Performance Analysis of Packet Sniffing Techniques Applied to Network Monitoring

D. Álvarez, P. Nuño, F. G. Bulnes and J. C. Granda

***Abstract*— Network monitoring based on packet sniffing is one of the most useful techniques applied by system administrators and security analysts in order to identify threats within a local network. Despite being supposedly a simple task, it could sometimes be a highly resource consuming process. In this paper, the use of two sniffing techniques, *raw sockets* and *scapy*, to achieve better performance in terms of maximum capture packet rate are analyzed and compared. Furthermore, both techniques are optimized by using *BSD Packet Filtering* to improve packet capture, and a multicore architecture in order to reduce the exposition to denegation of service attacks. Finally, a system based on those techniques that is able to automatically detect layers 2 and 3 common vulnerabilities and attacks within the scope of corporative networks is developed. The result is an enhanced system focused on link and network layers that can be deployed in corporative environments.**

***Index Terms*—network monitoring, network sniffer, security.**

I. Introducción

En los últimos años el auge adquirido por Internet ha provocado un aumento significativo del número de ataques informáticos a nivel mundial. Debido a ello, la ciberseguridad se ha convertido en un campo de especial relevancia a nivel gubernamental, corporativo y personal. El impacto económico causado por el cibercrimen, en sus diferentes variedades, se estima en decenas de miles de millones de dólares anuales [1]. Por tanto, en entornos relevantes como el ámbito corporativo, se deben establecer determinados mecanismos de seguridad capaces de anular, o mitigar, la probabilidad de sufrir un ataque, incluso si este fuese realizado desde su seno.

Las amenazas que comprometen la seguridad de una red corporativa desde su interior sacan partido de las características de esos entornos, las cuales se explican en [2]. En primer lugar, sin la implementación de redes locales virtuales (VLANs), todos los equipos de la red corporativa comparten el dominio de difusión en capa 2. En segundo lugar, no existen mecanismos que permitan autenticar mensajes en capa 2. Para solventar los riesgos que esto plantea, es necesario adquirir hardware de red con múltiples funcionalidades de seguridad, como la seguridad de puerto (p. ej. activando *Port Security*), el aseguramiento total del protocolo *Dynamic Host Configuration Protocol* (DHCP) (p. ej. configurando *DHCP Snooping*) y el aseguramiento del protocolo *Address Resolution Protocol* (ARP) (p. ej. configurando *Dynamic ARP Inspection* (DAI)).

Las funcionalidades citadas anteriormente conllevan, en tercer lugar, un diseño y configuración del entorno de red corporativo que puede resultar complejo, pudiendo desembocar en errores de los administradores de la red de tal manera que la red corporativa mantenga su disponibilidad, pero deje abierta la posibilidad a que dichas condiciones de error sean explotadas por los atacantes. Un posible método adicional de defensa, que es perfectamente complementario a las funcionalidades anteriores, es analizar la red de comunicaciones de manera pasiva, monitorizando todos los paquetes que circulen por ella, para detectar configuraciones defectuosas o erróneas, así como intentos de ataque. Dicho método se basa en un elemento individual, denominado *packet sniffer*, encargado de capturar los paquetes que circulan por la red para su posterior procesamiento [3].

Los sistemas de detección son parte fundamental de las medidas defensivas que deben estar implementadas en una red para dotarla de resiliencia [4]. En ese trabajo, los autores proponen el uso de sistemas de detección ligeros para ser desplegados en un primer nivel de defensa, con el objetivo de enviar rápidamente alertas que puedan ser procesadas por otros sistemas más complejos, o por los administradores de la red. Existen diversas herramientas de *sniffing* que permiten llevar a cabo una monitorización pasiva del tráfico que circula por la red. Recientemente han sido publicados artículos científicos donde se describen y analizan varias de dichas herramientas, y se compara su rendimiento [5] [6]. No obstante, en esos estudios no se proporciona información adicional sobre técnicas que permitan crear un *sniffer* personalizado y ligero con unos mínimos conocimientos de programación.

En ese sentido, la principal técnica que permite construir un *sniffer* personalizado es el uso de *raw sockets.* Además, dentro del ámbito de la seguridad, la librería *scapy* está ganando mucha popularidad y su uso es cada vez más creciente [7]. Mediante ambos enfoques se puede implementar un *sniffer* centrado exclusivamente en un número concreto de protocolos y eventos, y que además puede ser integrado con protocolos de gestión y notificación de incidencias de la red.

Por tanto, en este artículo se presenta un análisis y evaluación del rendimiento de la utilización de la librería *scapy* y el uso de *raw sockets* para implementar una herramienta de *sniffing* personalizada. Además, se estudian mecanismos de mejora de rendimiento en ambas técnicas como la aplicación de filtros *Berkeley Packet Filter* (BPF) y el aprovechamiento de una arquitectura multinúcleo.

Para realizar la comparativa entre ellas, ambas técnicas de *sniffing* han sido implementadas en un sistema de detección

Submission date: January 14, 2020

D. Álvarez, University of Oviedo, Asturias, Spain, uo237670@uniovi.es.

P. Nuño, Department of Computer Science and Engineering, University of Oviedo, Asturias, Spain, nunopelayo@uniovi.es, Corresponding author.

F. G. Bulnes, Department of Computer Science and Engineering, University of Oviedo, Asturias, Spain, bulnes@uniovi.es.

J. C. Granda, Department of Computer Science and Engineering, University of Oviedo, Asturias, Spain, jcgranda@uniovi.es.

capaz de identificar vulnerabilidades y patrones de ataque en las capas 2 y 3 del modelo *Open Systems Interconnection* (OSI), derivados en su mayoría de errores en la configuración de los dispositivos de red. El propósito del sistema de monitorización es doble. En primer lugar, establece reglas para la detección de vulnerabilidades potencialmente explotables por un atacante, posibilitando una mitigación proactiva de las amenazas. En segundo lugar, identifica diversos patrones de ataque, estableciendo ante ellos un mecanismo reactivo de respuesta.

Los resultados muestran que se han alcanzado unas tasas de captura y procesamiento de paquetes que posibilitan el despliegue de las técnicas propuestas en multitud de entornos de red corporativos y, debido a la naturaleza del tráfico que es monitorizado por dichas técnicas, pueden operar en redes de diferente tamaño.

II. Trabajo Relacionado

Dentro de la literatura científica se pueden encontrar diversos trabajos cuyo objetivo es analizar el funcionamiento interno que presentan los principales *sniffers* utilizados para la captura de paquetes [8] [9]. Estudios posteriores como [10] amplían en cierta medida las líneas de investigación, planteando la posibilidad de desarrollar *sniffers* personalizados que proporcionen un rendimiento óptimo.

Por otro lado, debido a la necesidad de implementar una eficiente monitorización de la red, se han realizado trabajos destinados a obtener una mejora de prestaciones de los sistemas de detección (IDS) y prevención de intrusiones (IPS) que analizan el tráfico existente en busca de anomalías [11]. Aunque estas herramientas son muy potentes, reportan un gran número de falsos positivos, lo que ha dado lugar a estudios cuyo objetivo es reducir el número de alertas ante ataques inexistentes generados por los sistemas IDS/IPS [12].

