Desarrollo e implementación de un protocolo Single Packet Authorization

Rubén Gutiérrez Guerrero

Resumen—En un paradigma cliente-servidor bajo una red TCP/IP no siempre puede ser una buena solución utilizar los protocolos clásicos en los que un servidor mantiene todos los puertos abiertos, ya que puede aportar información relevante a un atacante. En este artículo se presenta una propuesta que aporta una capa adicional de seguridad, evitando que un simple escaneo de puertos permita a un usuario malintencionado perfilar un servidor. Para esto, se propone implementar una variante de los protocolos clásicos de *portknocking* denominada *Single Packet Authorization* (SPA). Esta variante emplea un único paquete para auntenticar a un cliente, tras lo que se le permite la conexión a un determiando servicio. En el artículo se presenta adicionalmente una prueba de concepto que valida la viabilidad de la propuesta.

Palabras clave—Portknocking, Single Packet Authorization, HMAC.

Abstract—In a client-server paradigm under a TCP/IP network, it can not always be a good solution to use the classic protocols in which a server keeps all ports open, since it can provide relevant information to an attacker. This paper presents a proposal that provides an additional layer of security, preventing a simple port scan from allowing a malicious user to profile a server. For this, it is proposed to implement a variant of the classic protocols of *portknocking* called *Single Packet Authorization* (SPA). This variant uses a single paquet to authenticate a client, after which the connection to a certain service is allowed. The article also presents a proof of concept that validates the viability of the proposal.

Index Terms—Portknocking, Single Packet Authorization, HMAC

1 Introducció

En el paradigma cliente-servidor se establece, por definición, una comunicación entre dos partes a través de un canal, el cual permite el intercambio de mensajes. En este contexto, el *Servidor* es responsable de proveer de acceso a recursos o servicios, los cuales son consumidos por el *Cliente.* En el caso particular de los sistemas de información, el canal suele tratarse de una red de computadores, la cual es considerada como un entorno inseguro en tanto que su función es la de transportar los mensajes, y no la de proveer de mecanismos de seguridad para proteger la comunicación.

En el paradigma descrito anteriormente, y bajo redes basadas en la família de protocolos TCP/IP, el *Servidor* expondrá el acceso a recursos o servicios manteniendo abiertos un conjunto de puertos, a los que se conectará un *Cliente* para consumirlos.

Las características de los protocolos TCP/IP hacen que, bajo ciertas circunstancias, un atacante pueda recabar información del *Servidor*, tal como puertos abiertos o versiones del software que corre detrá de éstos. Este tipo de información, obtenida por un atacante en una primera fase de reconocimiento, puede ser explotada posteriormente de forma maliciosa para identificar vulnerabilidades y lanzar ataques que comprometan los sistemas.

En este artículo se presenta una solución que permite aplicar el concepto de defensa en profundidad. Para conseguir esto, la propuesta aquí recogida añade una capa adicional de seguridad, consistente en realizar una obertura de puertos selectiva para aquellos *Clientes* que se identifiquen como legítimos. De esta manera, los servicios no quedan expuestos libremente ante cualquier potencial atacante.

De forma más particular, la solución consiste en una variante del portknoking conocida como *Single Packet Authorization* (SPA) [2], la cual empleará un único mensaje que autentica al cliente mediante un *Message Authentication Code* (MAC) basado en HMAC. Esto reducirá el riesgo de que un atacante pueda suplantar a un cliente legítimo.

Adicionalmente, se presentará una prueba de concepto que se ha implementado, y que permite validar la viabilidad de protocolo SPA propuesto.

- *E-mail de contacto: rubengutierrezguerrero@gmail.com*

- *Mención realitzada: Tecnologies de la Informació.*
- *Treball tutorizado por: Sergio Castillo Pérez (Departament d'Enginyeria de la Informació i de les Comunicacions)*
- *Curso 2018/19*

1.1 Organización del artículo

El resto del artículo está organizado de la siguiente manera:

- La sección **"2 Objetivos"** incluye los diferentes objetivos a abordar en el artículo y a resolver en la prueba de concepto.
- La sección **"3 Estado del arte"** se analiza el estado del arte desde la visión de la servicios expuestos directamente en una red TCP/IP, en comparación a los mecanismos que permiten su ocultación como son el *portknocking* [7] y el SPA.
- La sección **"4 Metodología"** expone la metodología seguida para poder llevar a cabo la prueba de concepto, así como la distribución de tareas y fases a completar.
- La sección **"5 Desarrollo"** expone el desarrollo de dicha prueba, incluyendo diferentes puntos a tener en cuenta, tales como las librerías empleadas, la confección del mensaje SPA, el código implementado, o el funcionamiento de cada parte implicada.
- La sección **"6 Resultados"** muestra los resultados obtenidos, verificando la viabilidad del protocolo SPA propuesto a través de la prueba de concepto, y la ejecución del código asociado.
- La sección **"7 Conclusiones"** encapsula los objetivos, desarrollo y resultados, mostrando qué se puede extraer de ellos como conclusión, del mismo modo se habla de diferentes líneas de desarrollo futuras buscando expandir o mejorar la propuesta.

2 OBJETIVOS

En esta sección se recogen los diferentes objetivos del artículo, los cuales se centran en el estudio, el diseño, y la implementación de una solución al problema de visivilidad de los puertos y servicios de un servidor. Seguidamente se dan más detalles por cada uno de estos.

