mayor énfasis en la eficiencia, en la actualidad con la implementación de protocolos como IPV6 se trató de solucionar muchos de los problemas de seguridad, pero actuales investigaciones como[1], concluyen que los problemas de seguridad no fueron totalmente solucionados.

The Software-Defined Network (SDN) se ha convertido en una prometedora arquitectura de red en la que los dispositivos son controlados por un controlador SDN. Pero a la actualidad la prevención de ataques, especialmente la denegación de servicio distribuida (DDoS) abarca un gran problema de seguridad[2]. Actualmente muchas organizaciones cuentan con medidas de seguridad como NIDS (sistema de detección de intrusos en red), pero distintas investigaciones como * concluyen que no son una medida de seguridad del todo confiable ya que como se explica en[3] existen distintas formas de poder evadir esta medida de seguridad.

Al utilizar el algoritmo J48 con la colección de datos de Kyoto 2009+, se obtuvo un patrón sobre las características de distintos tipos de ataques los cuales ya fueron clasificados, siendo objetivo de este estudio centrarnos en la característica de tiempo para el análisis del conjunto de datos, y en trabajo conjunto con tensorflow la cual es una librería de Python liberada por Google preparada para redes neuronales, inteligencia artificial y Deep Learning (aprendizaje profundo), se entrenara a este para que pueda tomar decisiones sobre las futuras conexiones pudiendo clasificarlas como normal o sospechosa, en caso de evadir el sistema de seguridad de NIDS, y cumplir con las características de patrones generados como actividad sospechosa se elevara el nivel de seguridad de los nids utilizando más recursos de red pero obteniendo una mayor atención sobre esta conexión, al igual que se desplegara un hilo de control de auditoria sobre esta conexión en específico para monitorear su actividad, el controlador SDN utilizado para esta investigación es Floodlight, en el que se programó un módulo para poder guardar en una base de datos PostgreSQL 9.6 las direcciones Ipv4/Ipv6, direcciones MAC y payload que se transmiten atreves del controlador tanto de los nodos como de los switches, los cuales son emulados con el software mininet, del mismo modo en esta base de datos se guardaran distintos conjuntos de datos de páginas maliciosas para generar automáticamente reglas en los NIDS evitando que las conexiones puedan acceder a dichas paginas comprometiendo así la seguridad de toda la infraestructura. Es objetivo de la presente investigación que sirva como medio de consulta para servir de apoyo en la implementación de nuevos sistemas de seguridad.

A. Objetivos

- Análisis de conjunto de datos de seguridad para hallar patrones de comportamiento malicioso en base a tiempo y generar reglas automáticamente para los NIDS en nuestra organización.
- Predecir y prevenir nuevas conexiones maliciosas.
- Prevenir el ingreso a sitos o direcciones de alto riesgo.
- Establecer un control de seguridad en las redes.
- Seguimiento de conexiones sospechosas.
- Guardar historial de conexiones y direcciones(IPV4/IPV6/MAC) para auditorias futuras.

B. Estado del arte

Como se menciona en [2] los ataques DDS aún son un desafío para la seguridad en redes SDN, en este trabajo se realiza un análisis estadístico de las características reales de trafico de red tanto en estados normales como en estado de ataques, con base en dichos análisis se puede definir criterios para detectar con alta probabilidad el riesgo de sufrir un ataque así como en el futuro el análisis de cada paquete para una mejor verificación de ataques en tiempo real. En [4] se destaca la importancia de manejar tablas de flujo con la conciencia de su limitación de tamaño, así como las soluciones que pueden abordar los ataques en los recursos y desafíos de los mismos. De momento las técnicas basadas en el aprendizaje de máquina para el manejo de ataques e intrusiones ha recibido mucha atención en la computación siendo la inteligencia que maneja las redes uno de los principales pilares, como se explica en [5] el cual busca analizar distintas técnicas de aprendizaje máquina que pueden ser utilizados para solucionar varios de los problemas de seguridad en redes SDN. Siendo SDN una arquitectura emergente la cual es dinámica manejable y adaptable lo que la convierte en una arquitectura ideal para manejar grandes cantidades de ancho de banda, en[6] se muestra como los algoritmos de aprendizaje automático tienen un gran campo de aplicación en redes SDN para poder clasificar y mitigar distintos problemas de seguridad. En[7] se propone un mecanismo de defensa ante ataques DDos en redes SDN analizando dichos ataques y encontrando ventajas en SDN para construir un mecanismo de defensa demostrando con un ejemplo dicha investigación para la verificación de su propuesta ante ataques DDoS. En la investigación[8] se proporciona un esquema de mitigación de atraques en múltiples etapas para SDNH, extendiendo una arquitectura normal de SDN a una más completa que aprovecha todas las ventajas de SDN para mitigar ataques y su despliegue instantáneo, donde las funciones de seguridad se despliegan ampliamente en la red proporcionando características de vista global y control general para la arquitectura de seguridad, por esta razón la arquitectura proporciona una política ágil y funciones de respuestas eficaces, se propone un mecanismo y algoritmos de evaluación de seguridad impulsado por evidencias para resolver problemas de seguridad de forma dinámica. En[9] se presenta un enfoque para hacer frente a uno de los ataques más conocidos de envenenamiento del protocolo ARP, se propone un nuevo algoritmo para prevenir este ataque utilizando un controlador RYU para SDN demuestran que dicho algoritmo evita el envenenamiento ARP y muchos otros tipos de ataques que utilizan el mismo protocolo. Como se menciona en[10] se propone un enrutamiento dinámico basado en la fragmentación para proporcionar una mayor seguridad, al ignorar a los nodos menos fiables de la red para encaminar los fragmentos de datos. En[11] se propone un esquema de protección optimizado para redes SDN proponiendo un algoritmo de ajusta con inteligencia artificial que supera los métodos tradicionales para ahorrar recursos en SDN, en trabajos futuros propone ampliar su algoritmo para soportar una mayor cantidad de ataques. Como se describe en[12] las redes SDN en el cloud computing también pueden tener brechas de seguridad en 5g existe un tema amplio que aún debe ser estudiado a fondo. En[13] se presenta ABB que aprovecha la arquitectura centralizada de SDN para proporcionar una mitigación a los ataques DDoS, ya que tiene un coste por paquete actualmente es muy costoso pero se puede reducir el costo juntando estadísticas de ataques DDoS para mejorar la respuesta. En la investigación [14] se desarrolla un mecanismo de prevención para evitar ataques DoS en sus primeras etapas. En la investigación[15] se demuestra que pocos cambios son requeridos en la arquitectura actual del cloud computing, el mecanismo de monitoreo y control de red basados en SDN permite controlar y configurar los mecanismos de seguridad en la nube. En [16] se presenta un prototipo de un mecanismo capaz de restringir distintos tipos de ataques en SDN bajo un estricto control de acceso. En el artículo [17] se discute las razones por la que los ataques son cada vez más frecuentes y las alternativas que las redes SDN posee para hacerles frente, y al mismo tiempo las investigaciones que demuestran que no es una arquitectura del todo segura, se discute también sobre el futuro del cloud computing y como influirán las redes SDN. En[18] se trata de adaptar los IDS a varios dominios basados en SDN, se explica como la función de los IDS pero no se logra implementar con todas las pruebas ya que es un trabajo arduo que se deja planteado en el apartado de trabajos futuros. Como se demuestra en[19] se desarrollan técnicas para asegurar dominios SDN, se describe el diseño de una arquitectura para SDN que consiste en aplicaciones de seguridad que se ejecutan en la parte superior del controlador para especificar políticas de seguridad. Se observa en[20] que se utiliza las redes SDN para hacer frente a los ataques de ingeniera social en este caso phishing, implementando un algoritmo basado en redes neuronales en un controlador Ryu.