En [13] y [14] los autores evalúan y comparan el rendimiento de la captura de paquetes combinando distintos procesadores de la familia Intel y AMD junto con sistemas operativos Linux y FreeBSD. No obstante, no se analiza cómo influye en el rendimiento los diferentes enfoques de implementación de una técnica de *sniffing*.

El rendimiento de varias extensiones de la librería *libpcap* para construir un *sniffer* para arquitecturas multinúcleo es evaluado en [15]. Los autores comparan el rendimiento de la librería sobre tres tipos de sockets: los *sockets* por defecto de Linux, PF_RING *sockets* y PFQ *sockets*.

Dos técnicas basadas en *zero_copy* y *polling* se utilizan para diseñar un *sniffer* personalizado cuyo rendimiento es comparado con la librería *libpcap* en [16]. Como resultado, se obtiene una mayor tasa de paquetes capturados a la vez que se reduce el consumo de CPU. No obstante, los autores destacan que el rendimiento al manejar paquetes de pequeño tamaño, como los relacionados con las capas 2 y 3 en el ámbito de una red corporativa, necesita ser optimizado.

Por último, en [17] se diseña y analiza la sobrecarga inducida en los clientes finales por un *sniffer* personalizado, construido sobre la librería *libpcap*, cuyo objetivo es ser desplegado como parte de una red distribuida de *microsniffers* para monitorizar el tráfico a gran escala.

La librería *libpcap* utilizada en los artículos anteriormente mencionados es la que usa *scapy* de manera nativa en entornos Unix. Sin embargo, ninguno de dichos artículos evalúa el rendimiento de una técnica de *sniffing* implementada directamente sobre *scapy*. Por tanto, hasta donde alcanza el conocimiento de los autores, este trabajo es el primero que evalúa el rendimiento de diversas técnicas, siendo una de ellas la librería *scapy*, para construir un *sniffer* personalizado con el objetivo de evaluar las configuraciones y detectar ataques en el ámbito de redes corporativas.

TABLA I
PROTOCOLOS MONITORIZADOS

Propósito/Familia	Protocolos
Resolución de direcciones físicas	ARP
Descubrimiento de nodos vecinos en capa 2	LLDP, CDP, FDP
Control de la redundancia en capa 2	STP
Configuración y administración de VLANs	DTP, VTP, MVRP
Redundancia de primer salto (FHRP)	HSRP, VRRP, GLBP
Enrutamiento	RIP, OSPF, EIGRP
Configuración dinámica de parámetros de red	DHCP

III. Arquitectura Del Sistema

Este artículo analiza el rendimiento proporcionado por la librería *scapy* y por los *raw sockets* a la hora de desarrollar un *sniffer* de red personalizado, junto con la implementación de optimizaciones adicionales para mejorar el rendimiento global de ambas técnicas. Con el objetivo de estudiar el comportamiento de las dos técnicas de captura de paquetes, y las mejoras de rendimiento propuestas, se ha desarrollado un sistema de detección capaz de identificar vulnerabilidades y ataques comunes en una red corporativa.

Para ello se ha realizado, en primer lugar, un análisis de los protocolos más utilizados en las capas de enlace y de red en el ámbito corporativo. En segundo lugar, se ha implementado un módulo capaz de capturar, de acuerdo a las dos técnicas planteadas, los paquetes pertenecientes a los protocolos más relevantes derivados del análisis anterior. El catálogo de protocolos monitorizados se muestra en la Tabla I.

La presencia de algunos de esos protocolos en el entorno conmutado, como por ejemplo DTP, CDP o los protocolos de enrutamiento, es sinónimo de una mala configuración y, por tanto, de la existencia de vulnerabilidades en los dispositivos que componen la red corporativa. En otras situaciones, y en función del diseño de la red, la mera presencia de tráfico de cierto protocolo puede delatar la existencia de un equipo en la red intentando realizar ataques como sería el caso de detectar tráfico HSRP en un entorno conmutado que por su diseño carece de puerta de enlace de respaldo. Además de la detección de los protocolos mostrados en la Tabla I, el sistema es capaz de identificar y notificar los ataques indicados en la Tabla II.

En la Fig. 1 se muestra el diagrama funcional de bloques del sistema de detección desarrollado, compuesto por seis módulos relacionados entre sí. Como se puede observar, el núcleo del sistema es el *script* de monitorización, que es el encargado de implementar la captura de paquetes de acuerdo a cada técnica, así como su posterior tratamiento.

TABLA II
ATAQUES DETECTADOS

Nombre	Descripción
CDP Flooding	Inundación del entorno conmutado y saturación de las tablas MAC con tráfico CDP
ARP Spoofing	Manipulación ilegítima de las tablas ARP
DHCP Starvation	Múltiples solicitudes de direcciones IP vía DHCP
DHCP Spoofing	Asignación ilegítima de direcciones IP vía DHCP
Router Impersonation	Anuncio de rutas desde dirección IP no confiable
FHRP Impersonation	Anuncio de puerta de enlace de respaldo desde dirección IP no confiable

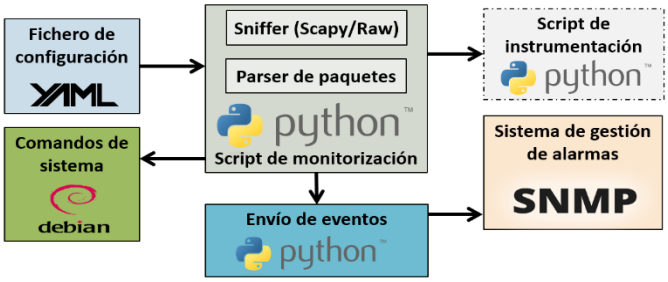


Fig. 1. Diagrama funcional del sistema de detección de vulnerabilidades y ataques en las capas 2 y 3 del modelo OSI.

El *script* de monitorización ajusta su comportamiento en base a una serie de parámetros modificables recibidos a través de un fichero de configuración. Además, interacciona con el sistema operativo de la máquina en la que corre para recabar información, y genera eventos en forma de notificaciones que son enviadas a un servidor centralizado como respuesta al descubrimiento de vulnerabilidades o ante la detección de ataques. Estas notificaciones se envían haciendo uso del protocolo *Simple Network Management Protocol* (SNMP). Finalmente, el *script* de monitorización inicializa un módulo de instrumentación para analizar el rendimiento de cada técnica en la captura de paquetes, aunque tras ello no intercambia ningún tipo de información con el mismo.

En la Fig. 1 se ilustran las tecnologías utilizadas, siendo el lenguaje *python* el elegido para implementar los módulos de monitorización, envío de eventos e instrumentación. Esta decisión de diseño implica que, debido a la presencia del GIL (*Global Interpreter Lock*) en el intérprete de *python*, no es posible aprovecharse de múltiples hilos concurrentes ya que el GIL fuerza que se ejecuten secuencialmente. Sin embargo, existen alternativas que permiten aprovechar la existencia de múltiples núcleos en el equipo anfitrión en el que se despliega el sistema, las cuales han sido implementadas como optimizaciones en el sistema desarrollado.