2.1 Estudio de mecanismos de protección de servicios expuestos en red

Este objetivo permite tener un conocimiento adecuado de la técnica del *portknocking*, las diferentes variantes y las estrategias de implantación, así como las particularidades a tener en cuenta (formato del mensaje, encapsulación del mismo, función hash a utilizar, ...). También incluye una vista a posibles utilidades necesarias en el momento de implementar al código (Por ejemplo, tener en cuenta que se reuqerirá utilizar una función HMAC sobre un mensaje o acceder a la hora del sistema).

Este objetivo es el pilar fudamental para poder llevar a cabo los recogidos en las subsecciones siguientes, siendo el punto de referencia para la definición del protocolo, la implementación, y guía a seguir [1, 4]. Y por tanto proyectar adecuadamente las fases de planificación y desarrollo según los datos y tareas necesarias para la impelmentación de dicho protocolo.

Este objetivo se verá materializado en la sección de "Estado del Arte", dónde se documentarán en términos generales los diferentes mecanismos analizados.

2.2 Diseño e implementación de una solución SPA

Este objetivo tiene una doble finalidad. En primer lugar, diseñar un protocolo SPA en base al estudio realizado en el objetivo anterior. El diseño de éste protocolo debe contemplar los aspectos estudiados así como las possibles librerías y lenguajes en el que desarrollar el código.

En segundo lugar, implementar la solución en sí. Para esto, se requerirá un entorno en red en que haya dos máquinas (un cliente y un servidor), el servidor deberá de poder abrir sus puertos o realizar acciones según el mensaje recibido del cliente, quien debe de poder enviar un mensaje al servidor. Del mismo modo, ambos, deben respetar el protocolo y el formato común del mensaje que se haya determinado en el objetivo anterio.

2.3 Ampliación de las funcionalidades SPA

Este objetivo nos permitirá automatizar acciones que podrá ejecutar un servidor al recibir un paquete del protocolo SPA, incrementando así sus funcionalidades. Se buscará que el servidor pueda realizar acciones, tales como: Envío de fichero, subida de fichero, obertura de puertos, envío de email… Funcionalidades que se determinarán durante el desarrollo de planificación de este objetivo.

Estas acciones deberían de ser realizadas según diferentes parámetros, tales como: puerto por el que se recibió el mensaje, contenido del mensaje, etc.

3 ESTADO DEL ARTE

En esta seccion se recoge el estado del arte en cuanto a diferentes modos de establecer una conexión entre un cliente y un servidor, y en relación al grado de exposición de los servicios. En concreto éstos son: 1) Comunicación clásica basada en el paradigma de TCP/IP, 2) La utilización de portknoking y 3) El uso del SPA. A continuación, se verá el funcionamiento de cada una de estas tres posibilidades.

3.1 Servicio expuesto a nivel de capa de transporte (TCP/IP)

Clasicamente, para la comunicación entre dos ordenadores conectados a una red se utiliza el paradigma conocido como TCP/IP. En este contexto, los protocolos de transporte más extendidos son los de TCP y UDP.

El primero de ellos está orientado a la conexión garantizando la entrega de los paquetes enviados. Para esto requiere de un establecimiento de conexión a través del denominado 3-way handshake.

Por contra, el protocolo UDP no está orientado a la conexión, lo que no garantiza la correcta recepción de los paquetes enviados. En la figura 1 se muestra un diagrama que resume graficamente las diferencias en el inicio de una comunicación.

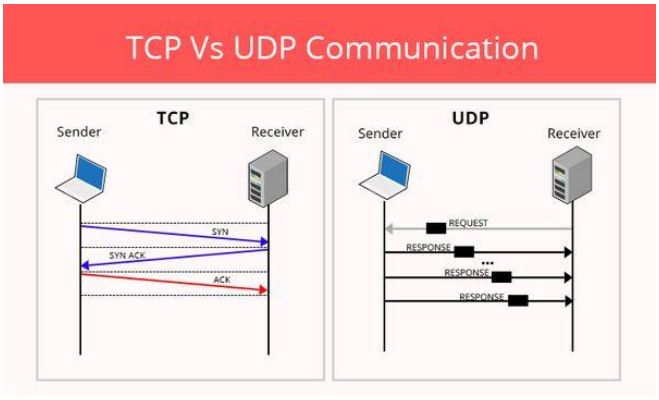

Figura 1. Comparativa entre la conexión TCP y la comunicación UDP.

Indiferentemente del protocolo TCP o UDP, los servicios siempre se presentan a través de un puerto al que accede un cliente. En cualquier caso ninguno de estos protocolos permite de forma nativa evitar que un atacante determine que puertos y/o servicios estan expuestos.

3.2 Portknoking clásico

El portknocking consiste en la utilización de una secuencia concreta de mensajes para la obertura del puerto solicitado. Esta secuencia sólo es conocida por el emisor y receptor por lo que es utilizada como medio de autentificación del emisor.

Este tipo de tecnología, por otro lado, permite proteger de manera "extra" la comunicación entre el ciente y el servidor, aplicando así el concepto de "defensa en profundidad" [14], que consiste en la ocultación de servicios por medio de protocolos o tecnologías que hacen de intermediario entre el cliente y el servicio del servidor a la que desea acceder. En la Figura 2 se puede observar de manera gráfica dicho concepto, dónde, para acceder al servidor es necesario pasar por portknoking, complementándose con el firewall y ssh, añadiendo ésta capa extra de seguridad. Cabe destacar que la variante SPA también permite aplicar dicho concepto.