II. MATERIALES Y MÉTODOS

La metodología de seguridad informática según Benson específicamente fue diseñada para apoyar a quienes

Digital Object Identifier: (to be inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).

trabajan con el desarrollo de la seguridad, las estrategias y planes para la protección de la disponibilidad, integridad y confidencialidad de los datos de los sistemas informáticos.

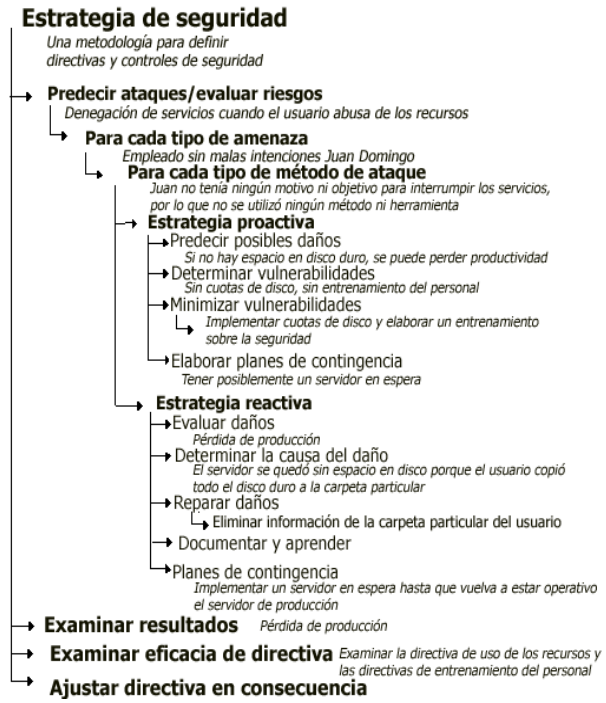


Fig 1. Metodología de seguridad informática según Benson [21].

Existen cuatro pasos a seguir dentro de esta metodología.

- Identificar métodos, herramientas y técnicas de ataques probables: Métodos, herramientas y técnicas de ataques que pueden abarcar desde algo como los diversos virus existentes hasta las nuevas metodologías de implantación codificada de sistemas que alteran e infringen contra la integridad y estabilidad de los datos.

- Establecer estrategias proactivas y reactivas: Nos encamina a reducir al mínimo las directivas de seguridad, así como de desarrollar planes de contingencia.

- Pruebas: Se debe llevar a cabo luego de que se haya puesto en marcha las estrategias proactivas y reactivas, con el fin de mejorar las directivas y controles de seguridad a implementar posteriormente.

- Formar equipos de respuestas a incidentes: Se identifican herramientas de software para responder a incidentes, realización de actividades formativas, junto con la ejecución de estudios a ataques al sistema.

Como se observa en la Fig.2. en el presente trabajo de investigación se propone mejorar la seguridad en las redes SDN, para lograr este objetivo al separar el plano de datos y el plano de control, integrando a este un IDS, así como también algoritmos de J48 y Tensorflow, de esta manera logramos mejorar la eficiencia de Suricata-IDS, autogenerando reglas basadas en distintos tipos de actividad sospechosa, por lo tanto, se mejora la eficiencia en la seguridad de redes SDN.