Las ventajas de desplegar como sistema de detección un *sniffer* personalizado mediante las técnicas propuestas en este artículo son varias, y se resumen en la Tabla III. En primer lugar, es un sistema ligero al estar enfocado exclusivamente en ciertos tipos de tráfico, y es integrable mediante el uso del protocolo SNMP con otros sistemas de detección o gestión más complejos, aspecto del que adolecen los trabajos citados en la sección II. Además, es posible variar su respuesta entre ejecuciones ante ciertos protocolos mediante el uso de ficheros de configuración. Así, es posible dejar de monitorizar protocolos si estos ya no son de interés, siendo

TABLA III
VENTAJAS E INCONVENIENTES DE UN SNIFFER PERSONALIZADO

Ventajas	Inconvenientes
Reduce el consumo de recursos	Sin experiencia previa de su uso por parte de terceros
Adaptable de manera exclusiva a ciertos protocolos	Requiere pruebas de validación e implantación
Integrable con sistemas de detección	Limitado al entorno conmutado
Flexible	No extrapolable a otros entornos

sencillo ampliar este comportamiento para que se realice en tiempo real. Por otra parte, existen inconvenientes comunes a otros trabajos relacionados que no son software comercial, y es que no existe experiencia previa que garantice su fiabilidad, luego requiere de ser probado y validado. Además, las técnicas propuestas están planteadas exclusivamente para monitorizar protocolos propios del entorno conmutado, y no para analizar tráfico proveniente del exterior de la red corporativa. Finalmente, no es sencillo para un administrador con conocimientos de programación extender el ámbito de actuación de estas técnicas al tráfico proveniente del exterior debido a la dificultad de programarlas para que identifiquen el tráfico objetivo de manera precisa.

IV. TÉCNICAS DE SNIFFING

El *script* de monitorización es el núcleo del sistema de detección desarrollado. Consta de un subsistema de captura de paquetes (*sniffer*), y de un subsistema de procesamiento de los mismos *(parser)*. En lo que respecta a la captura de paquetes se han considerado dos técnicas diferentes: el uso de la librería *scapy* y el empleo directo de *raw sockets*.

En primer lugar, el uso de la librería *scapy* es debido a que está considerado como uno de los *sniffers* más utilizados en la actualidad, junto con *Wireshark* [7], si bien solo *scapy* permite realizar implementaciones personalizadas de manera sencilla e intuitiva con unos conocimientos mínimos de programación. Por ejemplo, en [7] los autores desarrollan un sistema de detección de intrusiones (IDS) personalizado y, tras analizar diversos *sniffers*, seleccionan la alternativa de *scapy* para comparar el rendimiento de su prototipo. En segundo lugar, dado que la librería *scapy* está desarrollada en *python*, se ha decidido realizar la implementación de la técnica de *raw sockets* en ese mismo lenguaje para evitar que las diferencias en los resultados puedan ser atribuibles al uso de diferentes lenguajes de programación.

A pesar de que *scapy* utiliza *raw sockets* de manera nativa, difiere de usarlos directamente en un paso previo a la manipulación y procesado de cada paquete recibido. Por ello, el rendimiento ofrecido al utilizar un *raw socket* de manera directa se presupone superior a utilizar la librería *scapy*. En cambio, el tratamiento de los paquetes por el *parser* una vez han sido capturados es idéntico, con independencia de la técnica utilizada en el *sniffer*. La identificación de protocolos se lleva a cabo en base al valor que adquieren los campos presentes en las cabeceras de cada paquete. Por tanto, este procedimiento no plantea diferencias de rendimiento ya que es común a ambas técnicas.

En la Fig. 2 se muestra el proceso seguido por cada paquete capturado por el *sniffer* al implementar la primera técnica, basada en el uso de la librería *scapy*. El paquete, tras ser capturado físicamente por la tarjeta de red, se envía a su *driver* correspondiente y, tras ello, abandona el espacio de núcleo. El primer paso una vez que el paquete es copiado al espacio de usuario es realizar una deserialización de los datos, tras la cual el paquete puede ser procesado por el *parser* para el análisis de tráfico.

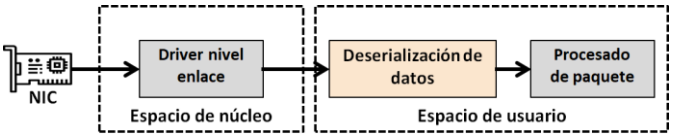


Fig. 2. Procesamiento en el *sniffer* basado en la librería *scapy*.

La deserialización de datos llevada a cabo por *scapy* para cada paquete incluye un mapeo de los valores contenidos en cada cabecera del paquete a un formato más legible para el ser humano. De este modo, en lugar de obtener una secuencia de *bytes* para un determinado campo, *scapy* es capaz de mostrar su valor de manera más intuitiva. Así, por ejemplo, se muestra como dirección IP de origen un valor decimal de cuatro octetos en lugar de su codificación en hexadecimal.

A pesar de aportar una funcionalidad interesante desde el punto de vista del usuario, este proceso de mapeo de datos implica la ejecución de varias instrucciones en la máquina en la que corre el *script* de monitorización. Dado que toda instrucción tiene un coste computacional, el consumo de recursos se ve forzosamente incrementado por esta deserialización de los datos. Por su parte, en la segunda de las técnicas de captura de paquetes implementada, se crea directamente en el *sniffer* un *raw socket*. A través de dicho *socket* es posible recibir paquetes en forma de secuencias de *bytes* que pueden ser manipulables directamente, sin sufrir ningún procesamiento previo, al contrario que ocurría con la primera técnica basada en el uso de la librería *scapy*.

En la Fig. 3 se presenta el proceso seguido por un paquete capturado a través de la técnica basada en *raw sockets*. Se puede observar como el proceso es idéntico a la técnica anterior, salvo la sustitución del bloque de deserialización de datos por un bloque de decodificación. En este caso, la decodificación de datos consiste únicamente en desempaquetar el contenido del paquete en formato hexadecimal, lo que provoca una reducción de las instrucciones ejecutadas por el *sniffer*. Con ello, se consigue rebajar el consumo de recursos necesarios para una misma tasa de recepción de paquetes, mejorando así su rendimiento. Asimismo, una vez decodificado, el paquete puede ser procesado de manera directa por el *parser* de igual modo que en la técnica basada en *scapy*.

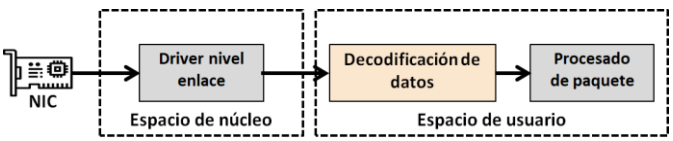


Fig. 3. Procesamiento en el *sniffer* basado en *raw sockets*.

V. Optimizaciones De Las Técnicas

A continuación, se describen los mecanismos que han sido implementados para mejorar el rendimiento de ambas técnicas. El objetivo fundamental es conseguir una elevada tasa de recepción de tráfico empleando para ello el mínimo de recursos computacionales posible.

A. *Filtrado de Berkeley (BPF)*

El filtrado de paquetes BSD, más conocido como BPF, es una técnica propuesta que permite descartar paquetes dentro del espacio de memoria del núcleo [18]. Este filtrado se lleva a cabo con independencia de la técnica implementada en el *sniffer*, como paso previo tanto para la deserialización como para la decodificación de los datos. Tal y como se muestra en la Fig. 4, el filtrado evita la copia de paquetes no deseados en los *buffers* de salida del espacio de memoria del núcleo y, por tanto, su copia al espacio de memoria de usuario.