Este conjunto de mensajes, enviados por un cliente hacia el servidor que lo autentificará, indican una secuencia exacta de mensajes, haciendo que sea posible, con diferentes secuencias, dar lugar a autentificaciones diferentes dependiendo del cliente que se conecte (Pudiendo así, incluso, configurar que, diferentes secuencias concretas de mensajes den lugar a acciones o servicios diferentes por parte del servidor).

Esto, como veremos más adelante, también será posible utilizando un sólo mensaje.

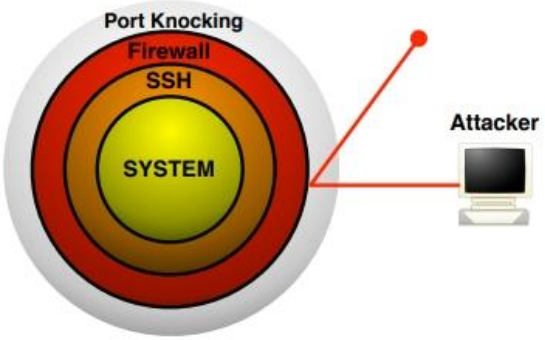

Figura 2. Concepto de defensa de profundidad aplicado al portknoking

Pero este sistema conlleva diferentes problemáticas, entre ellas:

- Un atacante podría monitorizar la secuencia y observar el comportamiento del servidor en consonancia.
- Limita la cantidad de información a transmitir.
- Un IDS podría interpretar los consecutivos paquetes del cliente como un escaneo de puertos.

Por esto razón existen diferentes implementaciones de portkocking, tratando de solventar los diferentes problemas que presenta.

3.3 Single Packet Authorizathion

Una variante particular del portknocking es la la denominada como *Single Packet Authorization* (SPA) [10]. Esta variante, en contraposición al portknocking clásico, tiene las siguientes características:

- Sólo se envía un mensaje, de modo que es menos llamativo y evita ser detectado por un posible atacante.
- El mensaje está autenticado y teniendo en cuenta aspectos como son la IP de origen o el timestamp que dificultan la reutilización del mismo por parte de un atacante.
- Un IDS tan solo vería un mensaje con datos indescifrables por lo que no se detectaría como un ataque, evitando así los obstáculos que acarrearía resolver este problema.

El funcionamiento del SPA puede resumirse en que un cliente, al tratar de conectar con el servidor, envía previamente un sólo mensaje que lo autentica de algún modo (como podría ser un HMAC). Este mensaje es leído por el servidor y, según el mensaje recibido, validará o no al cliente abriéndole posteriormente el correspondiente puerto de conexión.

Del mismo modo que ocurría con el portknocking, con un sólo mensaje es posible autentificar diferentes clientes o realizar diferentes acciones. Esto es, no se pierde flexibilidad a pesar de reducir el número de mensajes a uno, siendo esto un gran ventaja sobre el método "clásico" de portknoking.

También es importante señalar que SPA puede incluir en el mensaje un timestamp que permite reducir ataques de tipo replíca [12].

3.3.1 Hash-based Message Authentication Code:

La tecnología de SPA puede emplear diferentes estrategias para autenticar el cliente (por ejemplo *Message Authentication Codes*, firma digital, ...). Entre las diferentes alternativas, el uso de un MAC mediante funciones generadoras de tipo "Hash-based Message Authentication Code" (HMAC) suelen ser ideales en este contexto, al ser el tamaño del código resultante razonable para el propósito que nos ocupa.

El HMAC, especificado en el RFC 2104 [11], emplea una función hash sobre un conjunto de datos y una clave compartida. Esto requiere de un proceso previo al propio SPA para el intercambio seguro de dicha clave.

3.4 Resumen de diferentes tecnologías

En la Figura 3 se puede observar un resumen de las diferencias entre las diferentes tecnologías según cómo se establece la conexión, la comunicación, posibles problemas y el estado del servidor:

Tecnología	TCP/IP	Portknoking	SPA
Establecer conexión	Consiste en múltiples mensajes intercambiados entre el cliente y el servidor	Envío de diferentes mensajes, pero menos llamativo que una exposición directa	Mediante un solo mensaje.
Comunicación	Los mensajes, al ir en texto plano, no ocultan información sensible ni las intenciones del cliente con el servidor	Envío repetido de diferentes mensajes	El mismo mensaje que se envió para autentificar al cliente se ha utilizado para realizar peticiones.
Posibles problemas	Un atacante vería sin problemas tanto el contenido de los mensajes intercambiados como los diferentes servicios ofrecidos por el servidor	El envío repetido de mensajes al servidor podría ser confundido por un IDS por un escaneo de puertos	-
Visibilidad del Servidor	Requiere de una obertura de los puertos general y para cualquier cliente.	El servidor podría mantener servicios ocultos cerrando los puertos hasta que se solicite el servicio de dicho puerto	Servidor con los puertos cerrados, sin posibilidad de escaneo.

Figura 3. Resumen de diferencias entre TCP/IP, portknoking clásico y SPA

4 METODOLOGÍA

En este apartado se explone la metodología utilizada para alcanzar los objetivos indicados en la sección "Objetivos". Ésta se plantea como la ejecución de un conjunto de tareas de alto nivel, y que comprende las diferentes fases: *Planificación*, *Estudios de técnicas de portknocking*, *Diseño e implementación* y *Ampliación de funcionalidades SPA*.