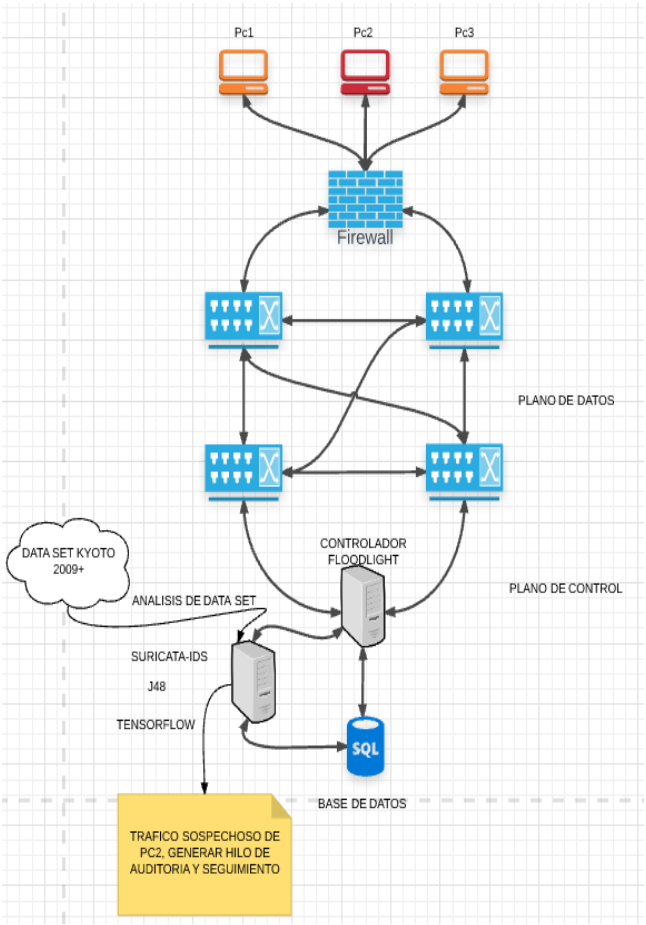


Fig 2. Arquitectura para mejorar la seguridad de redes SDN

III. Resultados y Discusiones

En el momento de desarrollar el componente personalizado del controlador Floodlight para redes SDN, se aprecia en la Fig. 3. que imprimimos todas sus características de conexión, para posteriormente almacenarlas en una base de datos, esto es de vital importancia ya que brinda una gran información sobre lo que se llevara a cabo en la capa de control.

```
@Override
public void init(FloodlightModuleContext context) throws FloodlightModuleExcep
    floodlightProvider = context.getServiceImpl(IFloodlightProviderService.cla
    macAddresses = new ConcurrentSkipListSet<Long>();
    logger = LoggerFactory.getLogger(NagataControllerLaccei.class);
}

    @Override
    public net.floodlightcontroller.core.IListener.Command receive(IOFSwitch sw
        Ethernet eth =
                IFloodlightProviderService.bcStore.get(cntx,
                                                       IFloodlightProviderService.CONTEXT
        Long sourceMACHash = eth.getSourceMACAddress().getLong();
        if (!macAddresses.contains(sourceMACHash)) {
            macAddresses.add(sourceMACHash);

            logger.info("Dirección MAC Origen: {} en switch: {}",
                    eth.getSourceMACAddress().toString(),
                    sw.getId().toString());

            logger.info("Dirección MAC Destino: {} en switch: {}",
                    eth.getDestinationMACAddress().toString(),
                    sw.getId().toString()
                    );
            logger.info("Ether Type: {}",
                    eth.getEtherType()
                    );
            logger.info("Payload: {}",
                    eth.getPayload()
                    );
            logger.info("Parent: {}",
                    eth.getParent()
                    );
            logger.info("VlanID: {}",
                    eth.getVlanID()
                    );
            logger.info("PriorityCode: {}",
                    eth.getPriorityCode()
                    );
            logger.info("Ports: {}",
                    sw.getPorts()
                    );
            logger.info("Status: {}",
                    sw.getPorts()
                    );
        }
        return Command.CONTINUE;
    }
```

Fig 3. Controlador Floodlight para guardar características de conexiones

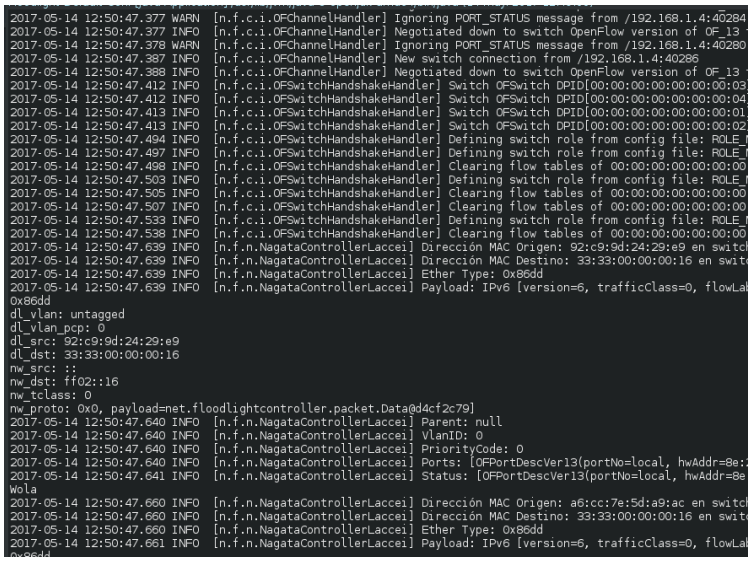
En la Fig.4. se observa la simulación de una red con el software mininet, el cual se conectará directamente a nuestro controlador Floodlight, siento este capaz de imprimir las características de conexión.

```
root@KPC:/home/rootf# mn --topo=linear,4  --
controller=remote,ip=192.168.1.4,port=6653  --switch ovsk

*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Fig 4. Emulación de una topología de red con Mininet

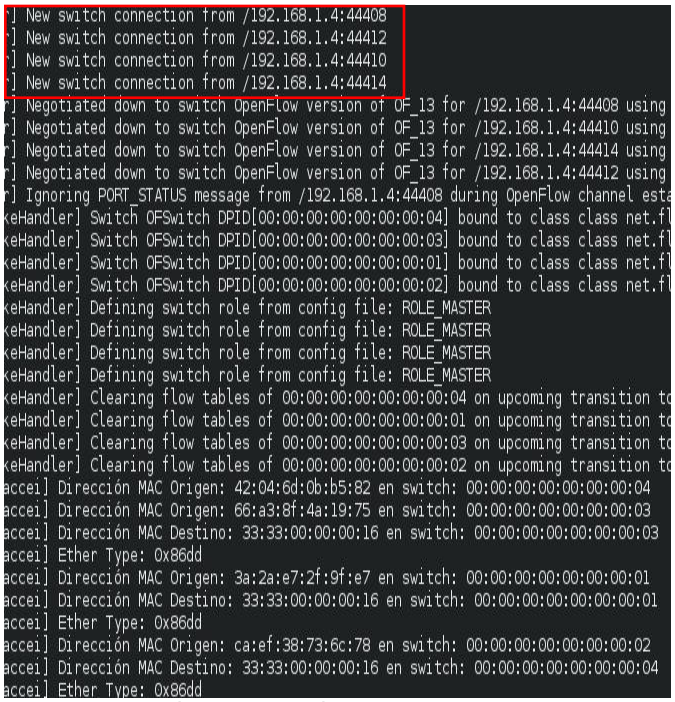
Como se aprecia en la Fig.5. nuestro controlador de Floodlight imprime las características de conexión de los distintos hosts, esto será almacenado en una base de datos para posteriormente mantener datos históricos de conexión ante cualquier eventualidad.



