Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Explotación de Vulnerabilidades de Seguridad

Autor: Alejandro Rodríguez Duro
Tutor(a): Fernando Pérez Costoya

Madrid, Abril 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Explotación de Vulnerabilidades de Seguridad

Abril 2024

Autor: Alejandro Rodríguez Duro
Tutor: Fernando Pérez Costoya
Arquitectura y Tecnología de Sistemas Informáticos
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Desde la aparición del primer virus informático 'Creeper' en 1971, la aparición de vulnerabilidades en los dispositivos electrónicos y su aprovechamiento por atacantes ha ido creciendo año tras año y siendo estos ataques cada vez más comunes hasta el punto que hoy en día existe toda una industria formada alrededor de estas incidencias que son capaces de ocasionar pérdidas millonarias, como en el caso del virus *Wannnacry*[1] o el ataque sobre las instalaciones de *Colonial Pipeline* [2].

Estos ataques, aparte de causar pérdidas astronómicas en grandes negocios e instalaciones públicas, también retumban en la vida del ciudadano medio, ya que acaban sufriendo parones en los servicios proporcionados por dichos negocios, como la paralización de las gasolineras en EEUU debido al ataque sobre Colonial Pipeline o, al tener su información utilizada como moneda de cambio, tal como ocurrió con el ransomware Wannacry.

Debido a la creciente necesidad de combatir el cibercrimen, este trabajo se centra en proporcionar a investigadores, profesionales de seguridad y estudiantes un entorno controlado y seguro para la preparación de pruebas y evaluación de las vulnerabilidades de seguridad que los atacantes suelen utilizar, todo esto en entornos virtualizados de rápido montaje.

En este proyecto se pretende utilizar tecnologías de virtualización para crear entornos aislados y reproducibles, lo que permitirá realizar pruebas sin comprometer la integridad de los sistemas de los investigadores. Además, se integrarán herramientas de gestión y análisis para facilitar la configuración, ejecución y evaluación de las pruebas.
Todo esto vendrá preparado para ser rápidamente accesible y de manera gráfica, los usuarios podrán elegir un exploit que haya sido añadido a la base de datos y, basándose en la compatibilidad de dicho exploit con el SO, podrán elegir la máquina virtual de las catalogadas que más se pueda ajustar al exploit y ya finalmente ejecutarlo y probar la eficacia de dicho exploit, todo esto mediante métricas de éxito y eficiencia. Además, el usuario podrá investigar CVEs recientes para poder realizar más trabajo de investigación y añadirlos a su catálogo personal.

Abstract

Since the creation of the first computer virus 'Creeper' in 1971, the discovery of vulnerabilities in electronic devices and their exploitation by attackers has been growing year after year and these attacks have become increasingly common to the point that, today, there is a whole industry formed around these incidents that are capable of causing losses in the millions, as in the case of the *Wannnacry* virus[1] or the attack on the facilities of *Colonial Pipeline* [2].

These attacks, apart from causing astronomical losses in large businesses and public facilities, also reverberate in the life of the average citizen, as they end up suffering downtime in the services provided by these businesses, such as the paralysis of gas stations in the US due to the attack on Colonial Pipeline or, having their information used as a bargaining chip, as happened with the Wannacry ransomware.

Due to the growing need to combat cybercrime, this work focuses on providing researchers, security professionals and students with a controlled and secure environment for test preparation and evaluation of security vulnerabilities that attackers commonly use, all in rapidly assembled virtualized environments.

Virtualization technologies will be used to create isolated, reproducible environments, enabling testing without compromising the integrity of the researcher systems. In addition, management and analysis tools will be integrated to facilitate the configuration, execution and evaluation of experiments.
All this will be prepared to be quickly accessible and in a graphical way, users will be able to choose an exploit that has been added to the database and, based on the compatibility of the exploit with the OS, they will be able to choose the virtual machine from those listed that best fits the exploit and finally run it and test the effectiveness of the exploit, all this through metrics of success and efficiency. In addition, the user will be able to investigate recent CVEs to be able to do more research work and filter by vulnerabilities that can be adjusted to the desired filters.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación

En la actualidad, existimos en un contexto en el que la tecnología se ha fusionado con nuestro día a día en casi todos los aspectos, lo cual significa que, aparte de depender de estos dispositivos para poder realizar nuestros quehaceres, gran parte de nuestra vida queda registrada ahora en estos aparatos. Esto expone la evidente necesidad de, por una parte, asegurar el correcto funcionamiento de estos dispositivos de los que dependemos, y por otro lado, el protegerlos de atacantes externos que pretendan causar cualquier perjuicio o ir tras nuestra información en detrimento de nuestra privacidad e integridad.

Todos estos ataques suelen surgir a raíz de vulnerabilidades, errores o comportamientos no definidos en los ordenadores que estos atacantes buscan descubrir y aprovechar. Para poder comprender y protegerse frente a las amenazas e incidentes que estas vulnerabilidades exponen, lo mejor es probarlo por uno mismo y comprender el funcionamiento de dichas vulnerabilidades. Para ello, la idea de tener una plataforma *todo en uno* resulta interesante, sobre todo por temas de comodidad y eficiencia.

Con este proyecto, aparte de poder investigar con, hasta cierto punto de profundidad, ciertos CVEs y exploits que aprovechen dichas vulnerabilidades, también se pretende realizar el proceso de investigar todos estos factores mucho más sencillo y automatizado de manera que pueda ser útil, tanto a estudiantes que se estén empezando a introducir en la materia como a investigadores o profesionales que requieran de testeo en diferentes especificaciones de una manera cómoda y rápida.

El objetivo de este trabajo es proporcionar una herramienta de fácil uso que permita probar exploits y vulnerabilidades, analizar sus detalles, comprobar si son efectivos en distintos sistemas operativos y sus varias versiones.

1.2. Trabajo previo y estado del arte

En el panorama actual existe ya una herramienta llamada Metasploit. Metasploit es una herramienta de penetración de código abierto que se utiliza para desarrollar y ejecutar exploits contra sistemas informáticos con el fin de evaluar su seguridad. Aunque la herramienta si permite ejecutar exploits de manera relativamente cómoda, no posee un repertorio de entornos de virtualización rápidos donde probar dichos exploits que, por ejemplo, estén disponibles a la hora de ejecutar un exploit que solo es vulnerable en cierta versión de un SO.

Respecto a las herramientas que se usaran para este proyecto, se utilizara Python[3] para el desarrollo de la interfaz gráfica y como medio de comunicación con Vagrant[4], herramienta que permite crear los entornos virtualizados rápidos. Python posee una librería llamada Qt[5] que se utilizara para el diseño e implementación de la interfaz gráfica, todo con el objetivo que el entorno sea visual, cómodo y accesible.

Respecto a Vagrant, nos provee de la capacidad de, una vez preparada una máquina virtual, que puede haber sido completamente personalizada, ya sea en la versión del SO, el kernel o aplicaciones que tenga instaladas, diferentes especificaciones de la configuración del sistema, nos permite empaquetarla y tener lista en un único comando una instancia de dicha máquina virtual ya preparada. Así se puede conseguir un repertorio de máquinas ya previamente configuradas de forma que se puedan probar en ellas herramientas de explotación que requieran de cualquier grado de precisión respecto a las condiciones de prueba.

La base de datos en la que se pretende recabar toda esta información se implementará utilizando PostgreSQL[6]. Esta elección se debe a la flexibilidad, extensibilidad y facilidad de uso que ofrece PostgreSQL, lo que garantiza que podamos almacenar y gestionar eficientemente los datos relacionados con las configuraciones de las máquinas virtuales y los exploits probados.