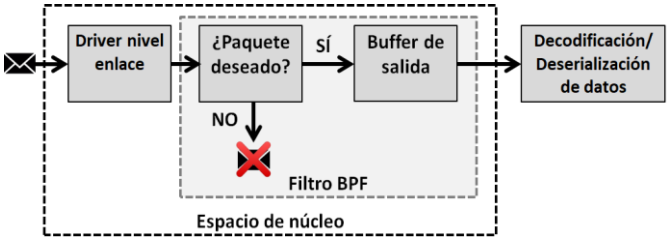


Fig. 4. Flujo seguido por un paquete en función de la decisión tomada por el filtrado BPF.

El resultado tras la aplicación del filtrado BPF es una mejora en el rendimiento del *sniffer*, ya que se descartan en el inicio de la cadena de procesado aquellos paquetes que no pertenecen a los protocolos de capas 2 y 3 del modelo OSI contemplados en el *script* de monitorización. Por tanto, uno de los beneficios inmediatos del uso de BPF es que, para provocar la saturación del sistema debido al procesamiento del tráfico recibido, este debe pertenecer exclusivamente a los protocolos contemplados, típicamente ligeros y de naturaleza esporádica. El tráfico restante, irrelevante para detectar vulnerabilidades y ataques, pero que podría suponer una carga, es descartado sin apenas coste computacional.

B. *Arquitectura multinúcleo*

El script de monitorización tiene una limitación derivada de su diseño, ya que debido al GIL no es posible aprovechar múltiples hilos para ejecutar tareas de manera concurrente. Esto provoca que en el sistema inicial el *script* se ejecute sobre un único hilo asignado a un solo núcleo de la máquina en la que haya sido desplegado, desperdiciando por tanto recursos computacionales presentes en esa máquina.

La solución para optimizar el rendimiento del sistema pasa por rediseñar su arquitectura. La mejora introducida consiste en implementar el multiprocesamiento en *python* mediante el aprovechamiento de múltiples núcleos existentes en la máquina anfitriona. En este nuevo diseño, por cada protocolo se instancia un intérprete de *python* independiente. Así, es posible distribuir el coste computacional derivado de la detección de los protocolos a monitorizar entre los núcleos disponibles en la máquina utilizada. Tal y como se muestra

en la Fig. 5, disponer de un mayor número de núcleos permite una mejor distribución de la carga, beneficiando así el desempeño global del sistema. No obstante, los procesos asociados a dos o más protocolos pueden correr sobre un mismo núcleo si las circunstancias lo requieren, aunque la capacidad de detección de cada núcleo disminuye progresivamente en función del aumento en el número de protocolos a analizar en dicho núcleo.

Por tanto, se produce una mejora al pasar de una solución monolítica, en la que un único núcleo del computador captura y procesa el tráfico de todos los protocolos, a una solución capaz de aprovechar los recursos disponibles en la máquina anfitriona, distribuyendo la carga entre todos los núcleos.

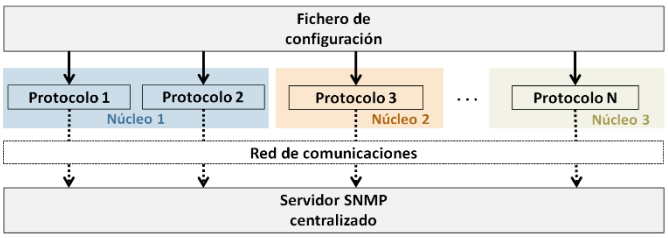


Fig. 5. Arquitectura propuesta para el aprovechamiento de múltiples núcleos por parte del *script* de monitorización.

La optimización basada en el filtrado BPF se realiza con una configuración diferenciada para cada protocolo. Así, los protocolos que se configuran para ser detectados en un proceso se configuran para ser descartados por los filtros BPF de los restantes procesos, evitando así operaciones innecesarias debido al solapamiento.

La posibilidad de establecer una configuración diferenciada de los filtros BPF a nivel de núcleo proporciona una mayor eficiencia a nivel de rendimiento, ya que la consecuencia de aplicar un filtro BPF es la reducción del tráfico a procesar, y una reducción del coste computacional. Al trasladar este paradigma desde un único núcleo a múltiples núcleos, el rendimiento aumenta y se reduce la capacidad de saturación del computador anfitrión, puesto que cada núcleo va a procesar un menor número de protocolos y, por tanto, un menor volumen de tráfico.

Por último, cabe destacar que este diseño multinúcleo aumenta la disponibilidad del sistema de detección en su conjunto. Debido al uso de múltiples núcleos, cada uno de ellos con protocolos asociados de manera exclusiva, es posible aislar aquellos protocolos que potencialmente pueden ser utilizados por un hipotético atacante para tratar de inutilizar el *sniffer* mediante una denegación de servicio (*DoS*). Un ejemplo de este tipo son los protocolos que puedan ser utilizados para inundar la red con tráfico de difusión, siendo el ejemplo más representativo el protocolo DHCP.

Así, en el diseño resultante se reserva un núcleo dedicado para cada protocolo que se estime como potencialmente peligroso debido a su posible aprovechamiento para realizar un *DoS* del propio *sniffer*. De este modo, el análisis de otros protocolos se realiza en el resto de los núcleos, y la disponibilidad tanto del *sniffer* en particular, como del sistema de detección en general, no se ven mermados ante intentos de provocar un ataque *DoS*.

C. Mitigación de ataques DoS

El diseño multinúcleo previene que el *sniffer* pueda sufrir un ataque *DoS* mediante la inyección de paquetes, pertenecientes a un protocolo analizado, a una tasa de envío suficientemente elevada. Un ejemplo de este tipo de incidente sería un ataque coordinado de *DHCP starvation*, que inunda la red con mensajes de difusión fraudulentos de tipo *DHCP Discovery*, y que son procesados ya que se encuentran dentro del catálogo de protocolos monitorizados. No obstante, el uso de múltiples núcleos no evita que el núcleo asociado con el protocolo objetivo de la realización del ataque *DoS*, DHCP en este ejemplo, se pueda saturar al procesar el tráfico asociado con dicho protocolo. Un problema derivado de esta situación es que, al situarse el uso de recursos en el núcleo afectado en un valor cercano al máximo, cabe la posibilidad de que mensajes de otros protocolos que estuviesen asignados a ese mismo núcleo no pudiesen ser procesados por el sistema. Este hecho impediría la detección de ataques concurrentes, o de vulnerabilidades sobrevenidas, relacionados con otros tipos de tráfico. Por tanto, esta situación podría desembocar en un *DoS* parcial que afectase a un subconjunto de los protocolos monitorizados por el *sniffer*.