```
2017-05-14 12:50:47.377 WARN  [n.f.c.i.OFChannelHandler] Ignoring PORT_STATUS message from /192.168.1.4:40284
2017-05-14 12:50:47.377 INFO  [n.f.c.i.OFChannelHandler] Negotiated down to switch OpenFlow version of OF_13
2017-05-14 12:50:47.378 WARN  [n.f.c.i.OFChannelHandler] Ignoring PORT_STATUS message from /192.168.1.4:40280
2017-05-14 12:50:47.387 INFO  [n.f.c.i.OFChannelHandler] New switch connection from /192.168.1.4:40286
2017-05-14 12:50:47.388 INFO  [n.f.c.i.OFChannelHandler] Negotiated down to switch OpenFlow version of OF_13
2017-05-14 12:50:47.412 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Switch OFSwitch DPID[00:00:00:00:00:00:00:03]
2017-05-14 12:50:47.412 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Switch OFSwitch DPID[00:00:00:00:00:00:00:04]
2017-05-14 12:50:47.413 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Switch OFSwitch DPID[00:00:00:00:00:00:00:01]
2017-05-14 12:50:47.413 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Switch OFSwitch DPID[00:00:00:00:00:00:00:02]
2017-05-14 12:50:47.494 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Defining switch role from config file: ROLE_
2017-05-14 12:50:47.497 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Defining switch role from config file: ROLE_
2017-05-14 12:50:47.498 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Clearing flow tables of 00:00:00:00:00:00:00:0
2017-05-14 12:50:47.503 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Defining switch role from config file: ROLE_
2017-05-14 12:50:47.505 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Clearing flow tables of 00:00:00:00:00:00:00:0
2017-05-14 12:50:47.507 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Clearing flow tables of 00:00:00:00:00:00:00:0
2017-05-14 12:50:47.533 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Defining switch role from config file: ROLE_
2017-05-14 12:50:47.538 INFO  [n.f.c.i.OFSwitchHandshakeHandler] Clearing flow tables of 00:00:00:00:00:00:00:0
2017-05-14 12:50:47.639 INFO  [n.f.n.NagataControllerLaccei] Dirección MAC Origen: 92:c9:9d:24:29:e9 en switch
2017-05-14 12:50:47.639 INFO  [n.f.n.NagataControllerLaccei] Dirección MAC Destino: 33:33:00:00:00:16 en swit
2017-05-14 12:50:47.639 INFO  [n.f.n.NagataControllerLaccei] Ether Type: 0x86dd
2017-05-14 12:50:47.639 INFO  [n.f.n.NagataControllerLaccei] Payload: IPv6 [version=6, trafficClass=0, flowLa
0x86dd
dl_vlan: untagged
dl_vlan_pcp: 0
dl_src: 92:c9:9d:24:29:e9
dl_dst: 33:33:00:00:00:16
nw_src: ::
nw_dst: ff02::16
nw_tclass: 0
nw_proto: 0x0, payload=net.floodlightcontroller.packet.Data@d4cf2c79]
2017-05-14 12:50:47.640 INFO  [n.f.n.NagataControllerLaccei] Parent: null
2017-05-14 12:50:47.640 INFO  [n.f.n.NagataControllerLaccei] VlanID: 0
2017-05-14 12:50:47.640 INFO  [n.f.n.NagataControllerLaccei] PriorityCode: 0
2017-05-14 12:50:47.640 INFO  [n.f.n.NagataControllerLaccei] Ports: [OFPortDescVer13(portNo=local, hwAddr=8e:
2017-05-14 12:50:47.641 INFO  [n.f.n.NagataControllerLaccei] Status: [OFPortDescVer13(portNo=local, hwAddr=8e
Wola
2017-05-14 12:50:47.660 INFO  [n.f.n.NagataControllerLaccei] Dirección MAC Origen: a6:cc:7e:5d:a9:ac en switch
2017-05-14 12:50:47.660 INFO  [n.f.n.NagataControllerLaccei] Dirección MAC Destino: 33:33:00:00:00:16 en swit
2017-05-14 12:50:47.660 INFO  [n.f.n.NagataControllerLaccei] Ether Type: 0x86dd
2017-05-14 12:50:47.661 INFO  [n.f.n.NagataControllerLaccei] Payload: IPv6 [version=6, trafficClass=0, flowLa
```

Fig 5. Características de conexión detectadas por Floodlight

Se aprecia del mismo modo en la Fig.6. las características y conexiones de los distintos switches.