Respecto al servicio Restful que se pretende implementar, se hará uso de la página de CVE oficial[7] a la cual se le mandaran peticiones desde nuestra aplicación y, a partir de los resultados devueltos por dicha página, se mostrara una lista de resultados atendiendo a los filtros aplicados. Esto permitirá a los usuarios poder buscar las últimas vulnerabilidades descubiertas u otras más antiguas que cumplan con unos filtros que ellos apliquen de forma rápida para poder realizar trabajo sobre ello.

Además, se explorarán otras herramientas y enfoques utilizados en proyectos similares. Se analizarán plataformas existentes para pruebas de seguridad, como Exploit-DB y VulnHub, para identificar características, ventajas y limitaciones que puedan influir en el diseño y desarrollo de la herramienta propuesta.

Capítulo 2

Desarrollo

El capítulo de Desarrollo constituye el núcleo de este Trabajo de Fin de Grado, describiendo en detalle el diseño, la implementación y la evaluación de una herramienta avanzada para la gestión y explotación de vulnerabilidades de seguridad en entornos virtualizados. Este capítulo detalla el proceso integral de creación de la herramienta, desde la concepción inicial hasta su despliegue final, subrayando las decisiones técnicas clave y las justificaciones detrás de la elección de tecnologías y metodologías de desarrollo.

Objetivos del Capítulo El objetivo principal de este capítulo es proporcionar una comprensión clara y detallada de cómo se ideó, diseñó, desarrolló y probó la herramienta. Esto incluye:

1. **Selección Tecnológica**: Examinación de las tecnologías elegidas para el desarrollo de la herramienta, incluyendo lenguajes de programación, frameworks, bases de datos y sistemas de virtualización.
2. **Diseño de la Arquitectura**: Descripción detallada de la arquitectura del sistema, incluyendo la interfaz de usuario, la lógica del negocio, la gestión de datos y la interacción con entornos virtualizados.
3. **Implementación**: Narración paso a paso del proceso de implementación, destacando los desafíos encontrados y cómo se abordaron.
4. **Pruebas y Validación**: Descripción de las pruebas realizadas para asegurar la funcionalidad y seguridad de la herramienta, junto con la metodología de pruebas y los resultados obtenidos.

Importancia del Desarrollo

Proporcionar un entorno seguro y controlado para la explotación de vulnerabilidades no solo ayuda a entender mejor estas amenazas, sino que también permite mejorar las defensas contra ataques maliciosos. La sección de desarrollo muestra el esfuerzo y la meticulosidad aplicados en la construcción de una herramienta orientada tanto a fines educativos como profesionales.

Estructura del Capítulo

Este capítulo está organizado en varias secciones que reflejan las fases del ciclo de vida del desarrollo de software:

- **Diseño**: Discusión sobre el diseño conceptual y detallado del sistema.
- **Tecnología**: Exploración de las decisiones tecnológicas y sus impactos.
- **Implementación**: Detalles sobre la codificación, integración y configuración del sistema.
- **Pruebas**: Cobertura de las pruebas de unidad, integración y sistema, junto con la validación de los resultados.

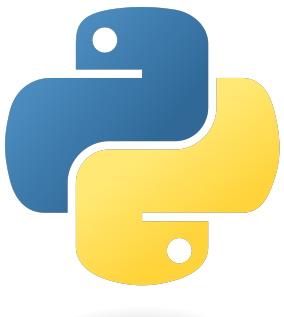
Con el desarrollo de esta herramienta, se busca avanzar en la comprensión de las vulnerabilidades de seguridad y mejorar la capacidad para enfrentar los desafíos de seguridad en el ámbito informático. Este capítulo, por lo tanto, no solo documenta el proceso de creación de una herramienta de software, sino que también refleja un esfuerzo concertado para contribuir al campo de la ciberseguridad.

2.1. Tecnología

Aquí se pretende mostrar con qué herramientas se ha construido todo lo presentado y también, de todas las herramientas disponibles, por qué se han utilizado las aquí mostradas.

2.1.1. Python

Se seleccionó Python debido a la sencillez que presenta este lenguaje de programación a la hora de programar y el gran repertorio de bibliotecas que tiene a su disposición, incluyendo en esta amplia lista bibliotecas gráficas como podían ser Qt o Tkinter, que permiten un gran ahorro de tiempo en ciertos casos que se requiera de una función específica que simplemente se puede suprimir el tiempo de desarrollo por el de importar una biblioteca que nos brinde esa función. También, al inicio del desarrollo se preveía que hubiese una funcionalidad para conectarse a dispositivos remotos mediante sockets, siendo también muy cómodo de realizar en Python, fue otro de los motivos que llevaron a la selección de este lenguaje.
En resumen, la elección de Python para el desarrollo de este proyecto se fundamenta en su simplicidad, en su gran capacidad de integración de librerías, la rapidez de desarrollo y el enorme soporte por parte de su comunidad. Estas características hacen de Python una opción ideal para abordar los desafíos técnicos y logísticos inherentes a la seguridad informática y la gestión de vulnerabilidades en entornos virtualizados



2.1.2. Vagrant

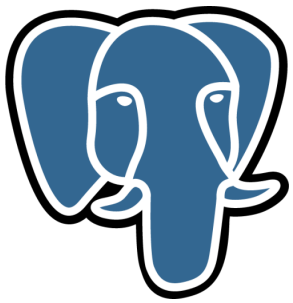
Ya que el proyecto requería de virtualizar entornos a la hora de ejecutar y medir los exploits probados, se buscaron opciones y las más prometedoras resultaron ser Docker, herramienta muy conocida y utilizada, y Vagrant, que aun siendo mucho menos conocida y común ha resultado ser la seleccionada por varios motivos.
De primeras, Vagrant gestiona máquinas virtuales que proporcionan un aislamiento completo del host, lo cual es esencial para probar software malicioso o exploits sin el riesgo de afectar el sistema operativo host. Esta característica resulta esencial a la hora de analizar software potencialmente peligroso para no perjudicar al investigador a la hora de examinar estas amenazas. Docker, aunque también ofrece aislamiento, no lo hace al nivel de una máquina virtual completa. Los contenedores comparten aspectos del sistema operativo host, lo que podría teóricamente permitir que un exploit comprometa no solo el contenedor sino también el host, especialmente si existe una vulnerabilidad en el propio Docker o en el kernel del sistema. Otro aspecto a tener en cuenta es que Vagrant permite configurar detalladamente el entorno virtual, incluyendo la elección del sistema operativo, la configuración de la red, el almacenamiento, y otros recursos del hardware virtual. Al poder manejar hasta el más mínimo detalle de la virtualización, ya sea el hardware que supuestamente usa el sistema, el software instalado y activo, u algún otro detalle específico, permite que Vagrant pueda preparar a la perfección el entorno de pruebas. Además, Vagrant es altamente efectivo en la gestión de múltiples entornos virtualizados de manera consistente a través de su interfaz de línea de comandos y su archivo Vagrantfile. Esto permite a los usuarios lanzar y gestionar entornos complejos con un simple comando, asegurando que cada entorno esté configurado de manera uniforme y según lo especificado. Docker también maneja múltiples entornos, pero se centra más en la portabilidad de la aplicación y menos en la reproducción exacta del entorno de hardware subyacente. Vagrant es ampliamente reconocido y utilizado en la comunidad de desarrollo de pruebas y seguridad informática. Tiene una gran cantidad de recursos y una comunidad activa que puede proporcionar soporte, plug-ins y cajas pre configuradas.

Por todos estos detalles y lo sencillo que resultaron las primeras pruebas con Vagrant, finalmente se seleccionó esta herramienta como entorno de virtualización predeterminado para el proyecto.