Con el objetivo de complementar el aislamiento proporcionado por el diseño multinúcleo, y así mitigar la exposición a ataques *DoS* anteriormente citada, se hace uso de un mecanismo basado en establecer un umbral de paquetes máximo para cada protocolo analizado. Este umbral es configurado por parte del administrador de la red. Una vez que el *script* de monitorización detecta que se ha superado el umbral establecido, envía una notificación (*SNMP trap*) al servidor SNMP. Además, se inicia un periodo de espera, configurable igualmente por el administrador, en el que no se procesan más paquetes del protocolo en cuestión, por lo que el núcleo afectado vuelve a estar disponible. Siguiendo con el ejemplo anterior, el ataque *DHCP starvation* sería detectado y notificado por el *script* de monitorización y, de superarse el umbral de paquetes establecido para el protocolo DHCP, el análisis de dicho protocolo sería detenido un tiempo. Todo ello sin que la captura del resto de protocolos asociados a dicho núcleo, ni su consiguiente análisis, se viesen afectados.

VI. Experimentación

La experimentación llevada a cabo consta de cuatro escenarios de prueba. En primer lugar, se ha realizado una comparativa del uso de recursos de las dos técnicas de captura que pueden ser implementadas en el *sniffer* en función de diferentes tasas de tráfico de fondo en la red. En segundo lugar, para la técnica más eficiente, se realizan una serie de pruebas de carga para determinar su límite máximo admisible (R_{MAX}). En tercer lugar, se analiza cómo afectan a ambas técnicas de captura las optimizaciones del sistema, basadas en el uso de filtros BPF y el aprovechamiento de la

arquitectura multinúcleo. Por último, se compara el rendimiento la técnica más eficiente con *sniffers* habituales como *TCPDump*, *Wireshark* y *Tshark*. Todas las pruebas han constado de cinco repeticiones en las que se monitoriza el rendimiento de cada técnica durante 120 segundos, salvo las pruebas donde se evalúan las optimizaciones de las técnicas y donde se comparan con otros *sniffers*, que se han prolongado durante 200 segundos. El tráfico de fondo ha sido inyectado en la red mediante el software *Ostinato*.

Para llevar a cabo el abanico de pruebas anterior, se ha construido un entorno de experimentación simplificado tal y como se muestra en la Fig. 6. El objetivo del mismo es emular un diseño plano de red corporativa.

En el entorno de experimentación simplificado se pueden apreciar las siguientes entidades: una máquina en la que se despliega el sistema de detección, una máquina que actúa como servidor SNMP centralizado, una máquina atacante que inyectará tráfico malintencionado en la red, una serie de inyectores que emulan el tráfico de fondo generado por usuarios legítimos de la red y, por último, un *switch* y un *router* que representan el entorno conmutado, y la puerta de enlace junto con el servidor DHCP de la red corporativa, respectivamente. Las características técnicas de la computadora en la que se despliega el sistema de detección son: Intel Core™ i7-7700HQ @2.8 GHz de 8 núcleos y 16 GB de memoria RAM. Estas características tienen una influencia directa en la tasa máxima de procesamiento admitida por el sistema. Por este motivo se ha utilizado un computador con unas características asequibles, en términos de coste, para una empresa de cualquier envergadura.

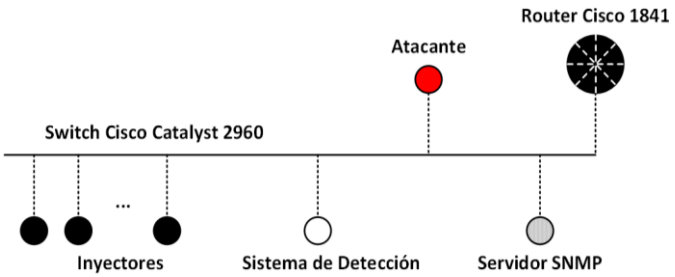


Fig. 6. Escenario de red utilizado durante la batería de pruebas.

Para la realización de las pruebas y la obtención de los resultados, el procedimiento seguido ha sido el siguiente. Con la red en un estado de equilibrio se lanza el sistema de detección usando la técnica de *sniffing* a evaluar. En los dos primeros escenarios se ejecuta el software *Ostinato* en un conjunto de inyectores para generar el volumen de tráfico necesario que será procesado por la técnica de *sniffing*. Los paquetes generados tienen una longitud fija de 64 bytes, el protocolo superior es TCP y no transportan carga útil. En los escenarios de prueba tercero y cuarto, donde se analizan las optimizaciones y se compara la técnica más eficiente respecto a otros *sniffers*, el equipo atacante envía masivamente tráfico CDP y DHCP generado desde la herramienta *Yersinia*, disponible en la distribución Kali Linux. Con independencia del tipo de escenario, el sistema de detección se instrumenta mediante la herramienta *atop* para medir con una frecuencia de un segundo el uso de recursos y, a partir del conjunto de muestras, determinar el promedio de uso de los mismos.

VII. Resultados

En esta sección se muestran los resultados obtenidos para los distintos escenarios planteados en la experimentación. En el primer escenario se determina la diferencia de rendimiento para las dos técnicas de captura de paquetes implementadas en el *sniffer* sin usar optimizaciones. Para ello, se analiza en el sistema de detección el consumo de recursos de procesamiento (CPU) y memoria por parte de cada técnica ante una misma tasa de bits recibida. Durante los distintos casos de prueba la tasa de bits que es inyectada en la red toma los siguientes valores: 150, 500, 1000, 1500 y 2000 kbps.

En la Fig.7 se ilustra la comparativa de rendimiento de CPU entre ambas técnicas. Cuando el *sniffer* implementa la técnica basada en *raw sockets* (R), representada con líneas punteadas, apenas se alcanza un consumo medio del 10% de CPU para las diferentes tasas de tráfico analizadas. Por su parte, si el *sniffer* implementa la técnica basada en *scapy* (S), se obtiene un consumo medio de CPU por encima del 40% en cualquiera de los casos analizados, llegando a un régimen cercano a la saturación a partir de 1500 kbps. En cuanto al uso de memoria, se obtiene un porcentaje de uso de 0.3% en el caso de *raw sockets* y un 0.8% en el caso *scapy*, por lo que la diferencia no resulta significativa entre ambas técnicas.

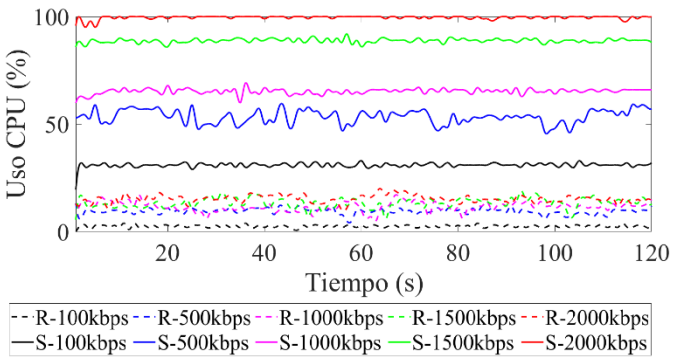


Fig. 7. Comparativa del rendimiento de las técnicas implementadas.

Por tanto, se puede afirmar que la técnica basada en el uso de *raw sockets* para la captura de paquetes en el *sniffer* aumenta significativamente el rendimiento del sistema de monitorización respecto al uso de la técnica basada en *scapy*.

Además, en la Fig. 7 se puede también observar que el *sniffer* basado en *scapy* satura la CPU de la máquina en la que corre el sistema para una tasa de tráfico inyectado de 2 Mbps, lo cual es un ancho de banda que queda muy por debajo del que puede ser utilizado en las redes corporativas de hoy en día. Por su parte, el rendimiento del *sniffer* basado en *raw sockets* en este primer caso de pruebas se mantiene lejos de saturar el sistema.