```
] New switch connection from /192.168.1.4:44408
] New switch connection from /192.168.1.4:44412
] New switch connection from /192.168.1.4:44410
] New switch connection from /192.168.1.4:44414
] Negotiated down to switch OpenFlow version of OF_13 for /192.168.1.4:44408 using
] Negotiated down to switch OpenFlow version of OF_13 for /192.168.1.4:44410 using
] Negotiated down to switch OpenFlow version of OF_13 for /192.168.1.4:44414 using
] Negotiated down to switch OpenFlow version of OF_13 for /192.168.1.4:44412 using
] Ignoring PORT_STATUS message from /192.168.1.4:44408 during OpenFlow channel esta
eHandler] Switch OFSwitch DPID[00:00:00:00:00:00:00:04] bound to class class net.fl
eHandler] Switch OFSwitch DPID[00:00:00:00:00:00:00:03] bound to class class net.fl
eHandler] Switch OFSwitch DPID[00:00:00:00:00:00:00:01] bound to class class net.fl
eHandler] Switch OFSwitch DPID[00:00:00:00:00:00:00:02] bound to class class net.fl
eHandler] Defining switch role from config file: ROLE_MASTER
eHandler] Defining switch role from config file: ROLE_MASTER
eHandler] Defining switch role from config file: ROLE_MASTER
eHandler] Defining switch role from config file: ROLE_MASTER
eHandler] Clearing flow tables of 00:00:00:00:00:00:00:04 on upcoming transition to
eHandler] Clearing flow tables of 00:00:00:00:00:00:00:01 on upcoming transition to
eHandler] Clearing flow tables of 00:00:00:00:00:00:00:03 on upcoming transition to
eHandler] Clearing flow tables of 00:00:00:00:00:00:00:02 on upcoming transition to
accei] Dirección MAC Origen: 42:04:6d:0b:b5:82 en switch: 00:00:00:00:00:00:00:04
accei] Dirección MAC Origen: 66:a3:8f:4a:19:75 en switch: 00:00:00:00:00:00:00:03
accei] Dirección MAC Destino: 33:33:00:00:00:16 en switch: 00:00:00:00:00:00:00:03
accei] Ether Type: 0x86dd
accei] Dirección MAC Origen: 3a:2a:e7:2f:9f:e7 en switch: 00:00:00:00:00:00:00:01
accei] Dirección MAC Destino: 33:33:00:00:00:16 en switch: 00:00:00:00:00:00:00:01
accei] Ether Type: 0x86dd
accei] Dirección MAC Origen: ca:ef:38:73:6c:78 en switch: 00:00:00:00:00:00:00:02
accei] Dirección MAC Destino: 33:33:00:00:00:16 en switch: 00:00:00:00:00:00:00:04
accei] Ether Type: 0x86dd
```

Fig 6. características de conexión de nuevos switches y nodos

Al iniciar Suricata-IDS se realiza el análisis de las conexiones y verificación de ataques más conocidos los cuales se encuentran almacenados en las reglas por defecto de Suricata-IDS, en la Fig.7. se observa la detección de ataques clasificados como “Network Tojan”, indicando las direcciones a las que se realizan las conexiones inversas, Suricata-IDS clasifica el tráfico malicioso guardando estas incidencias en su

propio log para posterior análisis, así como como la eliminación de estas conexiones para evitar incidentes de seguridad.

```
[**][Classification A Network Trojan was detected]
[Priority:1] {TCP}10.0.159.23:78562 -> 10.0…

[**][Classification A Network Trojan was detected]
[Priority:1] {TCP}10.0.159.25:78562 -> 10.0…

[**][Classification A Network Trojan was detected]
[Priority:1] {TCP}10.0.159.28:78562 -> 10.0…

[**][Classification A Network Trojan was detected]
[Priority:1] {TCP}10.0.159.28:78562 -> 10.0…
```

Fig 7. Detección de ataques por Suricata-IDS

Tras la verificación de la funcionalidad de Suricata-IDS, como se observa en la Fig.8. se toma un conjunto de datos de un honeypot de la universidad de kyoto (Kyoto2009+) los cuales han sido analizados bajo una arquitectura de seguridad y puestos a disposición para su análisis, un script vuelca los datos de este dataset en la base de datos(PostgreSQL), para el análisis de dichos datos extraeremos tanto la duración así como las banderas(flags), y la clasificación que se dio para su posterior análisis con el algoritmo J48 y Tensorflow.

```
10  select kyoto."Duracion",kyoto."Flag",kyo
```

Output pane

Data Output | Explain | Messages | History

	Duracion numeric	Flag text	Clasificacion text
1	0.12	SF	attack
2	33.9	SF	normal
3	0	S0	attack
4	0	S0	attack
5	0	S0	attack
6	0	S0	attack
7	2.77	SF	normal
8	0	RSTOS0	normal
9	35.58	RSTO	normal
10	74.96	RSTR	normal
11	74.98	RSTR	normal
12	3.19	SF	normal
13	0	S0	attack
14	0	S0	attack
15	6.65	SF	normal
16	35.28	SF	normal
17	0	S0	attack
18	1.58	SF	attack
19	1.39	SF	normal
20	74.76	RSTR	normal
21	74.78	RSTR	normal
22	8.82	RSTO	unknown_attack
23	53.9	SF	normal
24	74.57	RSTR	normal
25	3.93	SF	normal
26	2.87	SF	normal
27	0	S0	attack
28	0	S0	attack
29	2.6	RSTO	normal

Fig 8. Selección de los campos del conjunto de datos Kyoto 2009+ después de la inserción en la base de datos

Con la función "avg" en PostgreSQL se obtiene el tiempo promedio de demora del tráfico que fue clasificado como normal, así como el que fue clasificado como ataque, ambos considerados en milisegundos, en la Fig.9. se observa el primer valor (2.37) correspondiente al tiempo de demora de las conexiones clasificadas como ataques, así como el tiempo promedio (19.788) de las conexiones consideradas como tráfico normal.

```
1  (select avg(kyoto."Duracion")from kyoto where
2     kyoto."Clasificacion"='attack'
3     UNION ALL
4  select avg (kyoto."Duracion")from kyoto where
5     kyoto."Clasificacion"='normal')
```

Output pane

Data Output | Explain | Messages | History

	avg numeric
1	2.3728658651727828
2	19.7885953431121141

Fig 9. Tiempo promedio de duración de conexiones

Para establecer un patrón de características primero se aplica el algoritmo J48 sobre este conjunto de datos donde se analizará la matriz de confusión, como se observa en la Fig.10. Para las instancias clasificadas como ataques tenemos un 99% de confiabilidad, así como un 99.8% de confiabilidad en las instancias categorizadas como tráfico normal y un 13.1% en la matriz de desconocidos, también se observa que la mayoría de tráfico desconocido fue clasificado como ataque y no como tráfico normal por lo que hay que se debería tener una mayor cautela al evaluar tráfico con estas características.

