UAC Bypass & Research with UAC-A-Mola

Pablo González Pérez: pablo@11paths.com
Santiago Hernández Ramos: santiago.hernandezramos@telefonica.com

Resumen Ejecutivo

UAC-A-Mola es un framework diseñado para investigar, detectar, explotar y mitigar las debilidades denominadas bypass de UAC. Estas debilidades se encuentran en sistemas operativos Microsoft Windows. UAC-A-Mola permite automatizar la detección de un bypass UAC en una máquina Windows 7/8/8.1/10. UAC-A-Mola permite ejecutar diferentes módulos, personalizables, que permiten automatizar la investigación en busca de bypasses de UAC basados, principalmente, en fileless y DLL Hijacking. El framework permite incluir módulos orientados a la investigación y detección de otro tipo de bypasses. Además, UAC-A-Mola permite obtener una visión defensiva, para mitigar los posibles bypasses UAC operativos en el entorno Windows. UAC-A-Mola está escrita en Python y es un framework que permite extender funcionalidades a través de una interfaz sencilla y un sistema de creación de módulos.

1. Conceptos técnicos

El contexto de la investigación, diseño e implementación de una solución automatizada no se podría entender sin el entendimiento de algunos conceptos que rodean a los sistemas Windows. La solución de seguridad UAC, *User Account Control*, diseñada e implementada por Microsoft con la liberación y puesta en el mercado de los sistemas operativos Windows Vista y Windows Server 2008 marcó un hito para los usuarios de estos sistemas operativos.

En este apartado se detallan qué son y qué aportan ciertos conceptos que se deben entender antes de detallar algunas de las técnicas de ataque comunes para hacer un *bypass* de UAC. Además, es necesario para conocer los conceptos que envuelven a la herramienta diseñada e implementada que ayuda a la investigación, detección y a la automatización de este tipo de debilidades de los sistemas Windows.

1.1. UAC

El UAC (User Account Control) es un mecanismo de seguridad implementado por Microsoft en sistemas Windows con el objetivo de ayudar a los usuarios a controlar la autorización de los programas que van a ejecutarse o los cambios que se van a realizar en el equipo. A través de este método se pretende mitigar el impacto del malware en el sistema. [1]

Cada aplicación que requiera el token de administrador debe mostrar una ventana pidiendo el consentimiento. Solo hay una excepción, los procesos hijos heredan el token de usuario del proceso padre. Cuando un usuario estándar intenta ejecutar una aplicación que requiere el token de acceso de administración, UAC obliga al usuario a introducir unas credenciales de administración válidas.

Por defecto, los usuarios estándar y los administradores acceden a los recursos y ejecutan aplicaciones en el contexto de seguridad de un usuario estándar. Cuando un usuario se logea en un ordenador, el sistema crea un token de acceso para ese usuario. El token de acceso contiene información sobre el nivel de permisos de los que dispone el usuario.

Cuando un administrador se identifica se crean dos tokens de acceso distintos, un token de acceso estándar y un token de acceso como administrador. El token de acceso de usuario es utilizado para ejecutar que no realizan tareas administrativas. También es utilizado para mostrar el escritorio (explorer.exe). Explorer.exe es el proceso padre del cual todos los demás procesos iniciados por los usuarios heredan su token de acceso. Como resultado, todas las aplicaciones se ejecutan como usuario estándar a no ser que se proporcione consentimiento o credenciales para permitir a una aplicación utilizar el token de administrador.

Un usuario que es miembro del grupo de Administradores puede autenticarse en el sistema, y ejecutar aplicaciones utilizando el token de usuario estándar (sin privilegios

elevados). Cuando necesite realiza una acción que requiere el token de administrador, Windows automáticamente muestra una pantalla solicitando su aprobación.

Por defecto, cuando se instala un sistema operativo Windows, el primer usuario perteneces al grupo Administrador. Esto implica, que el UAC, mostrará una pantalla para solicitar la aprobación cada vez que se desee realizar una tarea que requiera el token de administración, pero no solicitará las claves de la cuenta de administrador.

1.2. Manifest

El manifest es un fichero XML que se encuentra incrustado en el interior de una aplicación. Este fichero especifica una serie de atributos sobre el binario al que acompaña. Uno de estos atributos importantes para este trabajo es "*autoElevate*", que indica la capacidad que tiene el binario para ejecutarse con integridad alta sin que el usuario tenga que realizar ninguna acción adicional, como, por ejemplo, aceptar un mensaje de UAC. Un ejemplo de manifest es el siguiente:

```
        processorArchitecture="amd64"
        name="Microsoft.Windows.WUSA"
        type="win32"/>

    <description>Windows Update Standalone Installer</description>
    <dependency>
       <dependentAssembly>
             <assemblyIdentity
                   type="win32"
                   name="Microsoft.Windows.Common-Controls"
                   version="6.0.0.0"
                   processorArchitecture="amd64"
                   publicKeyToken="6595b64144ccf1df"
                   language="*"/>
       </dependentAssembly>
    </dependency>
    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
        <security>
            <requestedPrivileges>
             <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
             </requestedPrivileges>
        </security>
    </trustInfo>
    <asmv3:application>
       <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
            <autoElevate>true</autoElevate>
            <dpiAware>true</dpiAware>
       </asmv3:windowsSettings>
    </asmv3:application>
</assembly>
NUL NUL NUL NUL NUL NUL NUL eETX 4 NUL NUL NUL V NUL S NUL _ NUL V NUL E NUL R NUL S NUL I NUL O NUL N NUL _ NUL I NUL N NUL F NUL O NUL N
NUL NUL NUL × : NUL NUL
```

Figura 1: Ejemplo de manifest.xml

1.3. Bypass de UAC

Microsoft no contempla un *bypass* de UAC, *User Account Control,* como una vulnerabilidad, debido a que depende de la configuración que la característica de seguridad tenga en cada momento.

