



## INTRODUCCIÓN

La pandemia y la posterior cuarentena que se vivió en el mundo en años pasados, forzó un periodo en las áreas de las tecnologías de la información como nunca antes visto. Se tenía que obligadamente forzar redes a un punto casi de la Saturación extrema. Redes que pese haber sido construidas con principios de escalabilidad hoy enfrentaban un tráfico y un escenario para el que nunca antes nos habíamos preparado.

Lo que de pronto supuso, que adicional a la expansión acelerada, era necesario contar con herramientas que ayuden a monitorear todo este tráfico que ahora se generaba, herramientas como Nagios, Observium, Zabbix, etc. El problema de estos programas ahora radicaba en que con el tráfico que se producía los servidores necesitaban más recursos de cómputo para poder monitorear todo el tráfico saliente.

Un estudio realizado por Ghaleb et al. (2020) utilizó Wireshark para monitorear el tráfico de red en una universidad durante la pandemia. Los resultados mostraron un aumento significativo en el tráfico de la red debido al uso de aplicaciones de videoconferencia y la educación a distancia. Los investigadores también identificaron

varias amenazas de seguridad, como intentos de phishing y ataques de denegación de servicio (DDoS).

Además, en un estudio más reciente, Iqbal et al. (2021) utilizaron Wireshark para analizar el tráfico de red en una universidad en Pakistán durante la pandemia. Los resultados mostraron un aumento significativo en el tráfico de la red debido al uso de aplicaciones de videoconferencia y la educación a distancia. Los investigadores también identificaron varios intentos de ataque de fuerza bruta y ataques DDoS.

Estos estudios destacan la importancia de Wireshark como herramienta de monitoreo de redes durante la pandemia y su capacidad para identificar amenazas de seguridad.

Por lo que el presente estudio tiene como objetivo evaluar el uso de expresiones regulares con el software mediante la recolección y clasificación de expresiones que, aplicadas a filtros y búsquedas, puedan servir como guía en un manual de usuario para la localización de problemas relacionadas con la red. Para ello se trabajará sobre los puertos y protocolos más usados en la actualidad para poder producir resultados que puedan cuantificar la mezcla del uso de Wireshark con las expresiones regulares.

## MARCO TEORICO

Wireshark es una herramienta de código abierto, lo que permite a cualquier usuario utilizarla sin problemas. Su potencia y eficiencia lo hacen un fuerte candidato para ser la herramienta por defecto en muchas organizaciones. En 2010, un grupo de estudiantes de la India evaluaron las capacidades de Wireshark para la detección de intrusiones en un sistema, y concluyeron que con el uso de los comandos de filtrado correctos y utilidades complementarias, Wireshark puede ser convertido en un software integral de detección de intrusiones (Banerjee, U., Vashishtha, A., & Saxena, M, 2010).

Unos años más tarde, un grupo de estudiantes de la universidad de Alabama y The Behrend College, motivados por el incremento de los ataques a redes en los últimos años, decidieron realizar un análisis forense de redes por medio de la herramienta. Concluyeron que los software de análisis de paquetes son vitales para el análisis forense de redes, y que los métodos actuales más comunes no eran suficientes ni efectivos para detectar los nuevos tipos de ataques (Ndatinya, V., Xiao, Z., Manepalli, V. R., Meng, K., & Xiao, Y, 2015).

Como muestran estos estudios, la potencia y fiabilidad de Wireshark es más que conocida. Por lo tanto, mucha gente ha sido capaz de llevar al software mucho más allá de sus capacidades y aplicaciones originales. Uno de estos fueron Dubey, S., & Tripathi, N. (2015) quienes utilizaron Wireshark para detectar comportamientos anómalos en tiempo real en una red WAN.

Otro pionero es Sikos, L. F. (2020) quien en su estudio combinó el concepto de la IA, Knowledge Representation, para realizar un análisis parcialmente automatizado de 'honeypots' basado en archivos de captura de paquetes del software en cuestión. Varghese, J. E., & Muniyal, B. (2021) también utilizaron Wireshark en su estudio piloto, con el fin de detectar ataques DDOS (Denegación distribuida de servicio).

Esto es una motivación importante para aportar más en la materia, un concepto que es poco conocido en el mundo del monitoreo de redes son las expresiones regulares. Las expresiones regulares, también denominadas regex, son las unidades de descripción de los lenguajes regulares, que se incluyen en los denominados lenguajes formales. Son un instrumento clave de la informática teórica, la cual, entre otras cosas, establece las bases para el

desarrollo y la ejecución de programas informáticos, así como para la construcción del compilador necesario para ello. Es por esto que las expresiones regulares (regex), basadas en reglas sintácticas claramente definidas, se utilizan principalmente en el ámbito del desarrollo de software. (Verma, P, 2015).

Existen pocas investigaciones que traten en profundidad las regex aplicadas con Wireshark. Una de ellas fue realizada en 2018 por Dinaki, P, quien hizo un estudio comparativo de las técnicas de Inspección Profunda de Paquetes: Expresiones regulares contra match exacto. Otra más reciente fue en 2020, por parte de Kaur, A, el cual por medio de Wireshark y Snort probó la protección de una red contra la herramienta de 'penetration testing' metasploit. En su trabajo sólo aparece la frase 'Expresiones regulares' 3 veces, lo cual es tendencia en los papers que usan ambos términos.