```
=== Evaluation on training set ===

Time taken to test model on training data: 0.48 seconds

=== Summary ===

Correctly Classified Instances      128145               99.4289 %
Incorrectly Classified Instances       736                0.5711 %
Kappa statistic                          0.988
Mean absolute error                      0.0067
Root mean squared error                  0.0578
Relative absolute error                  2.0983 %
Root relative squared error             14.4857 %
Total Number of Instances           128881

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,999    0,014    0,991      0,999    0,995      0,988    0,999     0,999     attack
               0,998    0,000    1,000      0,998    0,999      0,998    1,000     0,999     normal
               0,131    0,000    0,730      0,131    0,222      0,308    0,983     0,264     unknown_attack
Weighted Avg.  0,994    0,009    0,993      0,994    0,993      0,988    0,999     0,995

=== Confusion Matrix ===

     a     b     c   <-- classified as
 79385    16    34 |     a = attack
    77 48668     0 |     b = normal
   609     0    92 |     c = unknown_attack
```

Fig 10. Matriz de confusión del algoritmo j48

Adicionalmente se realizó un análisis con el algoritmo de expectación maximización (EM), se generaron 3 clústeres como se observa en la Fig.11. donde se analizó el atributo duración observando que el promedio de duración del clúster 0 y 1 es menor al promedio del clúster 2.

```
=== Clustering model (full training set) ===


EM
==

Number of clusters: 3
Number of iterations performed: 2


                                Cluster
Attribute                            0          1          2
                                (0.28)     (0.37)     (0.35)
============================================================
Duracion
  mean                          4.8918     0.7144    20.9149
  std. dev.                    32.0406     9.8541    78.9076
```

Fig 11. Resultado del algoritmo NM- Respecto al atributo duración

En el resultado que se observa en la Fig.12. respecto a la clasificación se observa que los clústeres 0 y 1 poseen 32077 y 47359 instancias respectivamente clasificadas como ataques, lo cual al compararlo con la variable de duración podemos distinguir que existe una mayor probabilidad de que se considere tráfico malicioso una conexión con menor tiempo de duración, estos resultados son contrastados con los resultados obtenidos al realizar la función avg de PostgreSQL para obtener el tiempo promedio de duración. Estos resultados serán de gran ayuda a Tensorflow para poder aprender y distinguir el tiempo de demora en una conexión normal de una maliciosa para así poder realizar una auditoría a dicha conexión.

```
FLAG
  SF                        21780.2007   281.2781 23433.5212
  S0                         2785.7119 41537.2881        3
  RSTOS0                      755.8374   522.1626      434
  RST0                       6714.5936   236.1999 15613.2065
  RSTR                        267.9973          2  1735.0027
  S1                               489         81       14
  RSTRH                        19.9332   128.0668      420
  REJ                        1000.3438   830.6562       11
  S3                                2          1       72
  OTH                        2124.3602  3763.6398        4
  SHR                          17.0114          3  3764.9886
  SH                                6         35       17
  S2                                2          1       11
  [total]                   35964.9896 47422.2915 45532.7189
Clasificacion
  attack                    32077.7085 47359.2915        1
  normal                     3184.2811         43 45520.7189
  unknown_attack                   693         10        1
  [total]                   35954.9896 47412.2915 45522.7189


Time taken to build model (full training data) : 12.89 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       36263 ( 28%)
1       47372 ( 37%)
2       45246 ( 35%)
```

Fig 12. Resultado del algoritmo NM- Con las clasificación

Se observa en la Fig.13. Las urls guardadas en PostgreSQL, estas son obtenidas de un conjunto de datos de páginas de seguridad informática que clasifican estas direcciones del tipo maliciosa, una vez guardadas se procede a auto generar reglas para Suricata-IDS, para evitar que las conexiones tengan acceso a estas direcciones ya que podrían comprometer toda la seguridad.

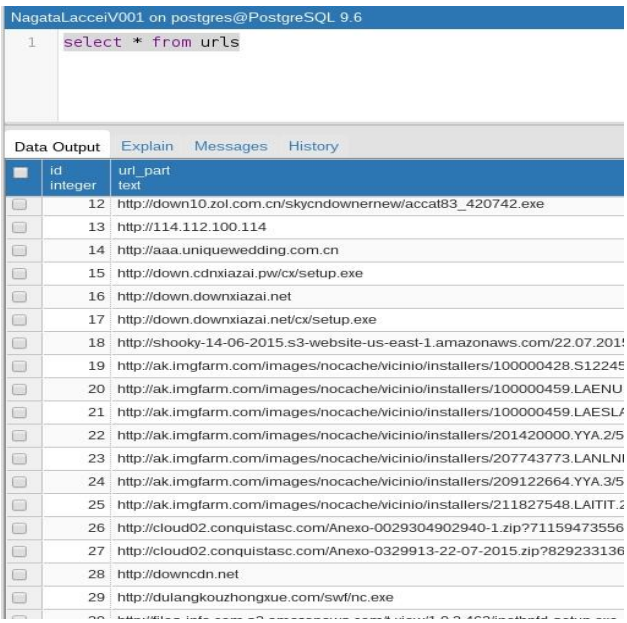


Fig 13. Almacenamiento en base de datos de direcciones maliciosas

En la Fig.14. se aprecia la auto generación de reglas de seguridad para Suricata-IDS, estas reglas fueron generadas a partir del conjunto de datos que fueron almacenados en la base de datos, cada generación de reglas tiene un atributo de Timestamp para poder verificar luego las mismas y llevar un control de la fecha y hora de creación.

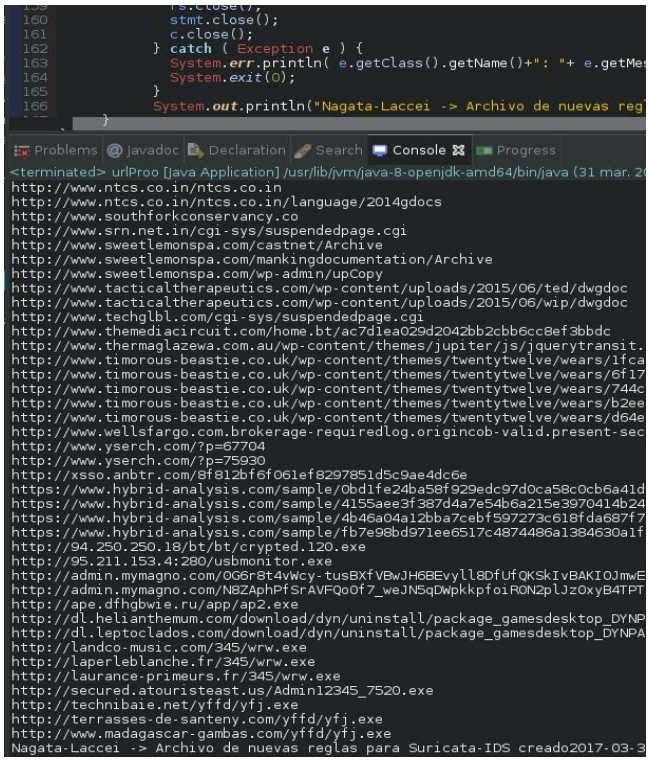


Fig 14. Auto generación de reglas para Suricata-IDS

En la Fig.15. se observa el archivo generado que se almacena en el directorio donde se encuentra la configuración de Suricata-IDS, como se aprecia en la Fig.16.