El concepto de *bypass* UAC es la consecución por parte de un usuario, que pertenece al grupo de administradores, de lograr ejecutar un proceso o código con privilegio sin tener

que dar el consentimiento al UAC. En otras palabras, cuando se consigue ejecutar un proceso que el usuario quiere, sin que el UAC proporcione la aprobación.

Para que un *bypass* de UAC se pueda llevar a cabo, generalmente, se tienen que cumplir una serie de condiciones:

- El usuario que ejecuta el proceso que se utilizará para el *bypass* debe formar parte del grupo administradores.
- La política del UAC debe estar ejecutándose, generalmente, con su configuración por defecto.
- El binario que se utilizará para llevar a cabo el *bypass* debe estar firmado por *Microsoft* y tener la opción de *AutoElevate* a *true*. En este caso, ante la ejecución del UAC, con su configuración por defecto, el proceso se ejecutará y no se solicitará el consentimiento del UAC.

Puede existir casos en los que las condiciones cambien, pero generalmente, el usuario se encontrará con estos casos.

En el desarrollo de este trabajo se han utilizado técnicas basadas en la utilización de un proceso que está firmado por *Microsoft* y que en su *manifest* tiene el atributo *AutoElevate* a *true*. Si se consiguiera que ese binario se ejecutara y ejecutase otro proceso, este nuevo proceso lo haría con el mismo nivel de integridad o mismo nivel de privilegio que el proceso autoelevado.
Lógicamente, este tipo de técnicas se utilizan en entornos de *pentesting*, aunque también se han localizado en campañas de malware. El *pentester* puede estar en remoto a través de una *shell* o un *meterpreter* en el equipo, por lo que, si se cumplieran las condiciones anteriores, podría lograr ejecutar código como *System* gracias al *bypass* de UAC.

2. Técnicas

Cada vez son más las técnicas utilizadas para llevar a cabo un *bypass* de UAC. En este trabajo se muestran dos técnicas de las más utilizadas y potentes para llevar a cabo este proceso de evasión de UAC y consecución de ejecución con privilegio.

Para entender mejor el uso de la herramienta y las posibilidades para la investigación, detección, explotación y mitigación de las debilidades del UAC que ésta ofrece, se presentan ejemplos de técnicas de *bypass* de UAC. El proyecto UACMe [2] ofrece una gran cantidad de información sobre diferentes métodos y técnicas empleadas para aprovecharse de un *bypass* de UAC.

2.1. DLL Hijacking

La técnica de *DLL HIjacking* consiste en hacer un *"secuestro"* de la DLL con el objetivo de que se ejecute la que interesa a un atacante. De este modo, se logra la ejecución de código por parte de un proceso, sin ser dicho código el que esperaba el proceso. Este *"secuestro"* de DLL se puede llevar a cabo a través de la suplantación en disco de la DLL

original, ya sea porque la DLL es sobrescrita o porque no se encuentra en la ubicación esperada.

Esta técnica permite evitar la ejecución del UAC cuando el proceso que ejecutará la DLL es un binario firmado por *Microsoft* y que tiene el atributo *"autoElevate"* del *manifest* a *true*. Con una política por defecto de UAC en *Windows*, ese proceso se ejecutaría en un contexto elevado sin la solicitud de consentimiento del UAC. En este punto, el código cargado de la DLL por el proceso se ejecutaría con el mismo nivel de integridad, es decir, alto.

2.1.1. Bypass UAC: DLL Hijacking en CompMgmtLauncher.exe

Para ejemplificar la técnica de *DLL Hijacking* se utilizará un caso cercano a nosotros [3]. Este caso es funcional en *Windows 7, 8* y *8.1*. En primer lugar, si se visualiza el *manifest* del binario *CompMgmtLauncher.exe*, por ejemplo, con la herramienta *sigcheck.exe* se observa que el *"autoElevate"* está a *true*. Esto quiere decir que el binario, cuando se ejecute como administrador, se ejecutará con privilegio, en un contexto de integridad alto y sin solicitar el consentimiento de UAC.

```
    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
        <security>
            <requestedPrivileges>
                <requestedExecutionLevel
                    level="requireAdministrator"
                    uiAccess="false"
                />
            </requestedPrivileges>
        </security>
    </trustInfo>
    <asmv3:application>
        <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/Windo
wsSettings">
            <autoElevate>true</autoElevate>
        </asmv3:windowsSettings>
    </asmv3:application>
</assembly>
```