A continuación, se expondrá la metodología que se ha seguido en las distintas fases que requerían de desarrollo, consistiendo ésta en el uso iterativo por cada nueva funcionalidad de **Planificación**, **Definición**, **Implementacfión** y **Testeo**; tales como en la fase de Diseño e Implementación y de Ampliación de funcionalidades SPA.

4.1 Metodología de desarrollo

La metodología seguida en las fases de desarrollo es la siguiente:

1. **Planificación**: Se revisa y utiliza como guía la información recopilada respecto a la tarea actual de modo que facilite y agilice la implementación y permita acotar y definir la "Definición de la tarea a realizar".
2. **Definición:** Una vez se halla recopilado toda la información necesaria se define exactamente qué tarea se realizará. Es decir "**Implementar la comunicación a través de HMAC entre el cliente y el servidor**" es algo muy general, en esta fase se busca encontrar exactamente qué funciones o módulos hay que hacer, qué entradas y salidas tendrá…
3. **Implementación:** Tras haber definido mejor en qué consiste la tarea se procede a la implementación de la misma, según lo acotado en la fase "**2. Definición**".
4. **Testeo:** Una vez se haya implementado, y verificado la funcionalidad de, la fase "**3. Implementación**" se procede a testearla y a comprobar resultados. En esta fase, adicionalmente, se documenta qué se ha obtenido de esta iteración. Esta fase está orientada al testeo de las funcionalidades recién implementadas.

5 DESARROLLO

La metodología planteada en este artículo en el momento de realizar la prueba de concepto, se ha orientado en

completar diferentes fases:

- Planificación: Fase centrada en la definición y ordenación de cada una de las tareas a realizar.
- Estudios de técnicas de portknoing: Estudio de las diferentes estrategias y soluciones portknoking y su implementación.
- Diseño e implementación: Desarrollo de una versión básica de la solución SPA.
- Ampliación de funcionalidades SPA: Adición de funcionalidades extra a la autentificación tras la recepción y aceptación de un mensaje.

También se detallará el funcionamiento y la lógica seguida por el servidor tras finalizar las fases de "Diseño e implementación" y de "Aplicación de funcionalidades SPA". Así como el funcionamiento del cliente que, a pesar de ser más simple, también debe cumplir con el protocolo establecido en la planificación de la prueba de concepto para elavorar un mensaje correcto.

Finalmente se verá la arquitectura, librerías y detalles utilizados por el cliente y servidor para hacer de soporte de la prueba de concepto.

5.1 Planificación

Se ha buscado documentación e información respecto a la tarea actual de modo que facilite y agilice la implementación y permita acotar y definir la "Definición de la tarea a realizar" en las fases posteriores.

Entre otros aspectos, se prioriza el diseño de los mensajes a enviar utiizando la tecnología de HMAC, así como la gestión de los paquetes enviados y recividos tanto en el servidor como en el cliente, llegando a concluir en el uso de la librería de scapy por parte del cliente y en el uso de netfilter [1] por parte del servidor.

5.2 Estudio de las técnicas de portnocking

En esta fase se profundiza en los diferentes aspectos de la técnica portkocking, recopilando la información necesaria para llevar a cabo la fase siguiente de "Diseño e implementación" Esta fase la podemos subdividir en dos sub-tareas:

- Estudio de diferentes opciones e implementaciones de la técnica de portkocking, obteniendo así un panorama de cómo y qué existe en este ámbito
- Recopilación de información sobre la técnica a utilizar, estudiando así, cómo debe implementarse y cuáles son las tareas que seguir en la elaboración de paquetes y comunicación entre cliente y servidor

5.3 Diseño e implementación

En esta fase se determina el diseño según la información recopilada en las fases anteriores y se procede a implementar este diseño. Esta fase la podemos subdividir en:

- Estudio del sistema de filtrado de paquetes de red en Linux, buscando conocer el funcionamiento del mismo y cómo debe de utilizarse para llevar a cabo los objetivos planteados en el artículo y la interacción con la implementación
- Fase de diseño de la solución. En esta fase se determina los diferentes aspectos de la implementación, definiendo los módulos necesarios para el cliente y el servidor, así como de qué funcionalidades tendrá cada uno y de sus interacciones (tanto internas como a través de la red)
- Fase de implementación. En esta fase se sigue la guía establecida en el apartado de "Planificación", de acuerdo con los contenidos estudiados en el primer y segundo apartado. Implementando así los módulos de cada una de las partes (cliente y servidor) y de la comunicación entre estos.
- Fase de testing. En esta fase se definen baterías de testeo para la implementación a modo de buscar y corregir los posibles errores cometidos. Del mismo modo se buscará verificar que cumple los objetivos definidos en la sección "Objetivos".

5.4 Ampliación de las funcionalidades del SPA

En esta fase se definen qué funcionalidades se implementan en la parte del servidor, así como la implementación de dichas funcionalidades

- Fase de diseño de funcionalidades. En esta fase se definen qué funcionalidades tendrá el servidor, así como cómo llevarlas a cabo.
- Fase de implementación. En esta fase se implementan las funcionalidades establecidas en la fase anterior, modificando en caso de ser necesario lo desarrollado en la fase de "Diseño e implementación".