Encasillar las 'expresiones regulares' como un concepto residual o secundario, es común debido a esta falta de conocimiento, y en pro de motivar nuevas investigaciones en el ámbito es pertinente un estudio íntegramente dedicado a la utilización de las *regex* para análisis de tráfico sobre Wireshark.

### Bases Teóricas

Wireshark permite ver todo el tráfico capturado vía GUI (Interfaz gráfica de usuario) con el propio programa. Una característica fundamental de cualquier analizador de paquetes son los filtros, para que únicamente nos muestre lo que queremos que nos muestre, y ninguna información más que nos generaría un trabajo extra.

Wireshark es capaz de leer y escribir en diferentes formatos de captura, como en formato de archivo tcpdump (libpcap), pcap ng, y otras muchas extensiones. Además, es capaz de leer datos de diferentes tecnologías de redes como Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI y otros. Hoy en día tenemos muchos protocolos con datos cifrados, con la clave privada adecuada, Wireshark es capaz de descifrar el tráfico de diferentes protocolos como IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, y WPA/WPA2.

### Conceptos Relevantes

#### *Expresión Regular (Regex)*

Una expresión regular es una cadena de caracteres que es utilizada para describir o encontrar patrones dentro de otras cadenas de texto, en base al uso de delimitadores y ciertas reglas de

sintaxis. Las expresiones regulares aplicadas al campo de la computación es una herramienta poderosa que ayuda a incrementar la productividad. Guru, S. (2008).

### **Análisis De Paquetes**

La detección, rastreo o sniffing de paquetes es la práctica de obtener, recopilar y registrar algunos o todos los paquetes que pasan a través de una red de ordenadores, independientemente de cómo se enruten dichos paquetes. De esta manera, se puede almacenar cada paquete (o subconjunto definido de paquetes) para un análisis posterior. Los datos recopilados sirven para una amplia variedad de propósitos, como la supervisión del ancho de banda y el tráfico.

Un rastreador de paquetes, también llamado a veces analizador de paquetes, se compone de dos partes principales:

Primero, un adaptador de red que conecta el rastreador a la red existente.

Segundo, un software que proporciona una forma de registrar, ver o analizar los datos recopilados por el dispositivo.

### **El Modelo OSI**

El modelo OSI (Open Systems Interconnection) es una estructura

fundamental para la transferencia de datos a través de una red de computadoras. Según Lam and Wang (2019), este modelo divide el proceso de comunicación en siete capas, cada una con una función específica que contribuye a la transferencia de datos exitosa. En la capa física y de enlace de datos (primera capa), Wireshark actúa como un sniffer de red para capturar y analizar tramas, identificar direcciones MAC y comprobar si los datos están siendo transmitidos correctamente.

La segunda etapa del modelo OSI se centra en la transferencia de datos a nivel lógico, donde la capa de red se encarga de la transmisión de datos a través de redes múltiples y proporciona la funcionalidad de enrutamiento. Según Kanapickas et al. (2020), en esta capa, Wireshark puede utilizarse para analizar el tráfico de red y rastrear la ruta de los paquetes a través de la red. Además, Wireshark puede identificar posibles problemas de enrutamiento, como reenvíos innecesarios o tráfico congestionado.

La capa de presentación y aplicación (tercera y última etapa) se enfoca en la presentación de datos al usuario final. En esta capa, Wireshark puede analizar el contenido de los paquetes, como la codificación y decodificación de los datos, y proporcionar servicios de red

específicos a las aplicaciones. Según Klimavicius et al. (2021), Wireshark puede ser utilizado para analizar y solucionar problemas en aplicaciones específicas, como navegadores web y correo electrónico. Además, Wireshark puede identificar posibles problemas de seguridad en la capa de aplicación, como transacciones no seguras o información confidencial que se transmite en texto plano.

En conclusión, el modelo OSI proporciona una estructura fundamental para la transferencia de datos a través de una red de computadoras. Wireshark, por su parte, es una herramienta de análisis de red que puede ser utilizada en cada una de las capas del modelo OSI para detectar y solucionar problemas en la red. Desde la captura de tramas en la capa física hasta el análisis del contenido de los paquetes en la capa de aplicación, Wireshark es una herramienta valiosa para los administradores de red y los ingenieros de sistemas que buscan optimizar y mantener el rendimiento de la red.

### ***Análisis Forense Informático***

El análisis forense digital se corresponde con un conjunto de técnicas destinadas a extraer información valiosa de discos, sin

alterar el estado de los mismos. Esto permite buscar datos que son conocidos previamente, tratando de encontrar un patrón o comportamiento determinado, o descubrir información que se encontraba oculta.

### **Investigación de Requerimientos para el Análisis de Tráfico de Datos**

#### **Regex - Recopilación**

Las expresiones regulares comprenden patrones que pueden usarse para buscar formas variables de texto, este tipo de expresiones regulares permiten buscar todo texto que responda a un patrón determinado, por ello, esta investigación intenta comprobar la mayor eficacia del uso de este método ante el análisis de tráfico tradicional.