Figura 2: Visualización de manifest en CompMgmtLauncher.exe

Las herramientas como *ProcMon* permiten observar la integridad de ejecución de los procesos en el sistema. Se puede observar con *ProcMon* que, en la ejecución de *CompMgmtLauncher*, el proceso no encuentra un directorio denominado *\Windows\System32\compmgmtlauncher.exe.Local*. Si se analiza las siguientes consultas que realiza el proceso, se detecta que se están buscando una DLL sobre dicho directorio.

compmgmtlaun...	2520	CreateFile	C:\Windows\Prefetch\COMPMGMTLAUNCHER.EXE-D8C6028E.pf	NAME NOT FOUND
compmgmtlaun...	3416	CreateFile	C:\Windows\Prefetch\COMPMGMTLAUNCHER.EXE-D8C6028E.pf	NAME NOT FOUND
compmgmtlaun...	3416	CreateFile	C:\Windows\System32\en-US\compmgmtlauncher.exe.mui	NAME NOT FOUND
compmgmtlaun...	3416	CreateFile	C:\Windows\System32\en\compmgmtlauncher.exe.mui	NAME NOT FOUND
compmgmtlaun...	3416	CreateFile	C:\Windows\System32\compmgmtlauncher.exe.Local	NAME NOT FOUND
compmgmtlaun...	3416	CreateFile	C:\Windows\System32\compmgmtlauncher.exe.Local	NAME NOT FOUND
compmgmtlaun...	3416	CreateFile	C:\Windows\System32\compmgmtlauncher.exe.Local	NAME NOT FOUND
compmgmtlaun...	3416	CreateFile	C:\ProgramData\Microsoft\desktop.ini	NAME NOT FOUND
compmgmtlaun...	3416	CreateFile	C:\ProgramData\Microsoft\desktop.ini	NAME NOT FOUND

Figura 3: "Name Not Found" en compmgmtlauncher.exe.Local

El proceso, después de buscar el directorio compmgmtluancher.exe.Local, busca la DLL en un directorio denominado *WinSxS*, en el caso de las arquitecturas de 32 bits, y

descubre la DLL que se estaba buscando. Si un atacante puede escribir una DLL con código arbitrario en una ubicación protegida como es *\Windows\System32\[folder compmgmtlauncher.exe.Local]*, se podría lograr que el binario encontrase la DLL y ejecutase la DLL del atacante.

Cuando la DLL maliciosa fuera cargada por el proceso, se podría ejecutar código en un contexto elevado. El binario CompMgmtLauncher.exe, que se ejecuta en un contexto elevado, cargaría la DLL, por esta razón el código de la DLL se ejecutaría en un contexto elevado.

La generación de la DLL maliciosa puede llevarse a cabo de diferentes maneras, por ejemplo, utilizando *Metasploit Framework* o herramientas como *msfvenom*. Otra opción es crear manualmente la DLL, implementándola nosotros mismos.

Para llevar a cabo la copia de la DLL maliciosa generada a una ubicación protegida se puede utilizar dos formas:

- La aplicación *wusa.exe*. Esta aplicación permite extraer el contenido de un fichero CAB en una ubicación protegida, debido a que dicha aplicación se ejecuta en un contexto de integridad alto. Esta opción sólo nos servirá en Windows 7, 8 y 8.1.
- La utilización de *IFileOperation*. La exposición de los objetos COM. Este objeto contiene métodos que permiten copiar, mover, eliminar objetos del sistema de archivos como ficheros y carpetas. Este método es válido en Windows 10.

Hay que indicar que el binario CompMgmtLauncher.exe puede ser utilizado en Windows 10 para hacer el UAC Bypass, aunque solo se pueda utilizar el método *IFileOperation* como medio para copiar la DLL a la ubicación protegida. En el Github del usuario *pablogonzalezpe* [4] se puede encontrar un script que automatiza el proceso de copia de la DLL a la ubicación protegida a través de la aplicación *wusa.exe*.

Para que *wusa.exe* pueda desempaquetar un fichero CAB en una ubicación protegida, en este caso, sobre *\Windows\System32\compmgmtlauncher.exe.Local\[folder]*, se deberá generar el fichero CAB.

```
#Create DDF File
$texto = ".OPTION EXPLICIT

 .Set CabinetNameTemplate=mycab.CAB
 .Set DiskDirectoryTemplate=.

 .Set Cabinet=on
 .Set Compress=on

 .Set DestinationDir=compMgmtLauncher.exe.Local\x86_microsoft.windows.common-controls_6595
b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f2b2
""compMgmtLauncher.exe.Local\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7
601.17514_none_41e6975e2bd6f2b2\comctl32.dll""""
```

Figura 4: Fichero DDF para la generación del fichero CAB

Este fichero DDF está compuesto por directivas que serán procesadas por el binario makecab.exe, el cuál es el encargado de generar el fichero CAB. En otras palabras, podemos decir que DDF indica a makecab.exe cómo se debe distribuir los archivos en el

CAB y qué características tiene el fichero y la compresión de éste. Una vez hecho esto, se tendrá ya el fichero CAB disponible para que *wusa.exe* pueda utilizarlo.

La siguiente operación es la de utilizar *wusa.exe* para llevar a cabo la extracción en *Windows**System32**compmgmtlauncher.exe.Local**[folder]*. Ahora que ya se tiene la jerarquía de carpetas que compone *comctl32.dll* en *Windows**System32* se debe arrancar el proceso *CompMgmtLauncher.exe* para que se lleve a cabo la ejecución de código, es decir, la DLL, en un entorno privilegiado.