2.1.3. PostgreSQL

PostgreSQL es ampliamente utilizado debido a que posee una gran robustez y confiabilidad como sistema de gestión de bases de datos. Aunque si es cierto que existían otras posibles opciones, probablemente igual de válidas para un proyecto de esta naturaleza en su etapa actual, PostgreSQL posee grandes cualidades que, ya de por sí lo hacen resaltar frente al resto de herramientas. PostgreSQL ofrece características de seguridad robustas que incluyen autenticación, encriptación en tránsito y en reposo, control de acceso basado en roles y etiquetas de seguridad para el control de acceso obligatorio (MAC). Estas características pueden resultar muy útiles para proteger datos sensibles y asegurar que solo los usuarios autorizados puedan acceder a información crítica. La comunidad de PostgreSQL es grande y activa, que puede proporcionar un amplio soporte a través de documentación detallada, foros en línea y contribuciones de código. Esto asegura que los desarrolladores puedan obtener ayuda y recursos rápidamente, lo cual es importante para mantener la continuidad y la estabilidad del desarrollo en proyectos de largo plazo. PostgreSQL maneja cargas de trabajo grandes con eficacia, gracias a su capacidad para manejar un gran número de transacciones por segundo y su soporte para bases de datos grandes. Además, ofrece herramientas para la replicación y partición de datos, facilitando la escalabilidad según van creciendo las necesidades del proyecto.

En conclusión, la elección de PostgreSQL para este proyecto se basa en su solidez, capacidad de manejo de datos complejos y avanzados, extensibilidad, características de seguridad robustas, y su desempeño escalable. Estas cualidades hacen que sea una excelente opción para proyectos que requieren alta confiabilidad, seguridad y una gestión eficiente de grandes volúmenes de datos en el campo de la ciberseguridad. Quizás en el estado actual del proyecto todas estas cualidades no puedan resultar explotadas al máximo, pero en vista al futuro resulta útil tener una base confiable y altamente escalable como lo es PostgreSQL.



2.2. Diseño e implementación de la herramienta

Empezando por el backend, se pretende explicar cómo se ha llevado a la realidad la idea de diseño inicial.

2.2.1. Interfaz del Usuario

Pestaña de inicio

La interfaz está dividida en cinco 'tabs', pestañas que permiten desplazarse desde dentro de la aplicación rápidamente y cada una de ellas tiene asignada su propia pantalla en la que podemos realizar las distintas acciones que nos permite esta aplicación o visualizar cierta información que puede resultar útil para la evaluación de vulnerabilidades/exploits. Al iniciar la aplicación se comienza en el tab de inicio, donde se puede elegir un perfil de ejecución de los ya creados a través de un combo box e ir a la pestaña de ejecución o, pulsar al botón de crear nuevo perfil para saltar directamente a la pestaña de creación de perfiles.

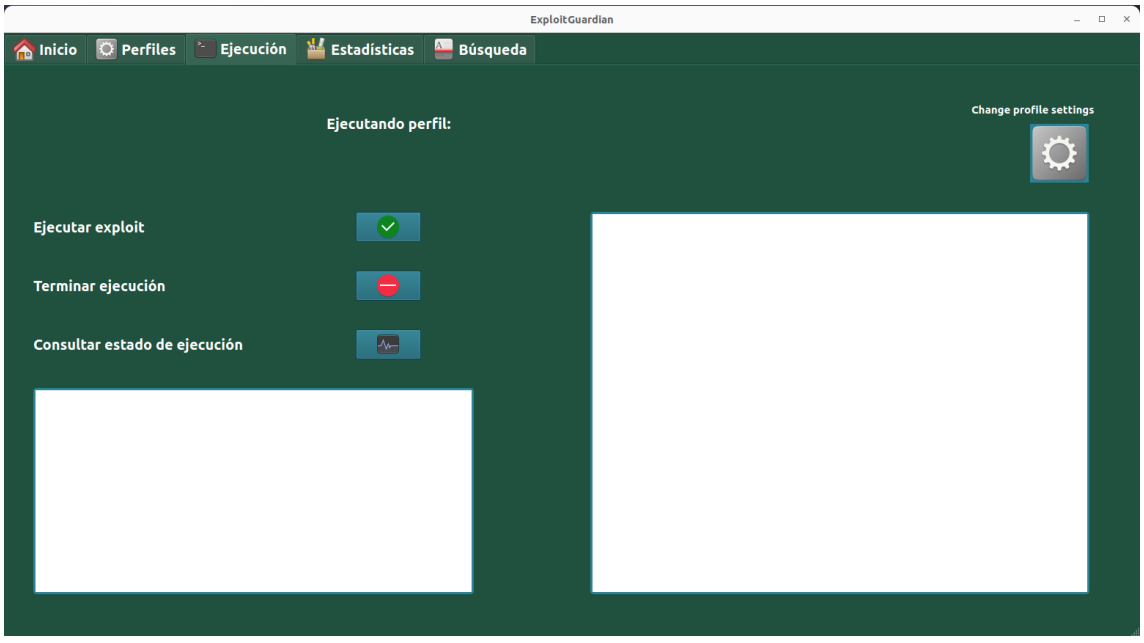
Figura 2.1: Pestaña de inicio del programa

Pestaña de perfiles

Si se va a la pestaña de perfiles, se puede agregar distintos componentes como un script de inicio, un exploit, un CVE, una máquina virtual... Una vez ya tenemos los componentes deseados para nuestro perfil de ejecución, se selecciona perfil en la caja combo y se añade, eligiendo los componentes ya guardados en la base de datos. También si se desea, se puede modificar un perfil ya existente para cambiar algún detalle si fuese necesario. Una vez está todo en orden se puede observar todos los componentes presentes en la base de datos desde una lista que está presente a la derecha de la pantalla, que permite filtrar entre componentes, perfiles y test runs.

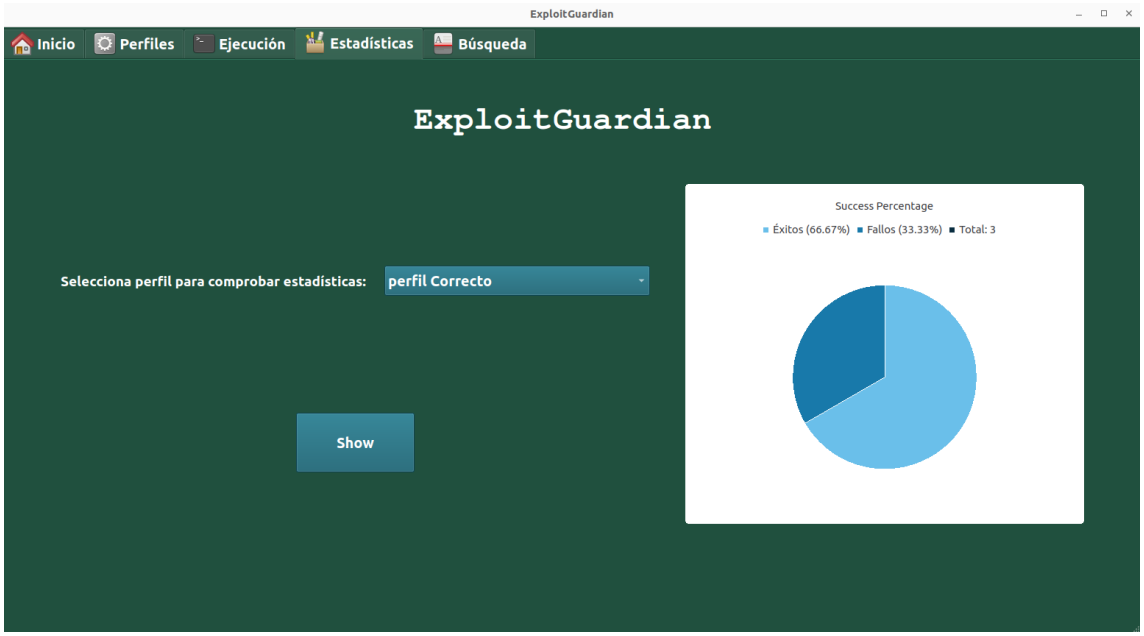
Pestaña de ejecución

Una vez ya en la pestaña de ejecución, podemos ver gracias a una etiqueta situada en la parte superior de la pantalla qué perfil está seleccionado dentro de la aplicación. Se nos presentan cuatro botones, uno que nos permite cambiar la configuración del Vagrantfile que afecta a la cantidad de memoria RAM y número de cores CPU destinados a nuestra virtualización, estos ajustes son aplicados una vez se inicia la máquina virtual y por ello esta opción está situada en esta pestaña. Los siguientes botones permiten tanto iniciar y parar la máquina virtual y la ejecución del script/exploit como la consulta del estado de la máquina virtual para comprobar si está en funcionamiento. Una vez iniciada la ejecución podemos observar en la parte derecha el texto que se imprimiría a través de una terminal, reportándonos el progreso de tanto la inicialización de la máquina virtual como de nuestra prueba. Si nuestra prueba acaba en éxito abajo a la izquierda se nos mostrará el tiempo desde que se pulsó el botón de iniciar hasta que el flag ha sido leída y capturada (momento en el que se entiende que la prueba ha sido un éxito).