Dentro del Wireshark, aunque la casilla de verificación de expresiones regulares en el diálogo Buscar/reemplazar esté activada, aún podrá buscar frases exactas. En este caso, se tienen que eliminar los operadores de expresión regular de la frase (consulte Sintaxis). También podemos hacer uso de los filtros en Wireshark y usar REGEX para con estos filtros poder encontrar cadenas de datos o paquetes, los filtros más comunes son:

- Filtro de dirección MAC
- Filtro de dirección IP

- Filtro de puerto
- Filtro de protocolo:
- Filtro de expresión:
- Filtro de tiempo
- Filtro de longitud
- Filtro de resolución de nombres

### Metodología

Esta investigación tiene un enfoque cuantitativo, ya que se intenta demostrar con cifras y estadísticas la viabilidad y efectividad del uso de expresiones regulares para el análisis de tráfico con el interés de descubrir problemas en la red y sus orígenes, con respecto a otros métodos. El alcance sería descriptivo - experimental, pues definirán y describirán los objetos principales a estudiar, siendo estos el uso de la herramienta Wireshark con la combinación de expresiones regulares, para lograr a medir e identificar dentro de las capturas de tráfico resultados cuantificables que darán respuesta sobre la capacidad que poseen las expresiones regulares aplicadas al análisis de redes usando como instrumento principal al programa Wireshark. En este sentido, la técnica implementada en esta investigación será la observación, teniendo en cuenta que se busca entender la manera en cómo se comportan las regex en este contexto.

En este caso, se busca demostrar la utilidad posible de la habilidad de poder usar expresiones regulares para buscar detalles específicos con Wireshark dentro del tráfico de red y encontrar problemas en la misma, también sobre la capacidad de construcción de las expresiones regulares a partir de las funciones y operadores lógicos presentados en las Figuras 1, 2 y 3.

Para ello, se va realizar una minuciosa revisión de literatura para la recolección de datos sobre problemas físicos y lógicos, de carácter común en redes informáticas.

Una vez clasificados los problemas, se escogen aquellos solucionables mediante el uso y análisis del tráfico de la red. La ponderación se realizará sobre una escala numérica según la afección que cause sobre el tráfico de la misma. A partir del análisis de los requerimientos, se recopilan y clasifican expresiones regulares que cumplan la función de poder encontrar el problema motivo del requerimiento, describiendo el significado y uso en el manual de Usuario.

El diseño de la investigación consiste en 4 etapas: recolección, preparación, desarrollo y evaluación, ya que se quiere explicar cómo es el procedimiento de análisis, que busca

comprender todo el proceso en cuanto al desarrollo de los escenarios y el conocimiento que se puede adquirir mediante el estudio.

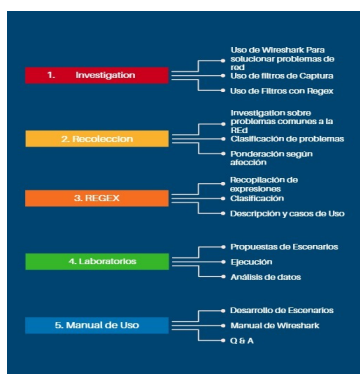


Figura 1. Fases de la investigación sobre el uso de las expresiones regulares con Wireshark

Es por ello que la primera fase se desarrollará mediante una recopilación de análisis y clasificación de datos de todos los requerimientos; la segunda fase consistirá en la búsqueda de expresiones regulares contra los requerimientos, la cual se dividirá en la recopilación y clasificación de las expresiones regulares disponibles y en la descripción del uso de las mismas; la tercera fase enmarcará el desarrollo de análisis de tráfico, donde se implementa una propuesta de escenario genérico que logre utilizar expresiones enfocadas en protocolos y puertos sin hacer uso de especificidades que puedan ser objeto de obsolescencia por arreglos sobre la capa de aplicación del Modelo OSI, escenario que posteriormente se

abordará en un laboratorio de pruebas, dónde los análisis sean analizados e interpretados, por último; la fase final será la creación de un manual de usuario con el fin de mejorar los resultados de análisis de una red de datos.

## Implementación

### Fase 1

En lo que atañe al proceso técnico de lograr encontrar un método que sin ser específico dentro del gran campo que supone el análisis de las capas que interactúan en una red, se ha tomado como consideración importante establecer un mecanismo que permita identificar problemas usando las expresiones regulares y Wireshark.

La identificación del problema es la primera y mas importante etapa en la solución de problemas de red. Para identificar el problema, se requiere un conocimiento solido de la topología de la red y los protocolos de red. Para lograr identificar cuellos de botella, problemas de conectividad y determinar la causa raíz del problema (Gibson, 2012).

### Fase 2

Para lograr este cometido se realizó una investigación exhaustiva para llegar a



determinar los problemas más comunes a las redes. Dando prioridad a los servicios y aplicaciones más usados.

## Determinación de Problemas Comunes

Los problemas más comunes dentro de la administración de una red fueron divididos en 5 Grupos:

**Problemas de conectividad.** Si hay problemas de conectividad en la red, Wireshark puede ayudar a identificar el problema. Por ejemplo, si los paquetes ICMP no están llegando a su destino, puede ser un indicio de que hay un problema de conectividad.

**Problemas de latencia.** Wireshark puede ayudar a identificar problemas de latencia en la red. Si hay un retraso en la transferencia de datos, Wireshark puede mostrar cuánto tiempo tarda un paquete en llegar a su destino.

**Problemas de velocidad.** Si la velocidad de la red no es la esperada, Wireshark puede ayudar a identificar la causa. Puede analizar el tráfico de la red para ver qué dispositivos o aplicaciones están consumiendo la mayor cantidad de ancho de banda.

**Problemas de seguridad.** Wireshark puede detectar tráfico sospechoso en la

red, como paquetes que intentan enviar contraseñas o información confidencial en texto sin formato.