```
drop dns any any -> any any (msg:DNS ALERTA
SEGURIDAD;content:Mark;classtype:policy-violation; sid:39398144;
rev:1;)
drop tcp any any -> any any (msg: "sitio malicioso";
content:Mark;classtype:policy-violation; sid:39398144)

drop dns any any -> any any (msg:DNS ALERTA
SEGURIDAD;content:http://www.bbqdzx.com;classtype:policy-
violation; sid:39398145; rev:1;)
drop tcp any any -> any any (msg: "sitio malicioso"; content:
http://www.bbqdzx.com;classtype:policy-violation; sid:39398145)

drop dns any any -> any any (msg:DNS ALERTA
SEGURIDAD;content:http://bbqdzx.com;classtype:policy-violation;
sid:39398146;)
drop tcp any any -> any any (msg: "sitio malicioso"; content:
content:http://bbqdzx.com;classtype:policy-violation; sid:39398146;
sid:39398144)

drop dns any any -> any any (msg:DNS ALERTA
SEGURIDAD;content:http://cnzhiyuan.com;classtype:policy-violation;
sid:39398147;)
drop tcp any any -> any any (msg: "sitio malicioso"; content:
http://cnzhiyuan.com;classtype:policy-violation; sid:39398144;
sid:39398147)

drop dns any any -> any any (msg:DNS ALERTA
SEGURIDAD;content:http://down.downcdn.net/cx/setup.exe;classtype:p
olicy-violation; sid:39398148; rev:1;)
drop tcp any any -> any any (msg: "sitio malicioso"; content:
http://down.downcdn.net/cx/setup.exe;classtype:policy-violation;
sid:39398148)
```

Fig 15. Generación de reglas para eliminar conexiones a webs maliciosas

```
root@KPC:/etc/suricata/rules# ls
app-layer-events.rules          emerging-rpc.rules
botcc.portgrouped.rules         emerging-scada.rules
botcc.rules                     emerging-scan.rules
BSD-License.txt                 emerging-shellcode.rules
ciarmy.rules                    emerging-smtp.rules
classification.config           emerging-snmp.rules
compromised-ips.txt             emerging-sql.rules
compromised.rules               emerging-telnet.rules
decoder-events.rules            emerging-tftp.rules
dnp3-events.rules               emerging-trojan.rules
dns-events.rules                emerging-user_agents.rules
drop.rules                      emerging-voip.rules
dshield.rules                   emerging-web_client.rules
emerging-activex.rules          emerging-web_server.rules
emerging-attack_response.rules  emerging-web_specific_apps.rules
emerging-chat.rules             emerging-worm.rules
emerging.conf                   files.rules
emerging-current_events.rules   gen-msg.map
emerging-deleted.rules          gpl-2.0.txt
emerging-dns.rules              http-events.rules
emerging-dos.rules              modbus-events.rules
emerging-exploit.rules          NagataLacceiRules_0_2017-03-29-00-52-10.rules
emerging-ftp.rules              NagataLacceiRules_0_2017-03-29-19-13-56.rules
emerging-games.rules            NagataLacceiRules_0_2017-03-31-01-13-08.rules
emerging-icmp_info.rules        NagataLacceiRules_0_2017-03-31-01-27-27.rules
```

Fig 16. Resultado de generación de reglas personalizadas

En la Fig.17. se aprecia como Tensorflow aprendió a detectar conexiones sospechosas a partir de los resultados generados por el análisis de datos del algoritmo J48, estos resultados son los que entrenaran a tensorflow ya que este estará en constante monitoreo de las nuevas conexiones al igual que Suricata-IDS, tensorflow al detectar conexiones que encajen con sus parámetros aprendidos clasificara esta conexión como sospechosa y generara nuevos hilos de auditoria generando reglas específicas en Suricata-IDS para evaluar el comportamiento de dicha conexión.