2.2. Fileless

La técnica *Fileless* ha marcado un hito en las técnicas de *bypass*. En cualquier técnica utilizada para llevar a cabo un *bypass* de UAC se necesita subir una DLL o un binario a la máquina comprometida. Con la aparición de las técnicas *Fileless* no se necesita subir ningún archivo al disco duro, ya que está basado en debilidades en el registro de *Windows*.

La idea es detectar binarios firmados por *Microsoft* y que tienen el atributo "*autoElevate"* a *true* de su *manifest* que interactúan con el registro de *Windows*. Dentro de esta interacción, es interesante detectar aquellos binarios que no encuentran claves en la rama *HKCU*. Esto puede desembocar en que un proceso ejecutándose en un contexto de integridad alto ejecute algo que se encuentra en una rama HKCU, provocando u obteniendo la ejecución de código referenciado a través de la rama HKCU. Esto quiere decir, que el código o binario, el cual no tiene por qué estar en local, referenciado en la rama HKCU se ejecutará en un contexto de integridad alto, es decir, en un contexto privilegiado.

A día de hoy, existen 3 *Fileless* distintos basados todos en el registro de *Windows*:

- *Fileless* 1 [5]. Esta primera técnica descubierta por *Enigma0x3* [6] y está basada en cómo el binario eventvwr.exe interactúa con el registro de *Windows*. La técnica se hizo pública en agosto de 2016.
- *Fileless* 2 [7]. Esta segunda técnica fue descubierta por *Enigma0x3* y está basada en cómo el binario sdctl.exe interactúa con el registro de *Windows* en busca de la clave *HKCU:**Software**Microsoft**Windows**CurrentVersion**App Paths**control.exe*. Esta clave proporciona un *"Name Not Found"*. La clave puede referenciar a un binario, el cual al ejecutar *sdctl.exe* se ejecutará con privilegio. La técnica se hizo pública en abril de 2017.
- *Fileless* 3 [8]. Esta tercera técnica está basada en cómo el binario *fodhelper.exe* interactúa con el registro de *Windows*. La clave que provoca la debilidad y la posibilidad de ejecutar código en un contexto elevado al ser invocada desde un proceso ejecutado como administrador es *HKCU:**Software**Classes**ms-settings**Shell**Open**command*. La técnica se hizo pública en mayo de 2017.

En el siguiente apartado se detallará el primer caso de *Fileless*. Hay que dotar de importancia y relevancia a esta técnica que ha sido utilizada en campañas de malware [9], para lograr que el malware ejecutase con el máximo privilegio en el sistema.

2.2.1. *Fileless 1 y el binario Eventvwr.exe*

Cuando el proceso *eventvwr.exe* se ejecuta realiza algunas consltas al registro de *Windows*. En particular, contra la sección *HKEY_CURRENT_USER* o *HKCU*. El binario *eventvwr.exe* genera dichas consultas ejecutándose en un contexto de integridad alta, es decir, se está ejecutando con privilegio. El binario *eventvwr.exe* se está ejecutando de forma autoelevada, ya que en su *manifest* tiene el atributo *"autoElevate"* a *true*, por lo que si se monitorizan claves del registro de *Windows* en la rama *HKCU* a través de un proceso elevado existe riesgo.

Si se analiza con la herramienta *ProcMon* las consultas que el binario *eventvwr.exe* realiza contra el registro de *Windows*, se puede encontrar que se realiza una consulta a la clave *HKCU\Software\Classes\mscfile\shell\open\command*. La clave anterior no se encuentra, por lo que después se realiza consultas contra la clave *HKCR\mscfile\shell\open\command* dónde se encuentra lo que el binario busca. El valor de la clave es el binario *mmc.exe*, por lo que se entiende que el binario *eventvwr.exe* está buscando al binario *mmc.exe* para ejecutarse en la consola de gestión.

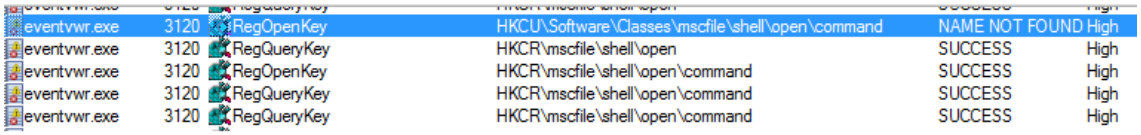

Figura 5: Clave no encontrada en el registro de Windows en la ejecución de eventvwr.exe

Como el binario *eventvwr.exe* está autoelevado, así se puede visualizar en su *manifest*, y está firmado por *Microsoft*, el sistema operativo lo autoeleva sin necesidad de lanzar el UAC.