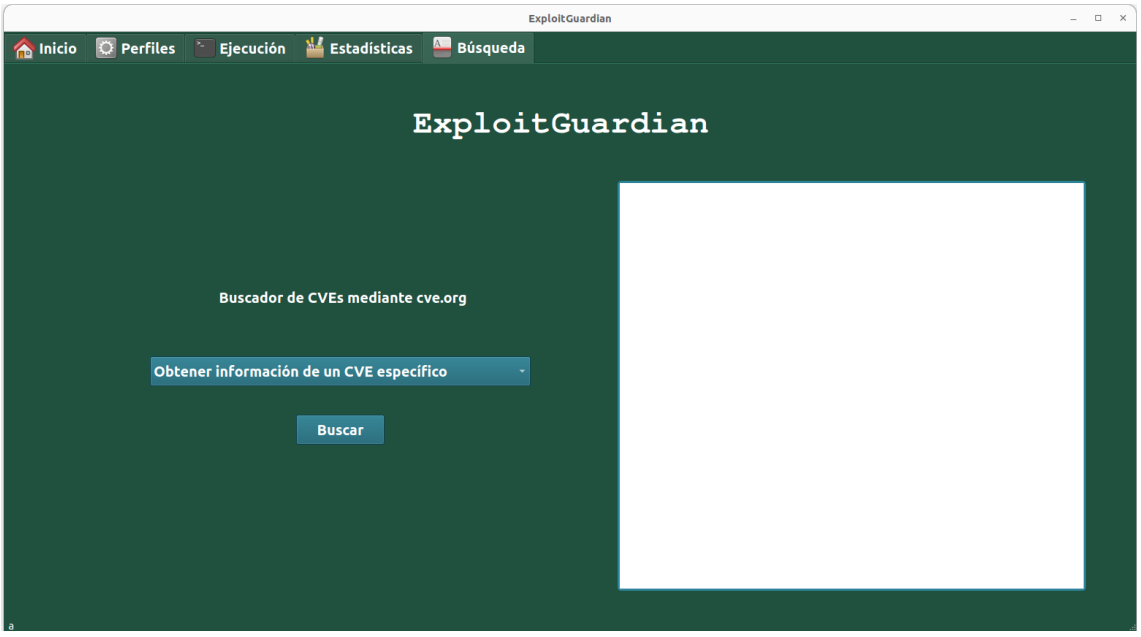


Pestaña de estadísticas

Avanzando ya a la pestaña de Estadísticas, filtrando por el perfil deseado, podemos observar el número total de ejecuciones de dicho perfil y el porcentaje de test runs exitosos de todas esas ejecuciones. Todo esto se muestra en un gráfico circular.



Pestaña de búsqueda Finalmente en la pestaña de búsqueda, tenemos la opción de buscar a través de CVE.ORG información acerca de un CVE en concreto. Una vez encontrado dicho CVE se nos mostrará a la derecha, pudiendo leer la información más importante de dicho CVE y pudiendo importarlo también a nuestra base de datos.



Con esto concluye una descripción general de la interfaz gráfica presentada para este proyecto.

2.2.2. Backend

Estructura del Backend

Como ya se ha mencionado, todo el código está escrito en Python, la estructura del código consiste en un archivo principal llamado **mainwindow.py**, que su labor es la de proporcionar el método principal del programa, inicializando la interfaz y las variables globales para luego llamar a otros métodos complementarios que residen en otros archivos como **popups.py** que recoge la implementación de las ventanas emergentes que aparecen en el programa que permiten que el usuario introduzca datos que a posteriori son introducidos a la base de datos mediante queries de SQL, **apiRequests.py** donde se sitúa el código que realiza las peticiones a la página de **cve.org** y se devuelve la información obtenida. Desde **uiform.py**, que viene producida desde **form.ui** a través de QtCreator, se establece en código Python todos los componentes de la interfaz gráfica del proyecto, simplemente estableciendo que objetos han de mostrarse en el programa, con sus posiciones, tamaños y demás características, estableciéndose las funcionalidades más avanzadas desde el archivo principal **mainwindow.py**.
El archivo **vagrantInterface.pyproject** sirve para guardar los ajustes en el QtCreator y comunicarle al programa que archivos ha de tener en cuenta a la hora de diseñar la interfaz gráfica del programa. La carpeta **Scripts** incluye scripts que reproducen el comportamiento por defecto de una prueba normal del programa y están disponibles como archivos de prueba. Finalmente, la carpeta resources guarda archivos utilizados para distintos componentes del proyecto, por ejemplo una imagen para un botón.

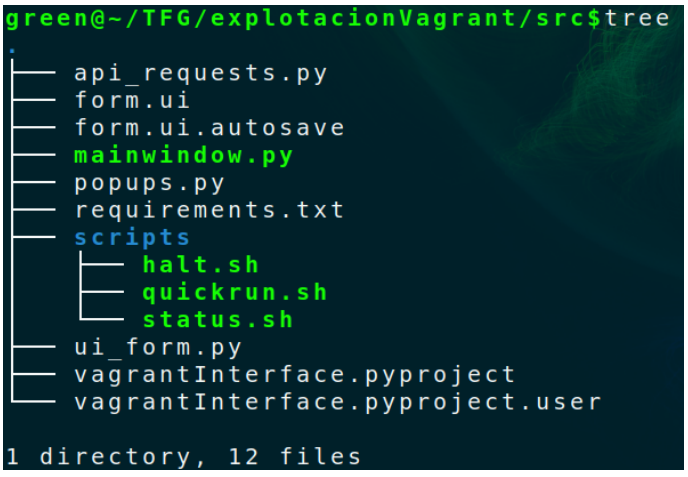


Figura 2.2: Estructura del proyecto con sus archivos

Lógica del Backend

El código comienza estableciendo una conexión a una base de datos PostgreSQL, base de datos en la que se almacenan todos los datos relacionados con los perfiles de configuración, los exploits, y otras informaciones relevantes para la aplicación:

```
# Conexion a bbdd
conn = psycopg2.connect(
    dbname="vagrantTools",
    user="postgres",
    password="postgres",
    host="localhost",
    port="5432"
)
```

Figura 2.3: Datos estándar para la conexión a la base de datos

Mencionando también las librerías más importantes, nos encontramos con `QtCharts` para la visualización de datos, gestión de procesos `subprocess`, y operaciones en segundo plano `QThread`. Incluye interacción con sistemas de archivos `os` y bases de datos PostgreSQL `psycopg2`, organizado a través de una interfaz diseñada en `QtWidgets`. Además de los otros archivos de código ya mencionados `ui_form`, `popups`. Todas estas librerías permiten la ejecución del proyecto de una manera más sencilla.

```
import os
from PyQt6.uic.properties import QtWidgets
from PySide6 import QtCharts
from PySide6.QtCore import QThread, Signal, QElapsedTimer
from subprocess import Popen, PIPE
from ui_form import Ui_MainWindow
from popups import *
import psycopg2
from PySide6.QtWidgets import QApplication, QMainWindow, QListWidgetItem
```