## Problemas de configuración.

Wireshark puede ayudar a detectar problemas de configuración en la red, como dispositivos que no están configurados correctamente o que están configurados para utilizar un protocolo incompatible.

Toda vez identificado el problema se contempla la creación de un manual de uso de la herramienta Wireshark, que dote al usuario de las herramientas necesarias para la captación de paquetes, tomando los criterios de la investigación.

Listas formadas con expresiones regulares que puedan identificar los puertos y protocolos mas comunes que tienen incidencia en los problemas anteriormente mencionados

[illegible]

Figura 2. Filtros de puertos y atributos, según protocolo de red, y operadores disponibles

Toda vez los parámetros de uso para los condicionales lógicos y funciones de Wireshark, se construyen un conjunto de expresiones

EXPRESIONES
"http"
"dns"
"tcp"
"tcp.dstport == 80"
"tcp.srcport == 22"
"ip.src == 192.168.0.1"
"ip.dst == 10.0.0.1"
"frame.len > 1000"
"icmp"
"http contains password"
"arp"
"udp"
"tcp.srcport == 53"
"tcp.dstport == 443"
"ip.src == 10.0.0.1"
"ip.dst == 192.168.0.1"
"frame.len < 100"
"tcp"
"udp.dstport == x"
"http.request.method == 'GET'"
"ip.src == x.x.x.x"
"http.request.uri > 1000"
"tcp.flags.syn == 1 && tcp.flags.ack == 0"
"udp.dstport == 123 && udp.srcport == 123"
"dns.qry.name == x && dns.flags.response == 1"
"icmp.type == 8"
"http.request.method == 'GET' && http.request.version == '1.1' && http.request.uri == '/'"
"tcp.len > 1000"
"dns"
"arp.opcode == 2"
"tcp.flags.ack == 1"
"http.request.method == 'GET'"

Figura 3. Expresiones y funciones relativas a puertos y protocolos

EXPRESIONES ARP
"arp.opcode == 1"
"arp.opcode == 2"
"arp.opcode == 3"
"arp.opcode == 4"
"arp.src.proto_ipv4 == 0.0.0.0"
"arp.dst.proto_ipv4 == 0.0.0.0"
"arp.src.hw_mac == xxxxxxxxxxxx"
"arp.dst.hw_mac == xxxxxxxxxxxx"
"arp.src.proto_ipv4 != x.x.x.x"
"arp.dst.proto_ipv4 != x.x.x.x"
"arp.src.hw_mac == ff:ff:ff:ff:ff:ff"
"arp.src.proto_ipv4 == 255.255.255.255"
"arp.src.hw_mac == 01:00:5e:00:00:00/01:00:5e:7f:ff:ff"
"arp.src.proto_ipv4 == 10.0.0.0/8    arp.src.proto_ipv4 == 172.16.0.0/12    arp.src.proto_ipv4 == 192.168.0.0/16"
"arp.htype != 1"
"arp.ptype != 0x0800"
"arp.hlen != 6"
"arp.plen != 4"
"arp.request_responses_ratio < 0.1    arp.request_responses_ratio > 10"

Figura 4. Expresiones y funciones focalizadas en el protocolo ARP(Protocolo de resolución de direcciones)

EXPRESIONES DHCP
"bootp.option.dhcp == 2 and bootp.option.message-type == 2"
"bootp.option.dhcp == 2 and bootp.option.message-type == 2" -T fields -e bootp.xid -e ip.src   awk '{xid[\$1] = xid[\$1] " " \$2} END {for (i in xid) print i ": " xid[i]}'

Figura 5. Expresiones y funciones con indicadores de compromiso sobre el protocolo de configuración dinámica de Host (DHCP).

### Fase 3

Para probar las expresiones que se han recopilado se ha usado un entorno virtualizado con máquinas virtuales (VM) conectadas entre sí.

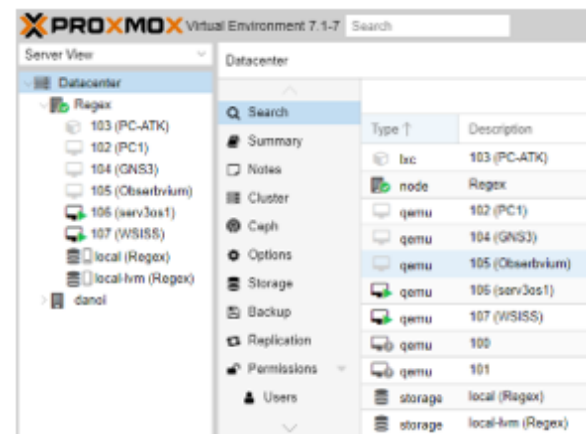


Figura 6. Hypervisor y maquinas virtuales

Para lo que se ha tomado en cuenta la siguiente preparación:

Descarga e instalación de Proxmox VE, en un servidor y creación de seis máquinas virtuales, una con Windows Server y IIS instalado y otra con Ubuntu Server y Apache instalado. Se destaca que ambas máquinas cuentan con 4 cores de un procesador i7 870 @ 2.93 con un Socket, 6 GiB de RAM y 60 GiB de almacenamiento. Las máquinas restantes generaran tráfico hacia las dos máquinas anteriores y es indiferente el sistema operativo.