A raíz de los resultados anteriores, se plantea un segundo escenario de pruebas donde el objetivo es determinar la tasa máxima de tráfico inyectado que la técnica basada en *raw sockets* es capaz de soportar. Para ello, se realizan un

conjunto de pruebas de carga aumentado el tráfico inyectado en la red que el *sniffer* debe procesar.

En la Fig. 8 se puede apreciar que el límite máximo del enfoque basado en *raw sockets* es 70 Mbps. Al igual que con la técnica anterior, este ancho de banda se puede superar en las redes corporativas actuales dado el auge de servicios en tiempo real que consumen un gran ancho de banda como las actividades de *e-learning* o las videoconferencias.

Por tanto, ninguna de las dos técnicas es desplegable sin optimizaciones en un entorno de red corporativa, ya que el ancho de banda de las mismas supera la capacidad de procesamiento de las técnicas. Para mejorar su rendimiento se hace necesario implementar las optimizaciones planteadas en la sección V.

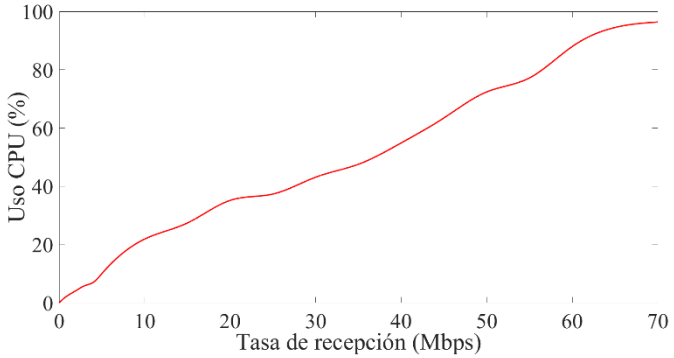


Fig. 8. Uso de CPU de la técnica basada en *raw sockets* en función de la tasa de bits capturada.

El siguiente escenario compara el rendimiento de ambas técnicas cuando se aplican sobre ellas los mecanismos de optimización. El objetivo es descubrir si el *script* de monitorización puede soportar un volumen de tráfico similar al generado en el ámbito de una red corporativa. En la Tabla IV se resumen los resultados obtenidos para ambas técnicas de captura utilizando la optimización del filtrado BPF. Cabe reseñar que, en el caso de no aplicar el filtrado BPF, se debe establecer qué porcentaje del tráfico total inyectado se corresponde con los protocolos a monitorizar, siendo el valor elegido en este escenario de pruebas un 5% del tráfico inyectado. Ese valor refleja una situación suficientemente realista ya que las normas de diseño de redes dictan que un 10% de tráfico *broadcast*, respecto al ancho de banda total de la red, es el máximo soportable por las redes corporativas más sensibles a este tipo de tráfico [19].

Como se ha comentado en la sección V, la principal consecuencia de aplicar las técnicas de filtrado BPF es la clasificación inmediata que hace el sistema entre paquetes que deben ser analizados y aquellos que deben ser descartados del análisis. Bajo esta optimización, el límite máximo soportado por ambas técnicas (2 Mbps para el caso de *scapy* y 70 Mbps para el caso de *raw sockets*) pasa a estar exclusivamente relacionado con tráfico perteneciente a los protocolos configurados para ser analizados.

TABLA IV
RESUMEN DE RESULTADOS

Técnica	Filtro BPF	R_{MAX} [Mbps]
scapy	☒	0,1
scapy	☑	2
raw sockets	☒	3,5
raw sockets	☑	70

Por tanto, bajo este supuesto, es posible asegurar que el despliegue del sistema utilizando ambas técnicas es viable en un entorno corporativo, dado que dichas tasas de bits son difícilmente alcanzables exclusivamente por los protocolos de capas 2 y 3 monitorizados por el sistema debido a sus reducidos tamaños de paquete, y el periodo de varios segundos entre notificaciones. No obstante, si el sistema fuese ampliado para detectar ataques de degradación de servicio (*DtoS*), como por ejemplo enviar tráfico pesado a direcciones MAC de difusión, solo sería viable el despliegue del sistema basado en *raw sockets* siempre que se trate de una red cuyo ancho de banda entre los equipos finales y el primer nivel de switches sea más o menos similar al límite soportado por la técnica. Por tanto, esta técnica podría usarse para detectar ataques *DtoS* en redes Fast Ethernet de hasta 100 Mbps.

Por otra parte, el uso de una arquitectura basada en múltiples núcleos permite efectuar un aislamiento entre protocolos. De este modo, se evitan ataques *DoS* contra el propio *sniffer* producidos por una inyección masiva de tráfico de alguno de los protocolos analizados, como por ejemplo la inundación de la red con tráfico DHCP en el ataque conocido como *DHCP starvation*. Sin embargo, la arquitectura multinúcleo implica más consumo de memoria. Su uso aumenta de manera lineal con el número de procesos utilizados para detectar paquetes, ya que es necesario replicar el código ejecutado para cada protocolo. Sin embargo, en el peor caso, se deben monitorizar un total de 15 protocolos, por lo que el máximo porcentaje de memoria utilizada es del 4.5% para la técnica basada en *raw sockets* y del 12% para la basada en *scapy*. A pesar del evidente aumento en el uso de memoria, los valores obtenidos siguen siendo moderados, permitiendo el despliegue de ambos enfoques. Por su parte, la inclusión de nuevos protocolos puede ser abordada sin que el uso de memoria suponga un obstáculo debido al amplio margen existente.

Continuando con el estudio, se plantean una serie de escenarios de prueba donde se analiza la resistencia del sistema al implementar técnicas de aislamiento y mitigación de ataques *DoS*. Para ello, se inducen sendos ataques de polución CDP y de *DHCP starvation*, y se compara el consumo de CPU respecto a la arquitectura mononúcleo sin dichas mejoras. Como se puede apreciar en la Fig. 9, el uso de CPU del sistema mononúcleo es del 40% durante un ataque de polución CDP. Al utilizar la arquitectura multinúcleo, junto con las técnicas de aislamiento y mitigación de ataques *DoS*, se producen varios picos en el uso de la CPU, si bien en ellos no se llega al 10% de uso en el núcleo afectado. El resto de los núcleos no se ven perjudicados.

Una situación límite se produce durante el ataque de *DHCP starvation*. En la Fig. 10 se observa cómo el uso de CPU del sistema mononúcleo se sitúa al máximo durante la fase de ataque. En cambio, al aplicar las técnicas

mencionadas sobre la arquitectura multinúcleo se producen picos puntuales de uso de CPU en el núcleo encargado de monitorizar el protocolo DHCP. Al utilizar técnicas de mitigación de ataques *DoS* los protocolos atacados dejan de ser analizados durante un tiempo configurable por el administrador de la red, reduciendo durante el mismo el consumo de recursos para evitar causar daño físico al sistema en el que se ejecute el *script* de monitorización (p. ej. derivados del sobrecalentamiento de componentes).