Figura 2.4: Librerías utilizadas en mainwindow.py

Resulta importante resaltar también el esquema diseñado y utilizado para la base de datos. Todos los datos que son requeridos del usuario para definir un perfil de ejecución son guardados en ella. Repasando los componentes, se nos pide guardar un exploit, el cual puede tener o no asociado un CVE, también se nos pedirá un script de inicio para personalizar la ejecución del script, lo cual permite también alterar en cierta medida la colocación del flag, que aunque tiene que ser que se lea en algún momento la cadena `"PWNED"`, con esto se da la flexibilidad de colocar el flag donde se desee. Finalmente se nos pedirá agregar una máquina virtual, la cual es el directorio en el que tengamos guardado nuestra máquina Vagrant con su Vagrantfile, esta máquina virtual tendrá asociada el sistema operativo que posee. Una vez todos estos componentes han compuesto un perfil, este perfil generara entradas del tipo `test_run` que almacenara si la ejecución fue un éxito y el tiempo que tardo en alcanzar el flag.

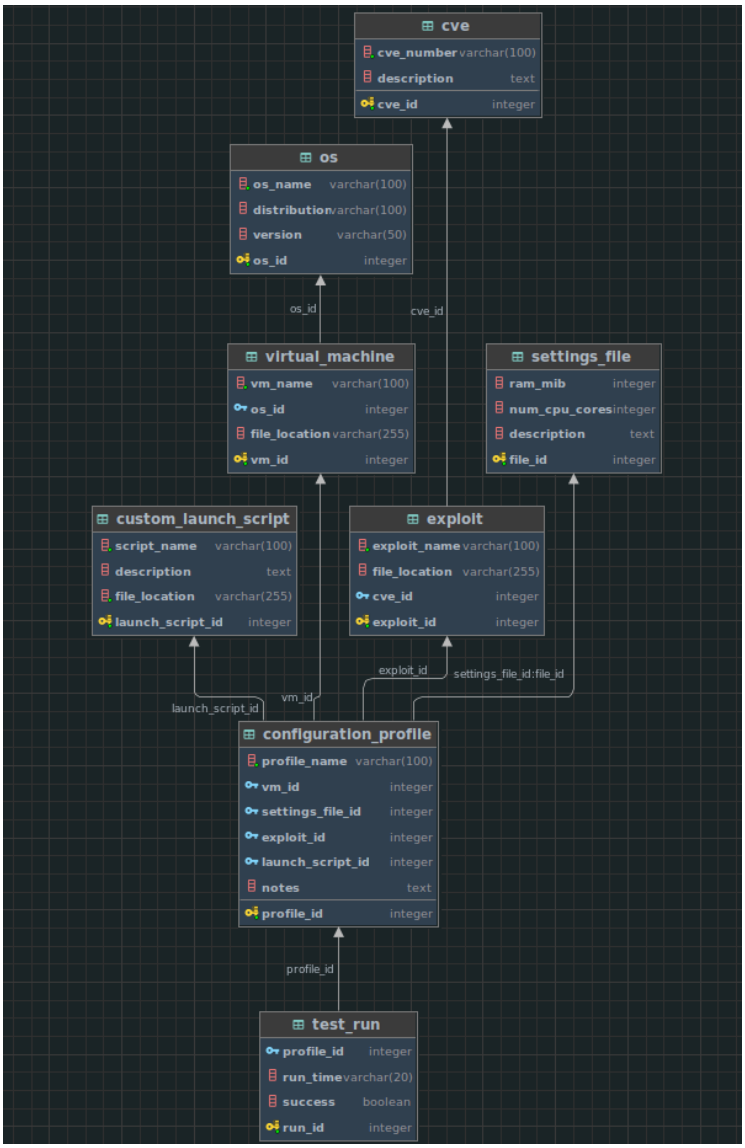


Figura 2.5: Diagrama de la base de datos

La clase `WorkerThread` es un subproceso que hereda de `QThread`. Esta clase maneja operaciones que pueden tomar mucho tiempo, como la ejecución de exploits, sin bloquear la interfaz de usuario. Utiliza señales para comunicarse con la GUI y actualizarla según los procesos terminan y se obtiene la información necesaria.
Se encarga también de ejecutar el exploit en la máquina virtual Vagrant y una vez ejecutado, si por pantalla se imprimiese el texto "PWNED", que hace las veces de palabra clave, la cual ha de ser colocada en el archivo que usemos de flag en la virtualización mandará una señal que indica que el exploit ha funcionado correctamente y este test run puede ser contado como exitoso. Podemos ver que el método run utiliza un script usando una ruta de archivos del sistema, la cual dependiendo del perfil seleccionado será diferente y habrá sido introducida por el usuario en la base de datos.

```
class WorkerThread(QThread):
    output_changed = Signal(str)
    pwned_signal = Signal(int)
    global selected_profile_id

    green
    def __init__(self, parent=None):
        super().__init__(parent)
        self.timer = QElapsedTimer()
        self.trigger = False  # Initialize trigger to False
        self.pwned_found = False  # Initialize pwned_found to False

    green
    def run(self):
        process = Popen( args: [quickrun_path, ""], shell=True, stdout=PIPE, stderr=PIPE, universal_newlines=True)
        self.timer.start()

        while process.poll() is None:
            output = process.stdout.readline()
            if output:
                self.output_changed.emit(output.strip())
                if "PWNED" in output:
                    elapsed_time = self.timer.elapsed()
                    self.pwned_signal.emit(elapsed_time)
                    self.update_database_success(elapsed_time)  # Update the database for success
                    self.pwned_found = True
                    break

        # If "PWNED" was not found, set trigger to True
        if process.poll() is not None and not self.pwned_found:
            self.trigger = True
```

Figura 2.6: Clase WorkerThread para utilizar en segundo plano funciones que bloquearían la interfaz

Poniendo de ejemplo la pestaña de `Estadísticas`, en el código es recurrente la estructura de definir ciertos detalles más avanzados en el archivo `mainwindow.py`, por ejemplo, aquí se realiza una query a la base de datos para buscar todos los perfiles existentes y mostrarlos en el combo box que nos permitirá seleccionar el perfil del cual queremos ver las estadísticas. Después, creamos el gráfico, definiendo tanto las características básicas como las avanzadas, esto se debe a que la versión de QtCreator en Linux tiene un pobre mantenimiento, dando problemas a la hora de importar el plug-in para colocar gráficos y herramientas estadísticas cómodamente como con el resto de componentes. Después de haber posicionado el gráfico, darle nombre y definir en que pestaña se va a mostrar, añadimos los perfiles devueltos por la consulta a la base de datos al combo box y le añadimos la función de showGraph al botón de mostrar estadísticas del perfil.

```
# STATISTICS WINDOW
# Consulta la base de datos para obtener los nombres de los perfiles
cur.execute("SELECT profile_name FROM configuration_profile")
profile_names = [row[0] for row in cur.fetchall()]

# Limpia cualquier entrada previa en la combo box
self.ui.stat_profSelect_cmb.clear()

# Create the chart and chart view
self.chart = QtCharts.QChart()
self.chart.setTitle("Success Percentage")
self.series = QtCharts.QPieSeries()
self.chart.addSeries(self.series)
self.chartView = QtCharts.QChartView(self.chart, self.ui.statistics_tab)
self.chartView.setGeometry(920, 160, 560, 480)
self.chartView.setParent(self.ui.statistics_tab)
#self.chartView.move(100, 100)  # Manually set the position

self.ui.stat_profSelect_cmb.addItems(profile_names)
self.ui.stat_show_btn.clicked.connect(self.showGraph)
```