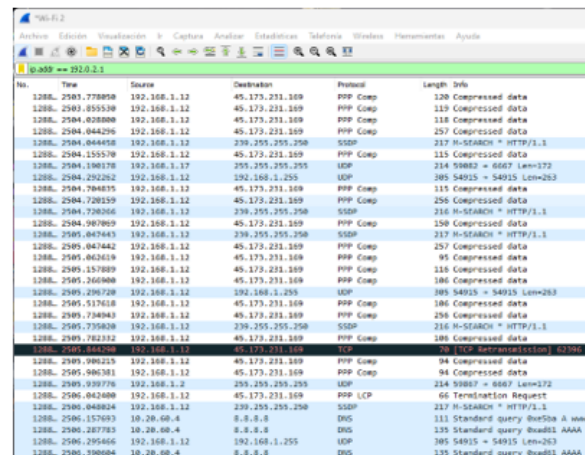
Descargar e instalar Wireshark en el computador que capturara el tráfico. De acuerdo a como se ilustra en el Manual

Iniciar la captura de tráfico de red y ejecutar pruebas de carga en cada servidor web, como solicitudes HTTP.

Analizar los datos capturados con Wireshark para medir el uso de red de cada servidor web. Se pueden observar estadísticas como el número de paquetes transmitidos, el tamaño de los paquetes, el tiempo de respuesta y la velocidad de transferencia.

Para hacer la comparación de los datos tomamos un tercer equipo para medir estadísticas en ambos archivos.

**Identificar los servidores web.** Una vez que se han capturado los paquetes, es necesario identificar los servidores web que se están utilizando. Para ello, se puede filtrar los paquetes por dirección IP de los servidores web.



No.	Time	Source	Destination	Protocol	Length	Info
1288.	2583.778808	192.168.1.12	45.173.231.169	PPP Comp	120	Compressed data
1288.	2583.855338	192.168.1.12	45.173.231.169	PPP Comp	119	Compressed data
1288.	2584.028808	192.168.1.12	45.173.231.169	PPP Comp	118	Compressed data
1288.	2584.044296	192.168.1.12	45.173.231.169	PPP Comp	257	Compressed data
1288.	2584.044458	192.168.1.12	239.255.255.250	UDP	217	H-SEARCH * HTTP/1.1
1288.	2584.155578	192.168.1.12	45.173.231.169	PPP Comp	115	Compressed data
1288.	2584.180170	192.168.1.12	239.255.255.250	UDP	214	98867 * 6667 Len=172
1288.	2584.292262	192.168.1.12	192.168.1.255	UDP	385	54915 * 54915 Len=263
1288.	2584.788815	192.168.1.12	45.173.231.169	PPP Comp	115	Compressed data
1288.	2584.728259	192.168.1.12	45.173.231.169	PPP Comp	256	Compressed data
1288.	2584.738366	192.168.1.12	239.255.255.250	UDP	216	H-SEARCH * HTTP/1.1
1288.	2584.987869	192.168.1.12	45.173.231.169	PPP Comp	150	Compressed data
1288.	2585.047543	192.168.1.12	239.255.255.250	UDP	217	H-SEARCH * HTTP/1.1
1288.	2585.047442	192.168.1.12	45.173.231.169	PPP Comp	257	Compressed data
1288.	2585.062618	192.168.1.12	45.173.231.169	PPP Comp	95	Compressed data
1288.	2585.157889	192.168.1.12	45.173.231.169	PPP Comp	115	Compressed data
1288.	2585.261008	192.168.1.12	45.173.231.169	PPP Comp	180	Compressed data
1288.	2585.296728	192.168.1.12	192.168.1.255	UDP	385	54915 * 54915 Len=263
1288.	2585.517618	192.168.1.12	45.173.231.169	PPP Comp	186	Compressed data
1288.	2585.738843	192.168.1.12	45.173.231.169	PPP Comp	256	Compressed data
1288.	2585.735828	192.168.1.12	239.255.255.250	UDP	216	H-SEARCH * HTTP/1.1
1288.	2585.782332	192.168.1.12	45.173.231.169	PPP Comp	180	Compressed data
1288.	2585.844768	192.168.1.12	45.173.231.169	TCP	50	TCP Reset(Sequence) 62586
1288.	2585.898235	192.168.1.12	45.173.231.169	PPP Comp	94	Compressed data
1288.	2585.986381	192.168.1.12	45.173.231.169	PPP Comp	94	Compressed data
1288.	2585.939776	192.168.1.12	239.255.255.250	UDP	214	98867 * 6667 Len=172
1288.	2586.042408	192.168.1.12	45.173.231.169	PPP Comp	66	Termination Request
1288.	2586.048824	192.168.1.12	239.255.255.250	UDP	217	H-SEARCH * HTTP/1.1
1288.	2586.157693	10.20.60.4	8.8.8.8	DNS	111	Standard query 0x50a A www
1288.	2586.287783	10.20.60.4	8.8.8.8	DNS	135	Standard query 0x401 AAAA
1288.	2586.295466	192.168.1.12	192.168.1.255	UDP	385	54915 * 54915 Len=263
1288.	2586.338864	10.20.60.4	8.8.8.8	DNS	135	Standard query 0x401 AAAA

Figura 10. Encontrando los servidores web

**Nota.** Se realiza el procedimiento utilizando el filtro "ip.addr == [dirección IP]"

**Analizar las estadísticas.** Wireshark ofrece una amplia variedad de estadísticas que se pueden utilizar para medir el uso de red de cada servidor web. Entre las estadísticas más útiles se encuentran:

**Número de paquetes transmitidos.** Esta estadística muestra la cantidad de paquetes que se han transmitido desde y hacia cada servidor web.