5.5 Funcionamiento del Servidor

A continuación se describe el funcionamiento del

Servidor, el cual puede desglosarse en tres fases (ver figura 4).

Figura 4. Esquema de eventos en el Servidor

5.5.1 Recepción del paquete

La recepción del paquete por parte del código que se encarga de realizar las diferentes funcionalidades del servidor requieren del uso de, primeramente, de la creación de reglas que indiquen que determinados paquetes deben enviarse a la capa de aplicación y, a continuación, un código que permita la recepción de dicho paquete para leer y guardar el HMAC que contenga para su posterior procesado.

Para ello se ha utilizado las siguientes reglas, utilizando iptables [3]:

iptables -I INPUT -p tcp –dport 5001 -j NFQUEUE

De modo que el puerto que se utilizará para las siguientes pruebas de concepto será el 5001 TCP. Esta regla indicará que los paquetes que se dirijan hacia la cadena de INPUT (esta es la cadena que, concretamente, utilizamos durante la prueba de concepto) por el puerto 5001 tendrán como objetivo "NFQUEUE", tal y como se esquematiza de manera más clara en la Figura 5:

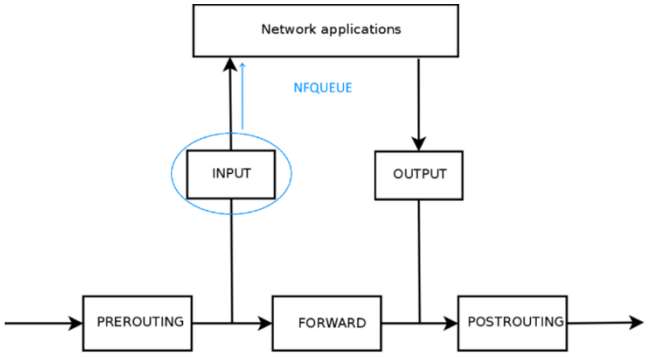

Figura 5. Vinculación de la cadena INPUT al aplicativo

Posteriormente, en la aplicación y mediante la librería de netfilter se vinvulará esta cadena al aplicativo, dónde se podrá acceder vía código.

Para realizar esta vinculación, los pasos que se han seguido mediante la librería de netfilter se resume en, primero, obtener el handler de la librería (nfq_open), a continuación se vincula el handler obtenido al tipo nfqueue (nfq_bind_pf), se liga la vinculación a una función callback, que será donde inicie el código y la lógica de la prueba de concepto (nfq_create_queue) y, finalmente, se indica cuanto del paquete se querrá obtener, que en este caso será el paquete completo, teniendo así a los datos del mensaje (nfq_set_mode, que se indicara el modo "NFQNL_COPY_PACKET"). Tras esta primera configuración el servidor escuchará en bucle los paquetes recividos (nfq_handle_packet) enviando los paquetes a la función que se le haya indicado en el callback.

Una vez la aplicación tenga acceso a dicha cola, se podrá leer el contenido del paquete que se facilitó mediante unos bytes de control (fffffff en este caso), se guardará el HMAC y se pasará a procesar la respuesta del Servidor.

5.5.2 Procesado del mensaje

Tras obtener el HMAC enviado por el cliente, se procederá a calcular los HMAC conocidos para la comparación con el HMAC que se ha recibido.

Para realizar este procesado por parte del servidor se ha optado por utilizar la librería de openssl [13], la cual ofrece una gran cantidad de funciones relacionadas con criptografía y hash, tales como el SHA-256 utilizado para generar el HMAC, función también disponible en dicha librería.

5.5.3 Decisión sobre el mensaje

Finalmente, el servidor procederá a decidir la aceptación del mensaje y la decisión a tomar en corcordanza. Pudiendo así realizar cualquier acción desde la activación de una regla que permita las conexiones a un determinado servicio del servidor hasta realizar acciones de gestión dentro del servidor. Adicionalmente se pueden tomar acciones preventivas ante mensajes extraños como, por ejemplo, que tras un determinado de intentos erroneos por parte de un cliente, se proceda a bloquear la IP de dicho cliente, mitigando así posibles ataques.

5.6 Funcionamiento del Cliente

Por el lado del cliente, se usa la librería de scapy [5], de Python, la cual permite la elavoración de paquetes y la edición de cada uno de los campos de las cabeceras de las diferentes capas (IP, TCP...).

Para un correcto funcionamiento del cliente, sólo se necesitan editar las cabeceras IP y TCP, así como la adición del payload, para a continuación enviar el paquete al servidor.

En la Figura 6 se puede observar de manera más clara la consecución de comandos utilizados para generar y

enviar el parquete con dicha librería:

```
payload = 'payload'
TCP(dport=port,sport=1024)
IP(dst=ipDst,src="192.168.1.111")
pkg = IP/TCP/payload
send(pkg)
```

Figura 6. Elavoración de paquete scapy simplificado

5.7 Estrucutra del mensaje

Tanto el cliente como el servidor deben de generar el mismo mensaje para que, en el momento de aplicar la función HMAC, se genere el mismo mensaje procesado, haciendo así posible que el servidor puede validar el mensaje del cliente.