Figura 2.7: Código correspondiente a la pestaña de estadísticas

El siguiente código corresponde a la función `showGraph()`, aquí se realizan dos peticiones a la base de datos, una para obtener el total de ejecuciones de prueba que pueda tener un perfil y, una segunda petición, para obtener solo las ejecuciones que han sido exitosas. De esta manera podemos realizar una simple división y una resta que nos permite calcular la tasa de éxito de un perfil. Esta tasa de éxito (o fracaso), al finalizar esta función, se actualizará el gráfico que ya estaba presente en la pestaña de estadísticas para reflejar estos resultados. El formateo a posteriori de los resultados muestra solo dos décimas por razones de simplicidad, ya que no es necesario más precisión para esta visualización.

```
def showGraph(self):
    if self.ui.stat_profSelect_cmb.count() == 0:
        QtWidgets.QMessageBox.warning(None, "Aviso", "No se encontraron perfiles.")
        return
    # perfil seleccionado
    selected_profile = self.ui.stat_profSelect_cmb.currentText()
    try:
        # get success data
        cur.execute( query: """
            SELECT COUNT(*)
            FROM test_run tr
            INNER JOIN configuration_profile cp ON tr.profile_id = cp.profile_id
            WHERE cp.profile_name = %s AND tr.success = TRUE
            """,  vars: (selected_profile,))
        success_count = cur.fetchone()[0]
        # Get fail data
        cur.execute( query: """
            SELECT COUNT(*)
            FROM test_run tr
            INNER JOIN configuration_profile cp ON tr.profile_id = cp.profile_id
            WHERE cp.profile_name = %s
            """,  vars: (selected_profile,))
        total_count = cur.fetchone()[0]
        # success %
        if total_count > 0:
            success_percentage = (success_count / total_count) * 100
            failure_percentage = 100 - success_percentage
        else:
            success_percentage = 0
            failure_percentage = 0
        # Reset y añadimos info
        self.series.clear()
        self.series.append("Éxitos ({:.2f}%)".format(success_percentage), success_percentage)
        self.series.append("Fallos ({:.2f}%)".format(failure_percentage), failure_percentage)
        total_label = "Total: {}".format(total_count)
        total_slice = self.series.append(total_label, 0)
        total_slice.setLabel(total_label)

    except Exception as e:
        # Error msg si falla algo
        QtWidgets.QMessageBox.critical(None, "Error", f"Error occurred: {e}")
```

Figura 2.8: Código correspondiente a la función showGraph

2.3. Pruebas y validación

El proceso de evaluación y pruebas es un componente crítico en el desarrollo de cualquier software. Este apartado detalla cómo se ha llevado a cabo la evaluación de la herramienta para comprobar su funcionalidad y eficacia. Las pruebas son esenciales no solo para verificar que el software cumple con los requisitos especificados, sino también para identificar áreas de mejora y asegurar que no se introduzcan nuevos errores durante el proceso de desarrollo.

2.3.1. Repositorio de la aplicación

Se ha almacenado todo el código de la aplicación en este link, el repositorio está disponible para todo el mundo.

Repositorio en Github

Pruebas funcionales

Se procederá a establecer una serie de objetivos y características que la aplicación debería cumplir para poder proporcionar la funcionalidad que prometía este proyecto.

Añadir máquina virtual a la bbdd
Añadir archivo de ejecución a la bbdd
Añadir CVE a la bbdd
Añadir exploit a la bbdd
Añadir perfil personalizado a la bbdd
Modificar ajustes de la virtualización
Ejecutar perfil y evaluar éxito de la ejecución
Observar estadísticas de la ejecución
Obtener información de CVE exterior a la bbdd

Cuadro 2.1: Lista de objetivos funcionales para la aplicación

Pruebas de Interfaz Gráfica y Navegación

- **Navegación Pestaña de Inicio** La prueba consiste en garantizar buena visibilidad de todos los elementos de la interfaz y poder trasladarnos, siguiendo la ejecución normal del programa y su uso previsto, a la pestaña de ejecución o de perfiles. La paleta elegida de colores presentes, permite una visibilidad de los componentes aceptable y suficiente para la visualización de todos los objetos importantes de la pantalla. Como podemos observar podemos elegir un perfil y pulsar el botón de avanzar para movernos a la pestaña de ejecución, o podemos seleccionar la opción de crear un nuevo perfil y ser redirigidos a la pestaña de perfiles.

Figura 2.9: Pestaña de inicio del programa

- **Navegación Pestaña de Perfiles** Pulsando el botón de crear nuevo perfil, somos redirigidos a la pestaña de perfiles. Aqui realizaremos distintas pruebas para confirmar la funcionalidad de agregar componentes a la base de datos y crear nuevos perfiles.
- Empezando por introducir una nueva máquina virtual

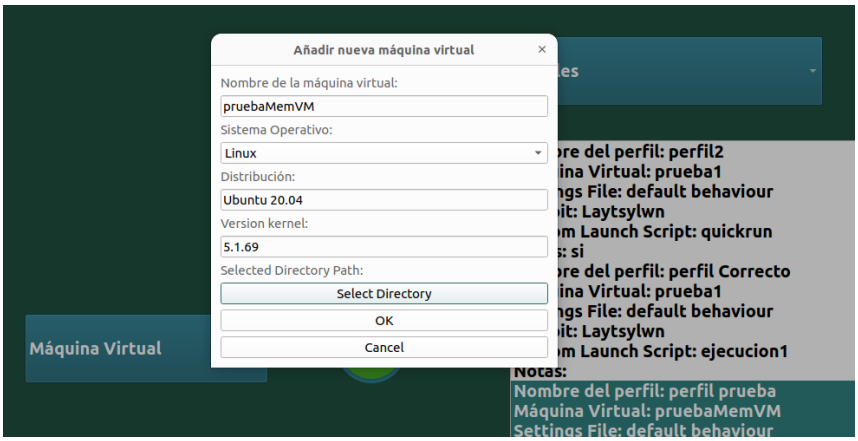


Figura 2.10: Introducimos los detalles para la máquina virtual

- Finalmente pulsamos el botón de aceptar, y seleccionamos para mostrar las entradas en la tabla de máquinas virtuales en la base de datos.

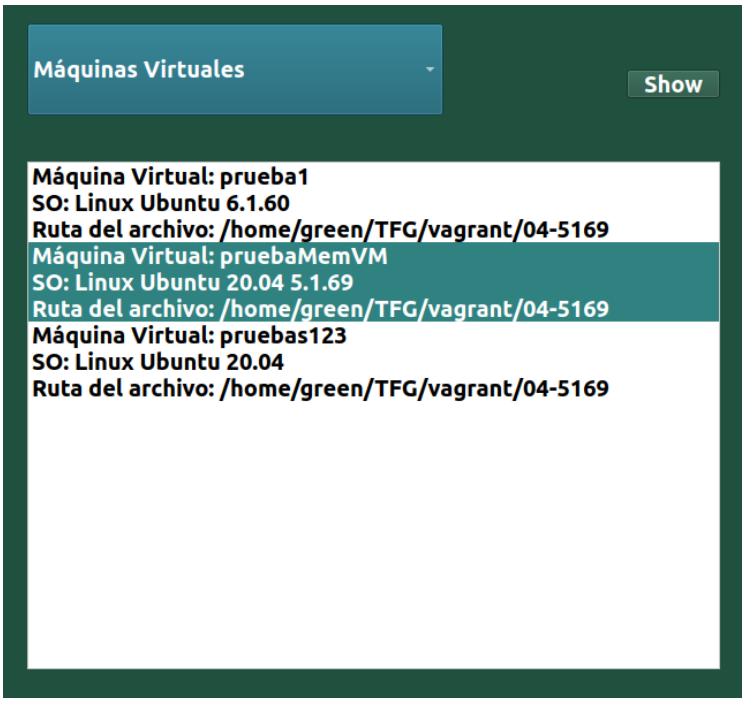


Figura 2.11: Comprobamos que se ha introducido correctamente en la tabla de máquinas virtuales