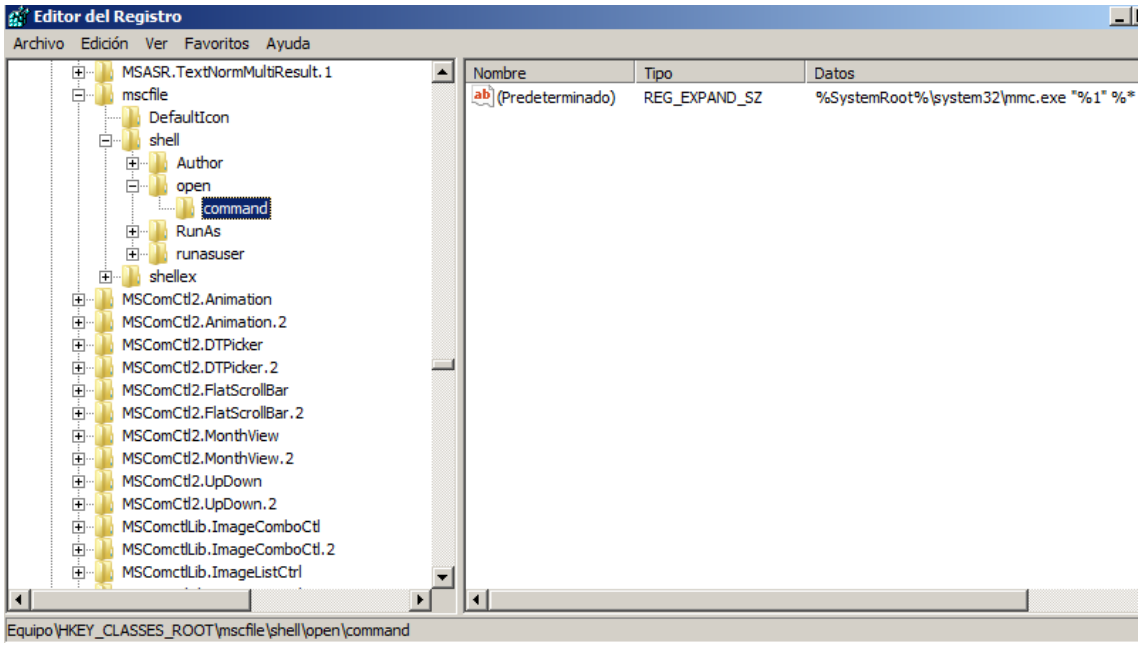

Figura 6: Clave del registro command

Aunque la consulta a *HKCU* devuelve un error, se realiza antes que a la clave alojada en *HKCR*, por lo que hay una posibilidad de poder manipular o interactuar con la rama *HKCU* con el objetivo de crear dicha clave. Creando la clave *HKCU**Software**Classes**mscfile**shell**open**command* y configurándole un valor, por ejemplo, de *powershell.exe*. En este instante, cuando se ejecuta el binario *eventvwr.exe* se estaría consultando la rama dónde antes no se encontrada un valor, y se estaría ejecutando el binario *powershell.exe*. Este binario se ejecutaría con el mismo privilegio que lo hace el *eventvwr.exe*, por lo que se habría logrado el *bypass* de UAC.

Lógicamente, realizar un *bypass* de UAC en local no tiene sentido, pero tiene mucho sentido si un *malware* se aprovecha de ello o un atacante con una sesión remota en el equipo consigue aprovecharse de este tipo de técnicas para lograr ejecutar código en un contexto elevado o de integridad alta.

3. UAC-A-Mola

Cómo se ha explicado anteriormente, existen numerosas técnicas que permiten saltar la protección proporcionada por el UAC bajo ciertas circunstancias. Con el objetivo de unificar todo el conjunto de técnicas existente y automatizar el descubrimiento de nuevas debilidades, así como la protección ante las que son conocidas, se propone la construcción de una herramienta basada en módulos que permita la detección y explotación de las debilidades conocidas y el descubrimiento de otras nuevas.

La herramienta sigue una metodología IDEM: Investigación de procesos potencialmente vulnerables a un *bypass* de UAC, detección de este tipo de debilidades, explotación y mitigación.

3.1. Arquitectura

La principal característica de la herramienta es su arquitectura modular. Una interfaz de línea de comandos permite a los usuarios cargar los módulos que quieran utilizar en tiempo de ejecución. Esto proporciona un uso sencillo y eficiente, permitiendo la utilización de varios módulos en una sola ejecución de la herramienta. Por otra parte, dota a la herramienta de una gran extensibilidad

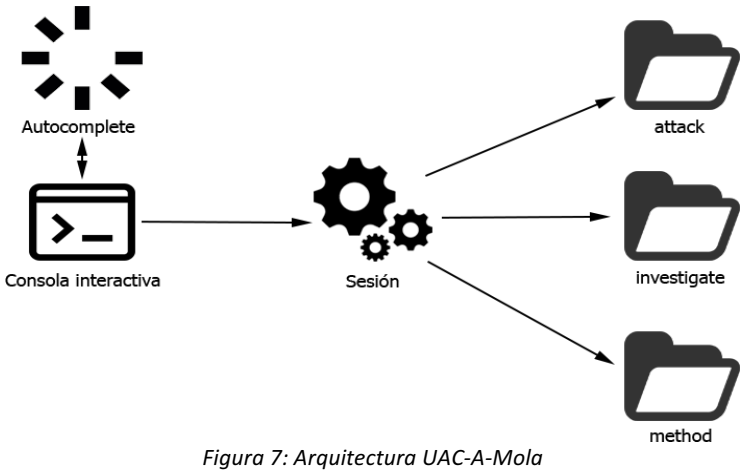

Figura 7: Arquitectura UAC-A-Mola

Su estructura modelo, vista, controlador es lo que proporciona la capacidad de extender cada una de sus partes de manera aislada y sencilla.