Seguirá la siguiente estrucura definida en la fase de diseño:

IP_SERVIDOR + IP_CLIENTE + PUERTO_CLIENTE +PUERTO_SERVIDOR + PAYLOAD + TIMESTAMP

El 'timestamp' será el equivalente al día, hora y minutos, evitando así la, baja, probabilidad de que los segundos cambien entre el envío del cliente y la recepción del servidor. En realidad esta precaución podría llevarse incluso hasta el nivel de los segundos, ya que el tiempo típico de un datagrama es del orden de los milisegundos (un modo sencillo de comprobar esto es mediante el comando 'ping', del cual se puede observar el tiempo de respuesta en la Figura 7)

```
slave@slave-VirtualBox:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=121 time=16.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=121 time=14.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=121 time=14.3 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=121 time=15.2 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=121 time=15.3 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=121 time=15.2 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=121 time=18.0 ms
```

Figura 7. Tiempo de respuesta de un envío

Pero de todos modos se ha optado por tomar esta medida, a modo de asegurar el envío.

El mensaje generado tras la concatenación de los datos comentados, se procederá a ser procesado por una función HMAC, la cual utilizará el SHA-256, para su posterior comparación con el HMAC recibido del cliente.

5.6 Arquitectura utilizada

Para llevar a cabo cada una de las fases anteriores y para dar soporte a las fases de implementación, los recursos necesarios para llevar a cabo la prueba de concepto serán los siguientes:

Ordenador multicore con un mínimo de 8 GB de RAM para alojar en virtualbox:

- Máquina virtual "Linux Master" que realizará el rol de servidor.
  - 2 GB de RAM.
  - Tarjeta de red modo bridge
  - S.O. Distribución Linux (Ubuntu)
  - Librerías
    - Iptables (Configurar los puertos y enviar los paquetes a nivel de aplicación)
    - gcc (Compilar el código generado) ▪ tcpdump (a modo de debug en el envío de paquetes)
    - lnetfilter (Librería para filtrar los paquetes)
    - lnfnetlink (Librería para manejar los paquetes desde la capa de aplicación)
- Máquina virtual "Linux Slave" que realizará el rol de cliente.
  - 1GB de RAM
  - Tarjeta de red modo bridge
  - S.O. Distribución Linux (Ubuntu) o Librerías
    - gcc (Compilar el código generado)
    - scapy (Para generar los paquetes)
    - hashlib (Para utilizar el hash que corresponda)
    - datetime (Para obtener la fecha local)

6 RESULTADOS

Los resultados los podemos diferenciar en los resultados funcionales (los resultados de la prueba de concepto, viendo así el funcionamiento del servidor y del cliente) y el diseño y software obtenidos a nivel de arquitectura y código (un cliente y un servidor funcionales capaces de realizar las acciones expuestas en este artículo y susceptibles de ser extenbdidos con nuevas funcionalidades.

6.1 Resultados de las pruebas

En las siguientes figuras se puede observar cómo el cliente envía el datagrama con el payload correspondiente al servidor (Figura 8), así como el mensaje original: la concatenación especificada en la subsección de "Procesado del mensaje" de la sección de Desarrollo.

```
send? y/n: y
.
Sent 1 packets.
Sended clean: 192.168.1.108192.168.1.11150011024payload180922
Sended hmac: fffffff94c23e8a2bbffb4923e94dd4f79aba9f0a0d6b3c5542efb2073050426d5a957d
send? y/n:
```

Figura 8. Envío del datagrama desde el cliente

Y cómo el servidor recibe y elabora su propio prototipo del mensaje oculto, mostrando también su mensaje

original a modo de debug, pudiendo así comparar tanto el mensaje generado por el servidor como por el cliente, cercionando que el mensaje es correcto (Figura 9).

```
FEEDBACK:
[DEBUG] RECIEVED: 94c23e8a2bbffb4923e94dd4f79aba9f0a0d6b3c5542efb2073050426d5a957d
[DEBUG] HMAC_256: 94c23e8a2bbffb4923e94dd4f79aba9f0a0d6b3c5542efb2073050426d5a957d
[DEBUG] Clean: 192.168.1.108192.168.1.11150011024payload180922
[DEBUG] HMAC digest: 94c23e8a2bbffb4923e94dd4f79aba9f0a0d6b3c5542efb2073050426d5a957d

[DEBUG] HMAC ACCEPTED
[DEBUG] RULE CREATED: Port 22 opened for 192.168.1.2
```

Figura 9. Recepción del mensaje desde la parte del servidor

Como se puede ver en el código adjunto de "envio.py", prácticamente es una línea por cada una de estas acciones. Como se ha comentado a lo largo del artículo (Concretamente en la sección de "Desarrollo"), se utilizará la librería de netfilter [4, 6] para pasar a capa de aplicación los datagramas recibidos por el servidor.

Además, la adición de la regla de la obertura del puerto 22 se ha realizado mediante la ejecución de un comando Linux del sistema, a través del código C++. Esto posibilita y facilita la ejecución de otros comandos y ficheros de formato bash. Esta característica será la aprovechada para la automatización de tareas en el servidor.

Para realizar esta obertura del puerto, se utiliza la librería de ufw [8, 9] para facilitar dichas operaciones. Más concretamente, el comando ejecutado para dicha acción que se ha utilizado a modo de ejemplo es:

system("ufw allow from 192.168.1.111 to any port 22");

Esto ejecutará en el sistema el comando "ufw allow 22" Que habilitará el puerto 22, de modo que la máquina solicitante tenga el puerto abierto para poder conectarse.