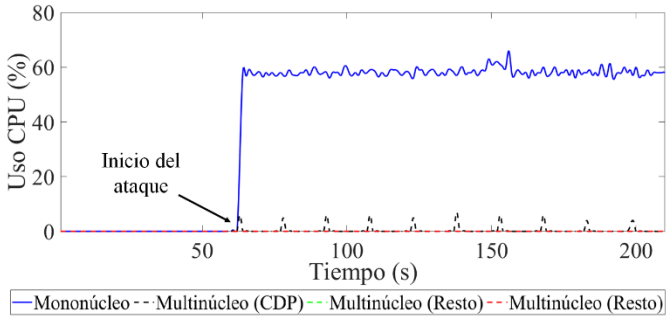


Fig. 9. Uso de CPU de ambas arquitecturas ante ataque de polución CDP.

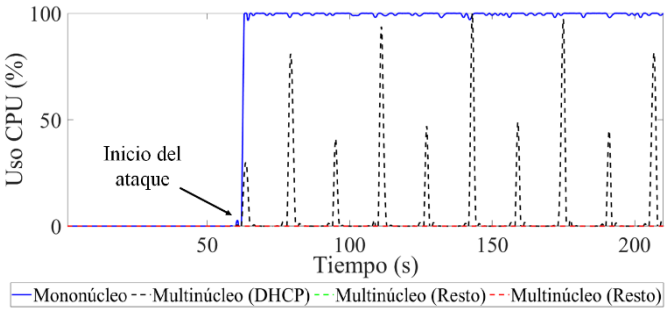


Fig. 10. Uso de CPU de ambas arquitecturas ante ataque de *DHCP starvation.*

Aunque ambas soluciones son válidas cuando la red se encuentra en un estado normal, bajo una situación de ataque *DoS* la única solución viable es la que aprovecha múltiples núcleos. En dicho escenario, a pesar de que la capacidad de detección del protocolo afectado se ve mermada, en el resto de los protocolos permanece inalterada. Se tiene así un aislamiento entre el protocolo afectado y el resto de los protocolos, aumentando la resiliencia frente a ataques *DoS*.

En la Fig.11 se compara el uso de CPU de la técnica basada en *raw sockets* (Raw S.) con otros *sniffers* de uso extendido durante un ataque de polución CDP. Se puede apreciar que el uso de CPU de la técnica basada en *raw sockets* con optimizaciones es superior al detectar el ataque, si bien es nulo durante el periodo de supresión posterior. En la Fig.12 se observa la misma tendencia bajo la realización de un ataque de *DHCP starvation*. En ambos casos el resto de *sniffers* proporcionan un uso de CPU estable, siendo en algunos casos idéntico, e incluso menor, que la técnica propuesta. No obstante, dichos *sniffers* carecen de mecanismos de generación de alertas en tiempo real, y requieren de escritura de datos en disco con el consiguiente consumo de espacio asociado.

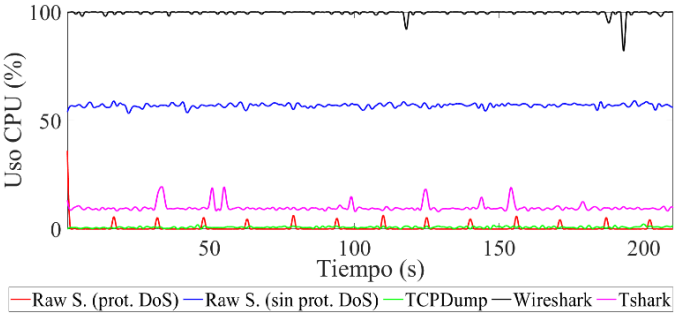


Fig. 11. Uso de CPU de varios *sniffers* ante ataque de polución CDP.

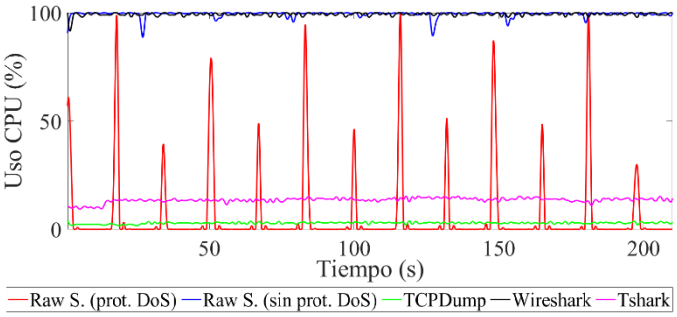


Fig. 12. Uso de CPU de varios *sniffers* ante ataque de *DHCP Starvation.*

Finalmente, en la Fig.13 se muestra un ejemplo de alertas generadas por el sistema y recibidas en el servidor SNMP. A través de ellas se notifica al administrador de la red la vulnerabilidad o ataque detectado, así como su origen.

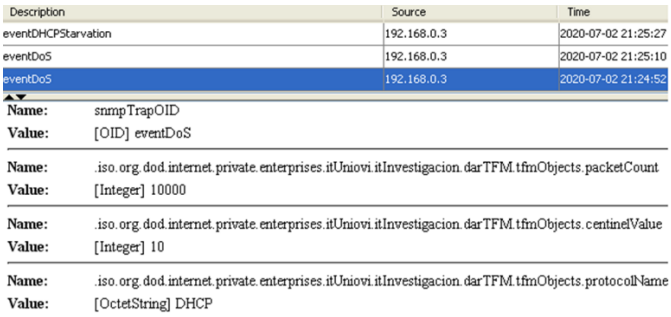


Fig. 13. Alerta recibida en el servidor SNMP centralizado del sistema.

VIII. Conclusiones

En este artículo se presenta un análisis de rendimiento de dos técnicas que permiten implementar un *sniffer* personalizado con el fin de monitorizar una red corporativa. Los dos enfoques evaluados son el uso de *raw sockets* y de la librería *scapy*, siendo este el primer artículo donde realiza dicha comparativa. Para ser estudiadas, ambas técnicas se implementan como núcleo de un sistema de detección de vulnerabilidades y ataques en las capas 2 y 3 del modelo OSI. Los resultados muestran que la técnica basada en *raw sockets* tiene un rendimiento mayor que la técnica basada en *scapy*.

No obstante, dadas las limitaciones que presentan ambas técnicas, se propone la aplicación de optimizaciones basadas en el filtrado de paquetes para mejorar el rendimiento, y el aprovechamiento de una arquitectura multinúcleo para la mitigación de ataques basados en inundación. Los resultados confirman que las optimizaciones provocan un rendimiento muy superior gracias al uso eficiente de recursos disponibles.

Los resultados demuestran que el sistema desarrollado es apto para ser desplegado en redes corporativas. Ambas técnicas son válidas cuando se usan conjuntamente con las optimizaciones planteadas. No obstante, solo la técnica basada en *raw sockets* es adecuada, aunque con limitaciones, si el sistema es configurado adicionalmente para detectar ataques de degradación de servicio.

Como líneas de trabajo futuro se plantea analizar el rendimiento de ambas técnicas con diferentes optimizaciones como son los filtros *extended BSD Packet Filtering* (eBPF) y *eXpress Data Path* (XDP), e igualmente estudiar la incidencia que pueden tener en la capacidad de procesamiento de las técnicas los distintos enfoques de despliegue en la red conmutada de los sistemas finales donde estén implementadas dichas técnicas.