## Figura 11

### *Paquetes transmitidos*

*Nota.* Se aplica el filtro "ip.addr >= [dirección IP inicial] && ip.addr <= [dirección IP final]"

**Tamaño de los paquetes.** Esta estadística muestra el tamaño promedio de los paquetes que se han transmitido desde y hacia cada servidor web.

**Tiempo de respuesta.** Esta estadística muestra el tiempo que tarda cada servidor web en responder a las solicitudes que se le han enviado.

## Figura 12

### *Tamaño de paquetes*

*Nota.* Se utiliza el filtro `http and http.content_length > 1000`

**Velocidad de transferencia.** Esta estadística muestra la velocidad a la que se están transmitiendo los datos desde y hacia cada servidor web.

## Figura 13

### *Velocidad de transferencia*

*Nota.* Se utiliza `http.request.method == "GET"` and `http.response`

**Interpretación de los resultados.** Una vez que se han analizado las estadísticas, es necesario interpretar los resultados. En general, se puede concluir que un servidor web está utilizando más recursos de red si se envían y reciben más paquetes, si los paquetes tienen un tamaño promedio más grande, si el tiempo de respuesta es más largo o si la velocidad de transferencia es más baja.

Se toman varias muestras con estímulos diferentes que puedan ser medidos. Se generan escenarios que simulan varias situaciones de una red normal. Todo esto manipulando la red de manera que el tráfico que se genere simula una situación real.

Una vez obtenidas las capturas, tendremos en cuenta que la recopilación de datos de red para este análisis forense fue tomado de manera general, sin ningún tipo de filtro de captura preestablecido, método comúnmente llamado "atrápalo si puedes" donde todos los paquetes que pasan por un determinado punto de tráfico donde se capturan y se escriben en el almacenamiento y el análisis se

realiza posteriormente por lotes. De tal manera, se procede a hacer la comparación del análisis de tráfico tradicional versus utilizar las expresiones regulares anteriormente descritas en esta investigación.

Para los filtros de búsqueda tradicionales se tienen en cuenta algunos de los mencionados por la documentación de Wireshark.

Y se toma la lista de expresiones regulares de lo que nos interesa capturar, dependiendo de los escenarios

Como se observa, en la búsqueda con el filtro mencionado logramos mostrar una cantidad de paquetes reducida, Concretamente 258 paquetes equivalente al 3.7% de la captura total con la dirección ip objetivo a estudio.

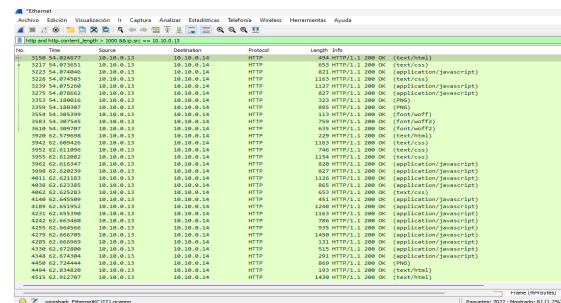
### Figura 14

*Gráfico de resultados con filtrado tradicional*

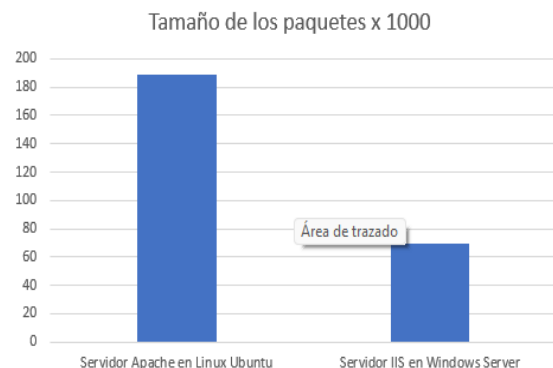
Sin embargo, se puede hacer una búsqueda más específica aplicando expresiones regulares a la fórmula para obtener mejores resultados o una muestra más reducida para facilitar el proceso de análisis forense en la red.

### Figura 15

*Aplicación básica de expresiones regulares "&&", Fuente: Captura de tráfico propia*



**Nota.** Se utiliza el filtro `http and http.content_length > 1000 && ip.src == [ip]`



En este caso se opta por aplicar la expresión regular “&&” que indica una condición de búsqueda donde ambos parámetros seleccionados, en este caso dos ip’s, coincidan. Por lo cual, el filtrado muestra los paquetes donde están los paquetes mayor de 1000 con la dirección de destino de uno de los servidores, se hace lo mismo para los dos, lo que nos proporciona la estadística antes mostrada.

Inclusive, otro ejemplo, donde se utiliza la expresión regular “||” que indica una condición de búsqueda donde cualquiera de los parámetros seleccionados se muestra.

**Figura 16**

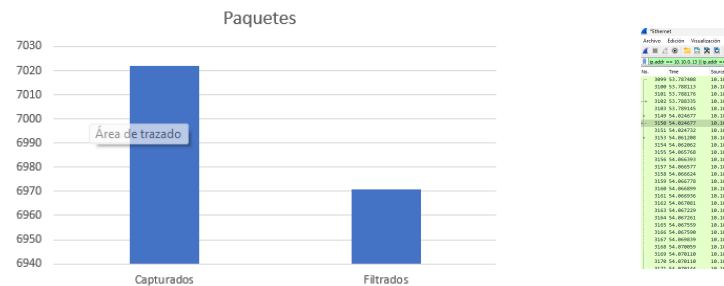
*Aplicación básica de expresiones regulares “||”*

Por esto, aunque el resultado sea una mayor cantidad de paquetes, el filtrado muestra los mismos donde una o ambas direcciones están presentes, con

un filtrado de 6971 paquetes equivalente al 99.3% de la captura total.