3.1.1. Vista

La vista se corresponde con la consola interactiva que se muestra en el diagrama anterior. Su principal función es recoger los argumentos que el usuario introduce por pantalla, realizar un procesamiento sobre ellos, e instanciar una sesión determinada en el caso de que se pretenda cargar un determinado módulo.

3.1.2. Controlador

El controlador se corresponde con la clase *session,* y permite la interacción entre la consola interactiva y los módulos personalizables. Cuando se carga un nuevo módulo, la consola instancia un objeto *session* determinado, que se encargará de realizar el *import* a través del propio modulo. A partir de este momento, todas las acciones que el usuario lleve a cabo en la consola interactiva se realizarán sobre esta sesión determinada, y sobre el módulo seleccionado. El objeto instanciado servirá de punto de unión entre la consola y el módulo personalizado importado por el usuario.

3.1.3. Modelo

El modelo lo conforman los módulos personalizables y el módulo padre del que heredan todos ellos. Es importando por la sesión en tiempo de ejecución, de tal manera que, aunque el número de módulos disponibles sea muy elevado, el rendimiento de la aplicación no se ve afectado, ya que no se importan todos los módulos al iniciarse.

3.2. Módulos

La herramienta dispone de un sistema de módulos personalizados que permiten y facilitan su extensibilidad. Un módulo es un fichero .py, que implementa la clase *CustomModule* y que hereda de la clase *Module* y sobrescribe algunos métodos de la misma.

3.2.1. Module

La clase *Module* constituye la clase padre de la que heredan todos los módulos personalizados que se almacenarán en los directorios ***attack, investigate o method.*** Establece algunos de los componentes obligatorios que los módulos personalizados deben implementar.

```
class Module(object):
    def __init__(self, information, options):
        self._information = information
        self.options = options
        self.args = {}
        self.init_args()
    def get_information(self):
```

```
        return self._information
    def set_value(self, name, value):
        self.args[name] = value
        self.options[name][0] = value
    def get_value(self, option):
        return self.args[option]
    def get_options_dict(self):
        return self.options
    def get_options_names(self):
        return self.options.keys()
    def init_args(self):
        for key, opts in self.options.items():
            self.args[key] = opts[0]
    @abstractmethod
    def run_module(self):
        raise Exception( 'ERROR: run_module method must be implemented in the child class')
    def check_arguments(self):
        for key, value in self.options.iteritems():
            if value[2] is True and str(value[0]) == "None":
                return False
        return True
```

Cómo se observa en el fragmento de código anterior, la clase *Module* recibe en su constructor dos argumentos, la información del módulo y los parámetros que los usuarios podrán establecer en la consola interactiva cuando lo hayan cargado.

- ***Information:*** Es un diccionario que contiene los siguientes campos, ***Name, Description y Author.*** Contiene la información necesaria para describir la herramienta cuando se invoca la opción ***show*** desde la interfaz de comandos. Es obligatorio rellenar sus campos.
- **Options:** Es un diccionario que contiene el nombre de los parámetros y tres atributos por parámetro, **default_value, description y optional.** Con estos atributos se describe el valor que tendrá por defecto el parámetro en el caso de que el usuario no lo modifique, la descripción del parámetro y si el parámetro es obligatorio para el funcionamiento del módulo o no.

El resto de métodos implementados, son algunos auxiliares para la manipulación de los parámetros y de la información.
Todos los valores introducidos por los usuarios, son almacenados en una variable de la clase ***args,*** que será heredado por los módulos hijo, y que les permite acceder a los parámetros de entrada.

Por último, el método ***run_module()*** establece uno de los requerimientos que las clases hijos deben cumplir. Este método es el que la sesión invocará cuando el usuario seleccione el parámetro ***run*** de la consola interactiva, y por lo tanto, debe ser implementado obligatoriamente por los módulos personalizados.

3.2.2. Custom Module

Como se ha indicado en el apartado anterior, la clase *CustomModule* es la que deben implementar obligatoriamente los módulos personalizados del *framework.* A continuación se muestra una plantilla que representa un módulo estándar:

```
from module import Module
class CustomModule(Module):
    def __init__(self):
```

```
        information = {"Name": "",
                       "Description": "",
                       "Author": ""}

        # -------------name----default_value--desc----required?
        options = {"option_name": [None, "description", True],
                   "option2_name": ["default", "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):
        # To access user provided attributes, use self._args dictionary
        self.args["option_name"]
        self.args["option2_name"]
```

Estas serían las lineas de código necesarias para la implementación de un módulo completamente funcional y que permite ser utilizado por la herramienta. Se puede observar la implementación de la clase ***CustomModule,*** que hereda de la clase ***Module***. En el constructor de la misma, se especifican cada una de las opciones necesarias para inicializar la clase padre.

A continuación, se implementa el método ***run_module*** definido como abstracto por la clase *Module* y a partir del cual se puede acceder a los argumentos introducidos por el usuario desde la consola interactiva a través de la variable de la clase padre ***args***.