- Introducimos un nuevo archivo de ejecución

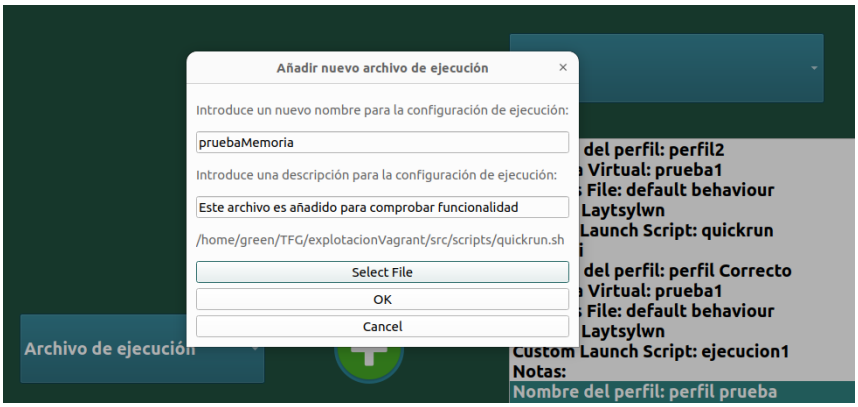


Figura 2.12: Introducimos los detalles para la máquina virtual

- Finalmente pulsamos el botón de aceptar, y seleccionamos para mostrar las entradas en la tabla de archivos de ejecución en la base de datos.

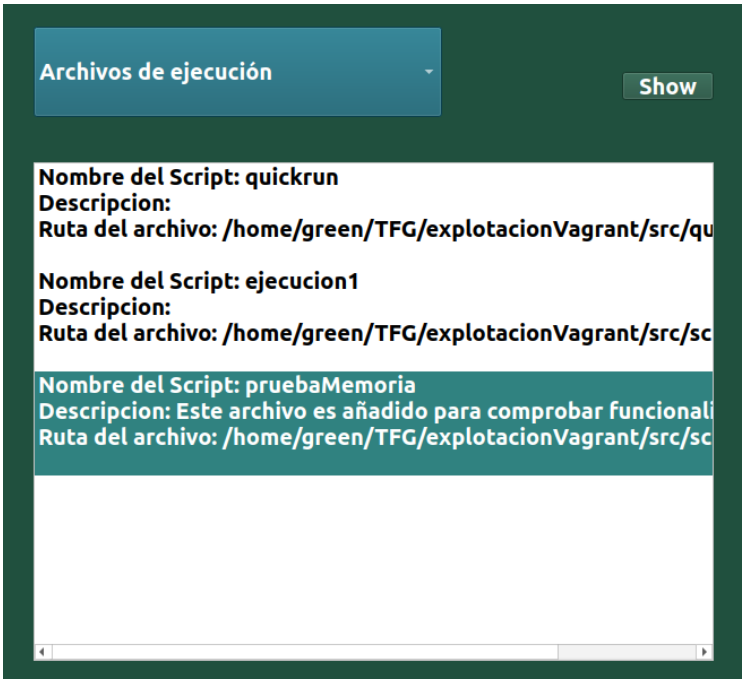


Figura 2.13: Comprobamos que se ha introducido correctamente en la tabla de máquinas virtuales

- Introducimos los datos para un nuevo CVE

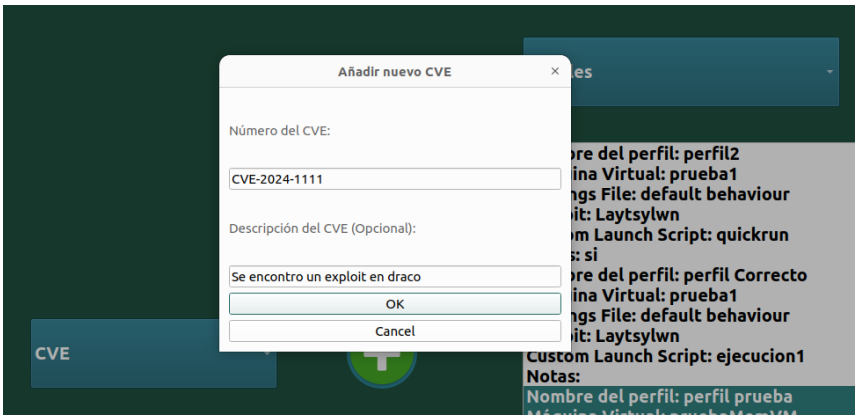


Figura 2.14: Introducimos los detalles para la máquina virtual

- Finalmente pulsamos el botón de aceptar, y seleccionamos para mostrar las entradas en la tabla de CVEs en la base de datos.

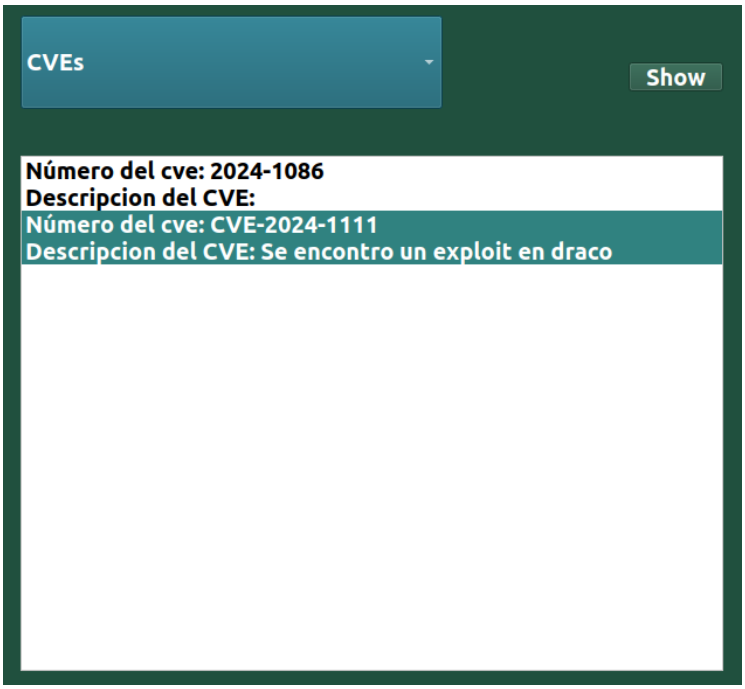


Figura 2.15: Comprobamos que se ha introducido correctamente en la tabla de máquinas virtuales

- Ahora vamos a introducir un nuevo exploit a la base de datos y deseamos que tenga como CVE relacionado el que acabamos de crear.

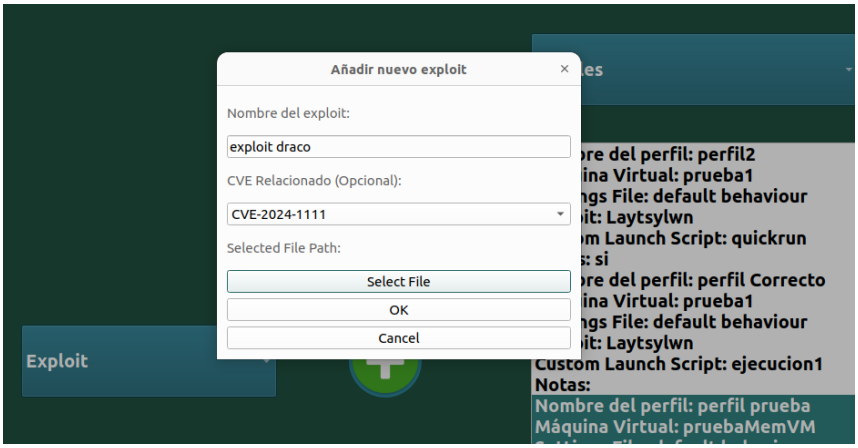


Figura 2.16: Introducimos los detalles para crear el nuevo exploit

- Como se puede apreciar, aparece el número del CVE que acabamos de crear y procederemos a añadirlo a la base de datos.