```
192.168.1.22
Trafico Normal
192.168.1.45
['Trafico Sospechoso']
Generando hilo de auditoria #1
192.168.1.71
['Trafico Sospechoso']
Generando hilo de auditoria #2
192.168.1.40
['Trafico Sospechoso']
Generando hilo de auditoria #3
192.168.1.17
Trafico Normal
192.168.1.89
['Trafico Sospechoso']
Generando hilo de auditoria #4
192.168.1.9
Trafico Normal
192.168.1.29
['Trafico Sospechoso']
Generando hilo de auditoria #5
```

Fig 17. Detección de tráfico sospechoso por tensorflow

IV. Conclusiones

- Los sistemas de detección de intrusos en redes SDN muchas veces no tienen en cuenta los detalles de la conexión para clasificarla como ataques o tráfico normal, es por ello que la implementación con algoritmos como J48 y Tensorflow da un mayor detalle acerca de la probabilidad de que una conexión sea maliciosa.
- La generación de reglas para los IDS de manera automática supone una gran ventaja en la seguridad de las redes SDN, ya que se ahorrarán recursos valiosos.
- Obteniendo los detalles de las conexiones se puede evaluar actividades sospechosas de manera individual teniendo en cuenta casos anteriores donde el tráfico resulto ser malicioso.
- Identificar los patrones de tramas clasificadas como maliciosas puede suponer una gran ventaja para el análisis de seguridad de conexiones actuales.
- Teniendo en cuenta las características de tráfico malformado que podrían ser considerados ataques, se puede tener una mejor gestión de seguridad en la red SDN.

Agradecimientos

Quisiera expresar mi sincera gratitud al Magister José David Esquicha Tejada, por el apoyo y asistencia para la redacción y edición del presente trabajo de investigación.

REFERENCIAS

[1] L. Lahoti, C. Chandankhede, and D. Mukhopadhyay, "Modified apriori approach for evade network intrusion detection system," *Proc. - 2014 13th Int. Conf. Inf. Technol. ICIT 2014*, pp. 374–378, 2014.

[2] P. Van Trung, T. T. Huong, D. Van Tuyen, D. M. Duc, N. H. Thanh, and A. Marshall, "A multi-criteria-based DDoS-attack prevention solution using software defined networking," *2015 Int. Conf. Adv. Technol. Commun.*, pp. 308–313, 2015.

[3] T. Tran, I. Aib, E. Al-Shaer, and R. Boutaba, "An evasive attack on SNORT flowbits," *Proc. 2012 IEEE Netw. Oper. Manag. Symp. NOMS 2012*, pp. 351–358, 2012.

[4] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "LineSwitch: Tackling Control Plane Saturation Attacks in Software-Defined Networking," *IEEE/ACM Trans. Netw.*, pp. 1–14, 2016.

[5] J. Ashraf and S. Latif, "Handling Intrusion and DDoS Attacks in Software Defined Networks Using Machine Learning Techniques," *2014 Natl. Softw. Eng. Conf. (Nsec - 2014)*, pp. 55–60, 2014.

[6] L. Barki, A. Shidling, N. Meti, D. G. Narayan, and M. M. Mulla, "Detection of Distributed Denial of Service Attacks in Software Defined Networks," no. December, pp. 2576–2581, 2016.

[7] S. Luo, J. Wu, J. Li, and B. Pei, "A Defense Mechanism for Distributed Denial of Service Attack in Software-Defined Networks," *2015 Ninth Int. Conf. Front. Comput. Sci. Technol.*, pp. 325–329, 2015.

[8] S. Luo, J. Wu, J. Li, and L. Guo, "A multi-stage attack mitigation mechanism for software-defined home networks," *IEEE Trans. Consum. Electron.*, vol. 62, no. 2, pp. 200–207, 2016.

[9] M. Z. Masoud, Y. Jaradat, and I. Jannoud, "On preventing ARP poisoning attack utilizing Software Defined Network (SDN) paradigm," *2015 IEEE Jordan Conf. Appl. Electr. Eng. Comput. Technol.*, pp. 1–5, 2015.

[10] P. M. Mohan, T. J. Lim, and M. Gurusamy, "Fragmentation-Based Multipath Routing for Attack Resilience in Software Defined Networks," *2016 IEEE 41st Conf. Local Comput. Networks*, pp. 583–586, 2016.

[11] T. V. Phan, T. Van Toan, D. Van Tuyen, T. T. Huong, and N. H. Thanh, "OpenFlowSIA: An optimized protection scheme for software-defined networks from flooding attacks," *2016 IEEE 6th Int. Conf. Commun. Electron. IEEE ICCE 2016*, pp. 13–18, 2016.

[12] F. Reynaud, F. X. Aguessy, O. Bettan, M. Bouet, and V. Conan, "Attacks against Network Functions Virtualization and Software-Defined Networking: State-of-the-art," *IEEE NETSOFT 2016 - 2016 IEEE NetSoft Conf. Work. Software-Defined Infrastruct. Networks, Clouds, IoT Serv.*, pp. 471–476, 2016.

[13] D. Sattar, A. Matrawy, and O. Adeojo, "Adaptive Bubble Burst (ABB): Mitigating DDoS attacks in Software-Defined Networks," *2016 17th Int. Telecommun. Netw. Strateg. Plan. Symp.*, pp. 50–55, 2016.

[14] S. Singh, R. A. Khan, and A. Agrawal, "Prevention mechanism for infrastructure based Denial-of-Service attack over software Defined Network," *Int. Conf. Comput. Commun. Autom. ICCCA 2015*, pp. 348–353, 2015.

[15] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and Software-Defined Networking," *Comput. Networks*, vol. 81, pp. 308–319, 2015.

[16] X. Wang, M. Chen, and C. Xing, "SDSNM: A Software-Defined Security Networking Mechanism to Defend against DDoS Attacks," *Proc. - 2015 9th Int. Conf. Front. Comput. Sci. Technol. FCST 2015*, pp. 115–121, 2015.

[17] R. In, "Distributed Denial of Service Attacks in Software-Defined Networking with Cloud Computing," no. April, pp. 52–59, 2015.

[18] J. Chukwu, "IDSaaS in SDN : Intrusion Detection System as a Service in Software Defined Networks," 2016.

[19] V. Varadharajan and U. Tupakula, "On the Design and Implementation of an Integrated Security Architecture for Cloud with Improved Resilience," *IEEE Trans. Cloud Comput.*, vol. 7161, no. c, pp. 1–1, 2016.

[20] M. Masoud, Y. Jaradat, and A. Q. Ahmad, "On Tackling Social Engineering Web Phishing Attacks Utilizing Software Defined Networks (SDN) Approach," 2016.

[21] C. Benson, «https://msdn.microsoft.com/en-us/library/cc723506.aspx,» 2000. [En línea]. Available: https://msdn.microsoft.com/en-us/library/cc723506.aspx. [Último acceso: 05 2017].