3.2.3. Implementación de un módulo real

En esta sección se presentan los pasos a seguir para la creación de un módulo real, desde su implementación hasta su integración en la herramienta.

- *Implementación*

La implementación de un módulo real parte de la plantilla presentada en el apartado anterior.

- *Integración en la herramienta*

La integración de un módulo en la herramienta es muy sencilla, lo único que se requiere es copiar el archivo *.py* que contiene la clase *CustomModule* en uno de los directorios dentro de la carpeta *modules.* La herramienta buscará automáticamente en este *path* del sistema cuando vaya a cargar un nuevo módulo e importará en tiempo de ejecución el que se seleccione.

3.2.4. Module loading

El proceso que se utiliza para cargar un módulo en la herramienta en tiempo de ejecución está representado por el siguiente diagrama:

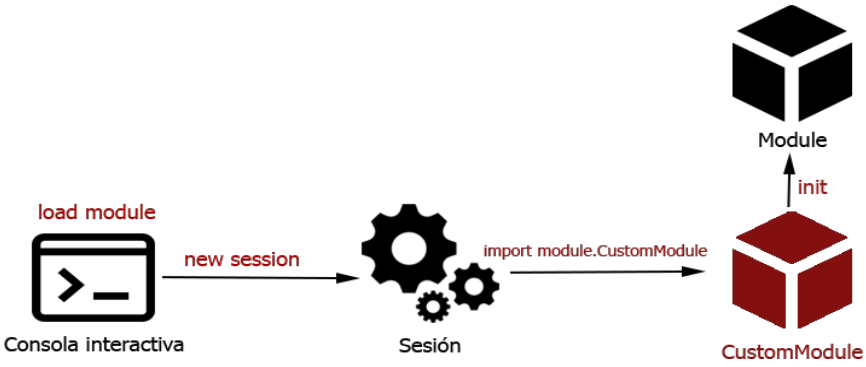

Figura 8: Carga de módulo

El proceso comienza en la consola interactiva, mediante el comando *load* el usuario selecciona un módulo determinado. Cuando se ejecuta este comando, el *framework* internamente crea una nueva sesión. La sesión tiene el contexto del módulo que se desea ejecutar y actuará de intermediario entre la consola interactiva y el módulo personalizado. A partir de este momento, todos los comandos que se ejecuten en la consola serán en una sesión determinada y el contexto de un módulo determinado. Cuando el usuario carga otro módulo distinto, la sesión presente (si la hay) se destruye y se crea una nueva sesión con el contexto del nuevo módulo.

3.2.5. Clasificación de los módulos

Los módulos de la herramienta se pueden clasificar en tres grupos *attack*, *investigate* y *utils*. Cada uno de los contenedores de módulos es un directorio, facilitando la integración de nuevos módulos en la herramienta. A continuación, se explicará cada uno de ellos según la funcionalidad que representan.

- ***Attack***

El directorio *attack* está compuesto por los módulos que automatizan los ataques existentes contra UAC. Con ellos se puede verificar si un sistema determinado es vulnerable a alguna de las técnicas existentes.

- ***Investigate***

En este directorio se dispone de los módulos de investigación o descubrimiento. Su función principal es la asistencia a la hora de identificar debilidades en sistemas operativos Windows 7, 8, 8.1y 10 o el descubrimiento de nuevos problemas de seguridad relacionados con el UAC.

- ***Utils***

En el directorio Utils se encuentran los módulos que sirven de soporte para el desempeño de las tareas nombradas anteriormente.

3.3. Session

El módulo *session* es una de las partes más importantes de la herramienta. Comprende el punto de unión entre la consola interactiva y los distintos módulos. Cuando el usuario carga un módulo determinado, este se ejecuta en una sesión determinada que almacena el contexto para dicho módulo, de tal manera, que todas las acciones que el usuario lleva a cabo desde la consola, se hacen en el contexto del módulo. A continuación, se presentan algunos fragmentos de código de la clase:

```
class Session(object):
    def __init__(self, path):
        self._module = self.instantiate_module(path)
        self._path = path
```

El constructor de la clase *session* es el encargado de cargar el módulo ent tiempo de ejecución. Una vez el módulo se ha cargado, se retorna una instancia de la clase *CustomModule* del mismo `return m.CustomModule()`, de esta manera, la clase session puede acceder a todos los atributos definidos por el creador del módulo para presentárselos al usuario y obtener su valor.

Algunos de los métodos interesantes que implementa son, `def show(self)` compone la respuesta que presenta la consola interactiva cuando el suaurio ejecuta el comando show en el contexto de un módulo determinado, `def run(self)` es el método encargado de invocar la función *run_module()* definida en la clase *Module* e implementada en la clase *Custo*mModule, `def instantiate_module(self, path)` se encarga de cargar el módulo seleccionado por el usuario desde la consola interactiva mediante el comando *load* e instanciar la clase *CustomModule* que debe implementar.