Finalmente se puede observar en la Figura 10 el proceso completo por parte del servidor, los diferentes mensajes intercambiados y la addición de la regla:

```
master@master-VirtualBox:~$ sudo ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 5001                       ALLOW IN    Anywhere
[ 2] Anywhere                   ALLOW IN    192.168.1.2 22
[ 3] Anywhere                   ALLOW IN    192.168.0.161 22
[ 4] 192.168.0.161 22           ALLOW IN    Anywhere
[ 5] 22                         ALLOW IN    192.168.0.161
[ 6] 5001 (v6)                  ALLOW IN    Anywhere (v6)

master@master-VirtualBox:~$
[DEBUG] RECIEVED: 67a16245b31fa34eedd8692a1b2d731111949a5b5721c7dbc6601009a2bd
b3
[DEBUG] HMAC_256: 67a16245b31fa34eedd8692a1b2d731111949a5b5721c7dbc6601009a2bd
b3
[DEBUG] Clean: 192.168.0.160192.168.0.16150011024payload091126
[DEBUG] HMAC digest: 67a16245b31fa34eedd8692a1b2d731111949a5b5721c7dbc6601009a
df3b3

[DEBUG] HMAC ACCEPTED
[DEBUG] RULE CREATED: Port 22 opened for 192.168.0.161

Skipping adding existing rule

FEEDBACK:
[DEBUG] RECIEVED: 67a16245b31fa34eedd8692a1b2d731111949a5b5721c7dbc6601009a2bd
b3
[DEBUG] HMAC_256: 67a16245b31fa34eedd8692a1b2d731111949a5b5721c7dbc6601009a2bd
b3
[DEBUG] Clean: 192.168.0.160192.168.0.16150011024payload091126
[DEBUG] HMAC digest: 67a16245b31fa34eedd8692a1b2d731111949a5b5721c7dbc6601009a
```

Figura 10. Activación de la regla por HMAC válido

Y, en la Figura 11 se puede observar la conexión del cliente vía ssh:

```
slave@slave-VirtualBox:~$ sudo python envio.py 192.168.0.160 5001 payload
send? y/n: y
.
Sent 1 packets.
Sended clean: 192.168.0.160192.168.0.16150011024payload91108
Sended hmac: fffffffb6de4f90acd05ab2ba47040c6beea6bbfa4ae623d4e22ed2ac73c7b46fb0
e6cc
send? y/n: y
.
Sent 1 packets.
Sended clean: 192.168.0.160192.168.0.16150011024payload91108
Sended hmac: fffffffb6de4f90acd05ab2ba47040c6beea6bbfa4ae623d4e22ed2ac73c7b46fb0
e6cc
master@master-VirtualBox: ~
applicable law.

master@master-VirtualBox:~$ exit
logout
Connection to 192.168.0.160 closed.
slave@slave-VirtualBox:~$ ssh master@192.168.0.160
ssh: connect to host 192.168.0.160 port 22: Connection timed out
slave@slave-VirtualBox:~$ ssh master@192.168.0.160
master@192.168.0.160's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.15.0-42-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

155 packages can be updated.
0 updates are security updates.

New release '18.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***
Last login: Sat Feb  9 11:18:11 2019 from 192.168.0.161
master@master-VirtualBox:~$
```

Figura 11. Conexión ssh tras la activación de la regla

6.1 Características del prototipo

Tras la realización de la prueba de concepto, se ha generado el software de un cliente y un servidor con las características y funcionalidades necesarias para llevar a cabo una comunicación vía SPA. El servidor puede recibir, procesar y decidir en función del HMAC analizado, necesitando unicamente, como configuración extra (más allá de compilar e iniciar el código) las reglas de iptables especificadas en el apartado de **Recepción del paquete** de la sección de desarrollo.

Por otro lado se ha obtenido el cliente que, con los parámetros de "ip del servidor", "puerto del servidor"y "paylaod" realiza las acciones necesarias para generar el mensaje y procesarlo mediante un HMAC utilizando el SHA-256.

Adicionalmente, en ambos casos, es necesario especificar las direcciones IP en el código.

7 Conclusiones

Como se ha podido ver a lo largo del artículo, y particularmente en la sección de estado del arte, existen diversas técnicas que permiten añadir una capa adicional de seguridad a los servicios expuestos través de una red TCP/IP. Así, el uso de portknoking clásico, o el SPA conforman una de éstos mecanismos.

En este artículo hemos presentado una implementación de SPA basada en GNU/Linux, en la que la parte *Servidor* emplea el subsistema de filtrado de paquetes de netfilter. Esta implementación emplea HMAC como *Message Authentication Code*, haciendo uso de la función hash SHA-256, lo que permite autenticar los paquetes enviados por la parte cliente y evitar ataques de suplantación o de repetición. Las diferentes pruebas realizadas de la implementación demuestran la viabilidad de la propuesta teórica, materializada en forma de prueba de concepto.

Adicionalmente, se ha podido comprobar como SPA puede extender su funcionalidad clásica de autorización de un cliente para el acceso a un servicio ejecutando comandos particulares en función del paquete enviado. El uso de la versión SPA efectivamente ha proveído, y se ha comprobado, que puede realizar dichas acciones de autentificación, autorizacion de acceso a un servicio, o de ejecución de comandos, a pesar de utilizar un sólo mensaje. Esto, junto con el uso de un HMAC apropiado aplicado a los datos enviados junto a un timestamp robustece la solución desde una perspectiva de seguridad. Asimismo, se ha comparado la solución SPA con la de portknocking, exponiendo las bondades de esta tecnología.