**Figura 17**

*Gráfico de resultados con expresiones regulares “||”*



Naturalmente, se demuestra que teniendo un filtro de búsqueda con ayuda de las regex se obtiene una muestra mucho más específica, que puede optimizar el trabajo de análisis.

Usando `ip.addr == [IP] || ip.addr == [ip] && http`, se tiene una búsqueda mas específica a lo que nos interesa,

No.	Time	Source	Destination	Protocol	Length	Info
484	30.979891	13.107.4.52	192.168.1.139	HTTP	1000	GET / HTTP/1.1
485	30.979891	13.107.4.52	192.168.1.139	TCP	60	65535 → 80 [RST] Seq=31102545 Win=0 Len=0
493	31.102545	13.107.4.52	192.168.1.139	TCP	60	65535 → 80 [RST] Seq=31102545 Win=0 Len=0
496	31.234332	192.168.1.1	192.168.1.139	SSDP	100	192.168.1.1:1900 → 192.168.1.139:1900 [RST] Seq=31268126 Win=0 Len=0
500	31.268126	192.168.1.104	192.168.1.139	ICMP	60	192.168.1.104 → 192.168.1.139: echo (ping) 0 bytes
504	31.715583	142.250.217.170	192.168.1.139	UDP	100	142.250.217.170 → 192.168.1.139: 54430 → 54430 [RST] Seq=32277347 Win=0 Len=0
508	32.277347	192.168.1.104	192.168.1.139	ICMP	60	192.168.1.104 → 192.168.1.139: echo (ping) 0 bytes
513	33.310909	192.168.1.104	192.168.1.139	ICMP	60	192.168.1.104 → 192.168.1.139: echo (ping) 0 bytes
519	34.276119	192.168.1.1	192.168.1.139	SSDP	100	192.168.1.1:1900 → 192.168.1.139:1900 [RST] Seq=3431349 Win=0 Len=0
522	34.331349	192.168.1.104	192.168.1.139	ICMP	60	192.168.1.104 → 192.168.1.139: echo (ping) 0 bytes
547	36.667695	52.226.139.180	192.168.1.139	TLSv1.2	100	52.226.139.180 → 192.168.1.139: 443 → 443 [RST] Seq=37275169 Win=0 Len=0
573	37.275169	192.168.1.1	192.168.1.139	SSDP	100	192.168.1.1:1900 → 192.168.1.139:1900 [RST] Seq=37949500 Win=0 Len=0
585	37.949500	192.168.1.1	192.168.1.139	DNS	100	192.168.1.1:53 → 192.168.1.139: 53 → 53 [RST] Seq=38002549 Win=0 Len=0
592	38.002549	192.168.1.1	192.168.1.139	TCP	60	592 → 80 [RST] Seq=38058226 Win=0 Len=0
593	38.058226	192.168.1.1	192.168.1.139	DNS	100	192.168.1.1:53 → 192.168.1.139: 53 → 53 [RST] Seq=38060897 Win=0 Len=0
595	38.060897	192.168.1.1	192.168.1.139	TCP	60	595 → 80 [RST] Seq=40672932 Win=0 Len=0
605	40.672932	192.168.1.1	192.168.1.139	SSDP	100	192.168.1.1:1900 → 192.168.1.139:1900 [RST] Seq=40685347 Win=0 Len=0
627	40.685347	52.226.139.180	192.168.1.139	TLSv1.2	100	52.226.139.180 → 192.168.1.139: 443 → 443 [RST] Seq=41713472 Win=0 Len=0
652	41.713472	142.250.217.170	192.168.1.139	UDP	100	142.250.217.170 → 192.168.1.139: 54430 → 54430 [RST] Seq=41834347 Win=0 Len=0
657	41.834347	192.168.1.1	192.168.1.139	ICMP	60	192.168.1.1 → 192.168.1.139: echo (ping) 0 bytes

obteniendo un total de 2062 de paquetes que corresponde al 29.4% de la captura total

**Figura 18**

*Grafico de resultados usando expresiones regulares “||”, “&&”*

### Figura 19

*Aplicación compuesta de expresiones regulares "=", "&&", "[|" y protocolo HTTP, Fuente:*

Así mismo, se puede demostrar la ventaja que agrega usar una cadena de regex más compleja. Donde se añade la expresión “==” al filtro anterior, que muestra los paquetes que coincidan con el parámetro indicado, y un parámetro extra siendo el protocolo de Hipertexto (HTML) de manera que muestra los estímulos de la red provocados para el estudio

Otro caso posible es la siguiente cadena de expresiones regulares.

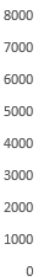
### Figura 20

*Aplicación compuesta de expresiones regulares "=", "||", "&&" y protocolo http, Fuente:*

Como se observa, al utilizar las regex, la muestra de paquetes para analizar se reduce a las que coinciden con las necesidades expresadas con las mismas, donde se obtiene un resultado de 241 paquetes equivalente al 3.4% de la captura total.

### Figura 21

*Gráfico de resultados con expresiones regulares “==”; “||”; “&&” y protocolo HTML*



Siendo así una disminución considerable de información que no es requerida en casos específicos como lo es el laboratorio de pruebas de este

[illegible]

estudio, de tal manera se comprueba que si existe una ventaja sustancial al



usar expresiones regulares en el programa Wireshark para el análisis de tráfico de una red.