3.4. Libraries

El directorio *libraries* que se encuentra en la carpeta principal de la aplicación, contiene algunos módulos complementarios que facilitan la realización de ciertas tareas. Cómo ejemplo de módulos que se encuentran en este directorio, cabe destacar:

- ***winreg.py***

Este módulo construye una interfaz sencilla sobre el módulo de python _winreg para el tratamiento del registro de windows. Implementa una serie de métodos que permiten la creación, modificación y eliminación de las claves y los valores. Algunos de los métodos a destacar son,

```
def create_key(self, key, subkey)
def non_existent_path(self, key, subkey)
def set_value(self, key, subkey, value)
def del_value(self, key, value='')
def create_value(self, key, value_name, value)
```

Además de las funcionalidades citadas anteriormente, también implementa un mecanismo de restauración del estado del registro al punto inmediantamente anterior de la edición, mediante el método `def restore(self, key, value='')`

- ***procmonXMLfilter***

Este módulo facilita el filtrado de información en un XML generado por la herramienta *procmon.* A través de este módulo se pueden realizar filtrados eficientes y sin consumir recursos de memoria elevados en fichero de tamaño elevado (200MB o superior).
Los principales métodos que implementa son:

```
def by_operation(events, operation)
def by_result(events, result)
def by_process(events, proc_name)
def by_path(events, path)
def by_pid(events, pid)
def by_pattern(events, pattern)
```

4. Resultados

A continuación, se presentan algunos de los resultados obtenidos con la herramienta:

Path: C:\Windows\System32			
OS	**Nº autoelvate binaries**	**Fileless**	**Binaries Names**
Windows7	56	4	eventvwr.exe CompMgmtLauncher.exe sdclt.exe sdclt.exe /kickoffelev
Windows8.1	60	4	eventvwr.exe CompMgmtLauncher.exe slui.exe sdclt.exe /kickoffelev
Windows10	60	3	sdclt.exe sdclt.exe /kickoffelev fodhelper.exe

Path: C:\Windows\System32			
OS	**Nº autoelevate binaries**	**DLL Hijacking**	**Binaries Names**
Windows7	56	12	CompMgmtLauncher.exe ComputerDefaults.exe eventvwr.exe hdwwiz.exe iscsipl.exe msconfig.exe MultiDigiMon.exe Netplwiz.exe odbcand32.exe OptionalFeatures.exe perfmon.exe tcmsetup.exe
Windows8.1	60	15	CompMgmtLauncher.exe ComputerDefaults.exe hdwwiz.exe iscsipl.exe MSchedExe.exe msconfig.exe MultiDigiMon.exe Netplwiz.exe odbcand32.exe OptionalFeatures.exe printui.exe systemreset.exe

			SystemSettingsRemoveDevice.exe tcmsetup.exe
Windows10	60	13	ComputerDefaults.exe fodhelper.exe iscsipl.exe MSchedExe.exe msconfig.exe MultiDigiMon.exe Netplwiz.exe odbcad32.exe OptionalFeatures.exe printui.exe systemreset.exe SystemSettingsRemoveDevice.exe tcmsetup.exe

Para obtener los resultados, se han utilizado los módulos autoElevate_search, que obtiene, a partir de una ruta del sistema o una lista predefinida, todos los binarios con el atributo autoelevate puesto a True dentro de su manifest, fileless_discovery, este módulo busca todas las llamadas a la clave HKCU del registro que no hayan sido encontradas por el binario, las crea y les introduce un valor determinado, posteriormente ejecuta el binario y comprueba el resultado. Por último, se utiliza el módulo dll_discovery, que proporciona una manera de automatizar el proceso de creación del árbol de directorios y la dll maliciosa necesaria para comprobar si el binario es vulnerable a dll hijacking.

Los resultados de dll hijacking que se muestran han sido verificados con una dll genérica, a pesar de esto, la herramienta detecta muchos otros binarios que, aunque no son vulnerables con dicha dll, podrían serlo realizando algunos ajustes a la misma. Verificando todos los binarios sospechosos, el número de resultados de dll hijacking se duplicaría.

La herramienta es capaz de detectar todas las técnicas de bypass de UAC existentes hasta el momento, permitiendo su uso como herramienta de diagnóstico tanto en versiones antiguas del sistema operativo Windows, como en versiones actuales o futuras. Adicionalmente, su capacidad para ser extendida de forma sencilla permite la incorporación inmediata de cualquier nueva técnica que vaya apareciendo.

5. Referencias

[1] https://support.microsoft.com/es-es/help/922708/how-to-use-user-account-control-uac-in-windows-vista

[2] https://github.com/hfiref0x/UACME

[3] http://www.elladodelmal.com/2017/03/una-nueva-forma-de-saltarse-uac-en.html

[4] https://github.com/pablogonzalezpe/metasploit-framework/blob/master/scripts/ps/invoke-compMgmtLauncher.ps1

[5] http://www.elladodelmal.com/2016/08/como-ownear-windows-7-y-windows-10-con.html

[6] https://enigma0x3.net/2016/08/15/fileless-uac-bypass-using-eventvwr-exe-and-registry-hijacking/

[7] http://www.elladodelmal.com/2017/04/fileless-2-un-nuevo-si-uno-mas-otra-vez.html

[8] http://www.elladodelmal.com/2017/05/fileless-3-bypass-uac-en-windows-10.html

[9] https://isc.sans.edu/forums/diary/Malicious+Office+files+using+fileless+UAC+bypass+to+drop+KEYBASE+malware/22011/