Desde la perspectiva futuras lineas de mejora podemos considerar las siguientes:

- **Alternativas a HMAC:** Una possible via de continuación sería estudiar alternativas a HMAC, como podría ser, por ejemplo, la firma digital.
- **Clientes concurrentes**: Una de las líneas de desarrollo sería la gestión y conexión de clientes concurrentes, de modo que un servidor (o incluso un cluster) pudiese servir a una gran cantidad de clientes de manera concurrente.
- **Extensión de funcionalidades**: Otra de las principales líneas es la de una extensión de funcionalidades, automatización de tareas en el servidor como solicitud por parte de un cliente, de modo que se ofreciesen diferentes servicios y configuracioens dependiendo del payload, puerto, cliente...
- **Gestión de usuarios**: Una vez se dispusiese de una gran cantidad de usuarios y funcionalidades, esto daría lugar a una gran cantidad de datos (ip, puertos, payloads...) que gestionar, del mismo modo que al momento de comparar HMAC's del cliente con esta gran cantidad de información. Sería posible utilizar una base de datos o cálculo y comparación en paralelo para una mayor eficiencia en esta gestión.
- **Diseño de software**: Una última línea de desarrollo sería generar una mejor arquitectura, tanto por parte del cliente como del servidor de modo que se generasen UI funcionales o mejores diseños en lo que se refiere al código, mejorando su flexibilidad a la hora de aceptar futuros cambios o extensiones.

Agradecimientos

Antes de nada, agradecerle a mi tutor los esfuerzos y la gran ayuda con las fuentes de información, ha sido algo que me ha ayudado a abanzar rápido y sin problemas en ese aspecto. Además, por supuesto, de todos los consejos a la hora de dar formato y corregir los diferentes informes. Del mismo modo también agradecer a mis compañeros y amigos por escuchar mis "desvaríos" sobre el tema y el darme su opición al respecto.

Por último, mi familia me ha ayudado mucho en otros aspectos (apoyo, comida preparada, tareas que han hecho por mí) son pequeñas cosas ajenas al artículo en sí pero que sin esta ayuda hubiese sido completamente imposible llegar hasta aquí.

Bibliografía

[1] G. Gorka Gorrotxategi (2015/07/08) "Netfilter NFQueue" [Online]. Available: https://blog.irontec.com/netfilter-nfqueue-firewalling-en-user-space/

[2] L. M. García "Técnicas avanzadas de filtrado dinámico en sistemas cortafuegos: Port Knocking y Single Packet Authorizatio". Available: https://e-archivo.uc3m.es/bitstream/handle/10016/9584/PF_Luis_Martin_Garcia.pdf

[3] Synex Introduces (Visited: 2018/11/02) "How Does It Work: IP-Tables" [Online]. Available: https://n0where.net/how-does-it-work-iptables

[4] libnetfilter_queue (2017/11/13) "nfqnl_test.c" [Online]. Available: https://www.netfilter.org/projects/libnetfilter_queue/doxygen/nfqnl__test_8c_source.html

[5] J. Calles (2011/02/28) "Scapy: Construyendo un paquete UDP" [Online]. Available: http://www.flu-project.com/2011/02/scapy-construyendo-un-paquete-udp_2637.html

[6] libnetfilter_queue (2017/11/13) "Message parsing functions" [Online]. Available: https://www.netfilter.org/projects/libnetfilter_queue/doxygen/group__Parsing.html#gaf79628558c94630e25dbfcbde09f2933

[7] KZKG^Gaara (2013/04/04) "Port Knocking: La mejor seguridad que puedes tener en tu ordenador o servidor" [Online]. Available: https://blog.desdelinux.net/port-knocking-la-mejor-seguridad-que-puedes-tener-en-tu-ordenador-o-servidor-implementacion-configuracion/

[8] J. Wallen (UFW) (2015/10/30) "An Introduction to Uncomplicated Firewall" [Online]. Available: https://www.linux.com/learn/introduction-uncomplicated-firewall-ufw

[9] Oriol (Visited: 2019/01/27) "Como Habilitar y Configurar el Firewall UFW en Ubuntu" [Online]. Available: https://computernewage.com/2014/08/10/como-configurar-el-firewall-ufw-en-ubuntu/

[10] Lyz (2016/04/07) "Port-Knocking y Single Packet Authorization" [Online]. Available: https://elbinario.net/2016/04/07/port-knocking-y-single-packet-authorization/

[11] H. Krawczyk, M. Bellare and R. Canetti (1997/02) "HMAC: Keyed-Hashing for Message Authentication" [Online]. Available: https://tools.ietf.org/html/rfc2104

[12] OWASP (Visited 2019/01/26) "Testing for WS Replay" [Online]. Available: https://www.owasp.org/index.php/Testing_for_WS_Replay_(OWASP-WS-007)

[13] OpenSSL (Visited 2019/01/26) "HMAC" [Online]. Avaliable: https://www.openssl.org/docs/man1.1.0/man3/HMAC.html

[14] S. Jeanquier (2006/09/09) "An Analysis of Port Knocking and Single Packet Authorization" [Online]. Avaliable: https://www.securitygeneration.com/wp-content/uploads/2010/05/An-Analysis-of-Port-Knocking-and-Single-Packet-Authorization-Sebastien-Jeanquier.pdf