### Conclusión

El uso de regex en Wireshark es beneficioso por varias razones. En primer lugar, permite a los usuarios buscar patrones complejos que no son posibles de encontrar con simples filtros de paquetes. Esto permite a los administradores de sistemas encontrar rápidamente paquetes relevantes y solucionar problemas de red de manera más eficiente.

En segundo lugar, el uso de regex en Wireshark es útil para reducir el tiempo necesario para analizar grandes cantidades de datos. Cuando se trata de redes grandes, es posible que se capturen una gran cantidad de paquetes en poco tiempo. La capacidad de filtrar paquetes utilizando regex permite a los usuarios reducir la cantidad de paquetes que se deben analizar y, por lo tanto, reducir el tiempo necesario para solucionar problemas de red.

Por último, el uso de regex en Wireshark es beneficioso porque es una técnica bien establecida y ampliamente utilizada para la búsqueda de patrones en grandes conjuntos de datos. Existen numerosas herramientas y recursos en

línea que pueden ayudar a los usuarios a aprender y utilizar regex de manera efectiva para el filtrado de paquetes y la búsqueda de patrones en Wireshark.

En conclusión, el uso de regex en Wireshark es una técnica poderosa que permite a los administradores de sistemas buscar patrones complejos en grandes conjuntos de datos de tráfico de red. Esto permite solucionar problemas de red de manera más eficiente y reducir el tiempo necesario para analizar grandes cantidades de datos. Por lo tanto, se recomienda el aprendizaje y uso de regex en Wireshark para mejorar la capacidad para solucionar problemas de red de manera efectiva.





### BIBLIOGRAFIA

Hauben, M. (2007). History of ARPANET *Site de l'Instituto Superior de Engenharia do Porto*, 17, 1-20.

Glowniak, J. (1998, April). History, structure, and function of the Internet. *Seminars in nuclear medicine* (Vol. 28. No. 2, pp. 135-144). WB Saunders.

McPherson, S. S. (2009). *Tim Berners-Lee: Inventor of the World Wide Web*. Twenty-First Century Books.

Spinellis, D. (2017). A repository of Unix history and evolution. *Empirical Software Engineering*, 22(3), 1372-1404.

King, A. (1994). *Inside Windows 95*. Microsoft Press.

Stanek, W. (2009). *Windows PowerShell 2.0 Administrator's Pocket Consultant*. Microsoft Press.

Renita, J., & Elizabeth, N. E. (2017, March). Network's server monitoring and analysis using Nagios. In 2017

*International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)* (pp. 1904-1909). IEEE.

Mishra, C. (2016). *Mastering Wireshark*. Packt Publishing Ltd.

Baxter, J. H. (2014). *Wireshark essentials*. Packt Publishing Ltd.

Banerjee, U., Vashishtha, A., & Saxena, M. (2010). Evaluation of the Capabilities of Wireshark as a tool for Intrusion Detection. *International Journal of computer applications*, 6(7), 1-5.

Ndatinya, V., Xiao, Z., Manepalli, V. R., Meng, K., & Xiao, Y. (2015). Network forensics analysis using Wireshark. *International Journal of Security and Networks*, 10(2), 91-106.

Iqbal, H., & Naaz, S. (2019). Wireshark as a tool for detection of various LAN attacks. *International Journal of Computer Science and Engineering*, 7(05), 833-837.

Dubey, S., & Tripathi, N. (2015). *Detection of Anomalous Behavior for Real Time Wide Area Network Traffic*. Using Wireshark. Sikos, L. F. (2020). *Knowledge representation to support partially automated honeypot analysis based on Wireshark packet capture files*.

*Intelligent Decision Technologies 2019* (pp. 345-351). Springer, Singapore.

Varghese, J. E., & Muniyal, B. (2021). *A Pilot Study in Software-Defined Networking Using Wireshark for Analyzing Network Parameters to Detect DDoS Attacks*. In *Information and Communication Technology for Competitive Strategies (ICTCS 2020)* (pp. 475-487). Springer, Singapore.

Verma, P. (2015). *Wireshark network security*. Packt Publishing Ltd.

Kaur, A. (2020). *Network security (confidentiality, integrity & availability) protection against Metasploit exploit using SNORT and Wireshark*.

Dinaki, P. (2018). *Deep Packet Inspection: A Comparison Study Between Exact Match and Regular Expression Techniques*.

Guru, S. (2008, June 5). *Expresiones Regulares. Conócelas y piérdelas el miedo*. SG Buzz. Retrieved September 6, 2022, from <https://sg.com.mx/content/view/545>

Wireshark <https://www.Wireshark.org/docs/>

Ubuntu Manuals  
<https://manpages.ubuntu.com/manpages/bionic/man4/Wireshark-filter.4.html>

Regular expression syntax *GRegex regular expression details* <https://developer-old.gnome.org/glib/stable/glib-regex-syntax.html>

Richard, S., Ed, W., & Ulf, L. (Version 4.1.0) *Wireshark User's Guide* <https://www.Wireshark.org/docs/wsug.html/>

Paessler. (2020, August 3). *IT Explained: Detección de Paquetes. ¿Qué es la detección de paquetes? Definición y detalles*. Retrieved September 6, 2022, from <https://www.paessler.com/es/it-explained/packet-sniffing>