AGRADECIMIENTOS

This research has been partially funded by the project RTI2018-094849-B-I00 of the Spanish National Plan for Research, Development and Innovation.

Financial support (SV-18-GIJON-1-03) given by the Asturian University Institute of Industrial Tecnhnology (IUTA) is acknowledged.

REFERENCIAS

[1] R. Anderson, C. Barton, R. Böhme, R. Clayton, M.J.G. van Eeten, M. Levi, T. Moore and S. Savage, "Measuring the cost of cybercrime", in *The Economics of Information Security and Privacy*, Berlin, Germany: Springer, Jan. 2013, ch. 12, pp 265-300.

[2] R. Khoussainov, A. Patel, "LAN security: problems and solutions for Ethernet networks", *Comput. Stand. Inter.*, vol. 22, no. 3, pp 191-202, Aug. 2000. DOI: 10.1016/S0920-5489(00)00047-7

[3] D. Zhou, Z. Yan, Y. Fu and Z. Yao, "A survey on network data collection", *J. Netw. Comput. Appl.*, vol. 116, pp 9-23, Aug. 2018. DOI: 10.1016/j.jnca.2018.05.004

[4] P. Smith, D. Hutchison, J. Sterbenz, M. Schöller, A. Fessi, M. Karaliopoulos, C. Lac ans B. Plattner, "Network resilience: a systematic approach", *IEEE Commun. Mag.*, vol 49, pp 88-97. DOI: 10.1109/MCOM.2011.5936160.

[5] I. Thiyeb, A. Saif and N. A. Al-Shaibany, "Ethical network surveillance using packet sniffing tools: A comparative study", *I. J. Netw. Comput. Inf. Secur.*, vol. 10, no. 5, pp 12-22, Jul. 2018. DOI: 10.5815/ijcnis.2018.07.02

[6] C. Anilkumar, P. Joseph, M. Viswanatham, A. Karrothu and B. Venkatesh, "Experimental and comparative analysis of packet sniffing tools", presented at the *2nd Int. Conf. on Data Engineering and Communication Technology* (ICDECT), Pune, India, Dec. 2017, pp. 597-605.

[7] A. Guezzaz, A. Ahmed, Y. Sadqi, A. Younes and Z. Tbatou, "A new hybrid network sniffer model based on Pcap language and sockets (Pcapsocks)", *I. J. Adv. Comput Sci Appl.*, vol. 7, no. 2, pp 207-214, Feb. 2016. DOI: 10.14569/IJACSA.2016.070228

[8] P. Asrodia and H. Patel, "Analysis of various packet sniffing tools for network monitoring and analysis", *I. J. Electr. Elec. Comp. Eng.*, vol. 1, no. 1, pp 55-58, May 2012. ISSN No. (Online): 2277-2626

[9] C. Gandhi, G. Suri, R. P. Golyan, P. Saxena, B. K. Saxena, "Packet sniffer - A comparative study", *I. J. Comput. Netw. Commun. Secur.*, vol. 2, no. 5, pp 179-187, May 2014. ISSN No. (Online): 2308-9830

[10] M. A. Qadeer, A. Iqbal, M. Zahid and M. R. Siddiqui, "Network traffic analysis and intrusion detection using packet sniffer", presented at the *2nd Int. Conf. Communication Software and Networks* (ICCSN '10), Washington, DC, USA, Feb. 2010, pp 313-317.

[11] L. J. G. Villalba, A. L. S. Orozco and J. M. Vidal, "Anomaly-based network intrusion detection system". *IEEE Lat. Am. T.*, vol. 13, no. 3, pp 850-855, Mar. 2015. DOI: 10.1109/TLA.2015.7069114

[12] J. M. Vidal, A. L. S. Orozco and L. J. G. Villalba, "Quantitative criteria for alert correlation of anomaly-based NIDS", *IEEE Lat. Am. T.*, vol. 13, no. 10, pp 3461-3466, Oct. 2015. DOI: 10.1109/TLA.2015.7387255

[13] F. Schneider and J. Wallerich, "Performance evaluation of packet capturing systems for high-speed networks", presented at the *1st ACM Conf. on Emerging Network Experiment and Technology* (CoNEXT '05), Toulouse, France, Oct. 2005, pp 284-285. DOI: 10.1145/1095921.1095982

[14] L. Braun, A. Didebulidze, N. Kammenhuber and G. Carle, "Comparing and improving current packet capturing solutions based on commodity hardware", presented at the *10th ACM SIGCOMM Conf. on Internet Measurement* (IMC 2010), Melbourne, Australia, Nov. 2010, pp 206-217. DOI: 10.1145/1879141.1879168

[15] N. Bonelli, F. Del Vigna, S. Giordano and G. Procissi, "Packet fan-out extension for the pcap library", *IEEE T. Netw Serv Man*, vol. 15, no. 3, pp 976-990, Sept. 2018. DOI: 10.1109/TNSM.2018.2828939

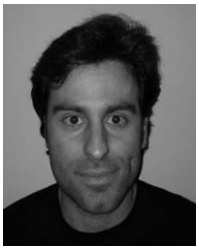
[16] X. Xu, Z. Li and Z. Li, "High-speed packet capture mechanism based on zero-bopy in linux", presented at *2nd Int. Conf. on Biomedical Engineering and Informatics* (BMEI 2009), Tianjin, China, Oct. 2009, pp 1-5.

[17] P. Roquero, E. Magaña, R. Leira, J. Aracil, "Performance evaluation of client-based traffic sniffing for very large populations", *Comput. Netw.*, vol. 166, pp 1-10, Jan 2020. DOI: 10.1016/j.comnet.2019.106985

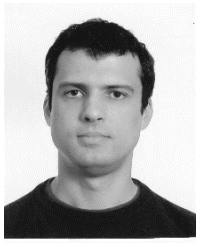
[18] S. McAnne and V. Jacobson, "The BSD packet filter: A new architecture for user-level packet capture", presented at the *USENIX Winter 1993 Conf.* (USENIX '93), San Diego, CA, USA, Jan. 1993.

[19] P. Oppenheimer. Top-down network design. Cisco Press. 2004.

David Álvarez Robles received his MS in Telecommunications Engineering from the University of Oviedo, Asturias, Spain in 2019. He is currently working at Alisec S.L, a Spanish IT company focused on cybersecurity. His current research interests include security and ethical hacking.



Pelayo Nuño Huergo received his MS and PhD in Computer Science from the University of Oviedo, Asturias, Spain, in 2009 and 2013, respectively. He is an assistant professor in the Department of Computer Science and Engineering at the University of Oviedo since 2015. His research interests include network security and multimedia networking.



Francisco González Bulnes received his MS and PhD in Computer Science from the University of Oviedo, Asturias, Spain, in 2007 and 2012, respectively. He is an associate professor in the Department of Computer Science and Engineering at the University of Oviedo since 2007. His current research interests include cybersecurity and real-time communications.

Juan Carlos Granda Candás received his MS and PhD in Computer Science from the University of Oviedo, Asturias, Spain, in 2004 and 2008, respectively. He is a tenured professor in the Department of Computer and Engineering at the University of Oviedo. He is involved research activities focused on network security, multimedia networking and IoT.