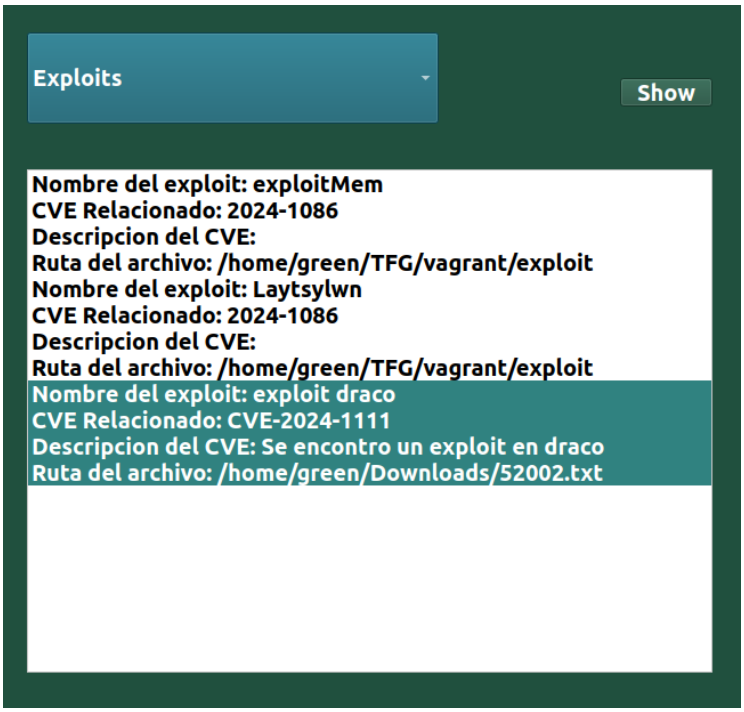


Figura 2.17: Comprobamos que se ha introducido correctamente en la tabla de exploits y apreciamos que aparecen los detalles del mencionado CVE

- Finalmente crearemos un nuevo perfil con todos los elementos nuevos que hemos añadido

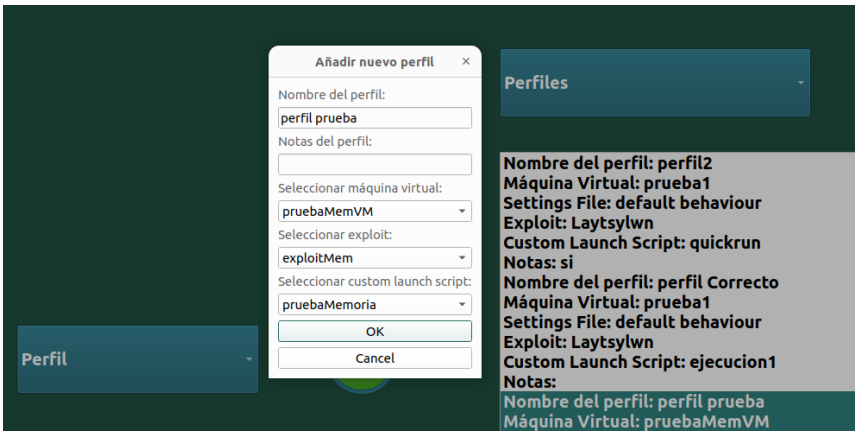


Figura 2.18: Seleccionamos los componentes que acabamos de crear

- Finalmente pulsamos el botón de aceptar, y seleccionamos para mostrar las entradas de los perfiles en la base de datos.

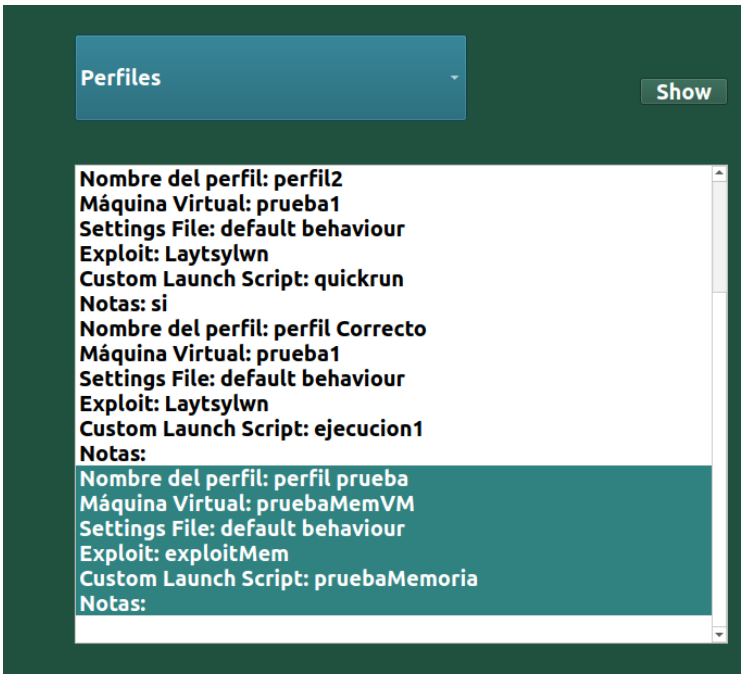


Figura 2.19: Podemos observar como el perfil se ha creado correctamente

- **Navegación Pestaña de Ejecución**
- Antes de comenzar vamos a cambiar la memoria asignada que tiene la máquina virtual y el numero de cores del procesador.

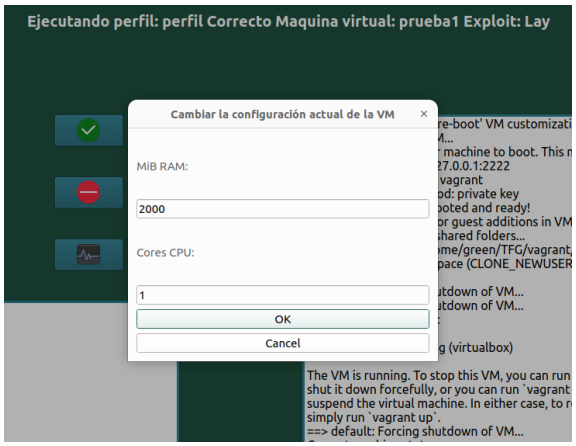


Figura 2.20: Aquí podemos observar los ajustes con los que se estaba ejecutando el perfil

- Cambiamos desde aqui los valores a los deseados y pulsamos aceptar

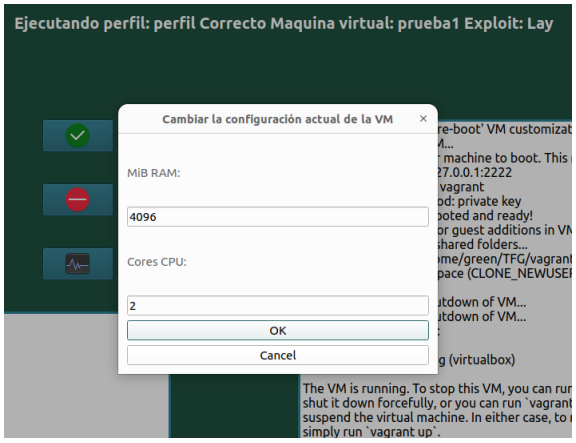


Figura 2.21: Introduciendo los valores deseados

- Procedemos a realizar una conexión ssh a la máquina virtual para comprobar el cambio.
  Ejecutando los comandos `nproc` para comprobar la cantidad de cores CPU destinados a esta máquina virtual y el mandato `free` para observar toda la memoria asignada

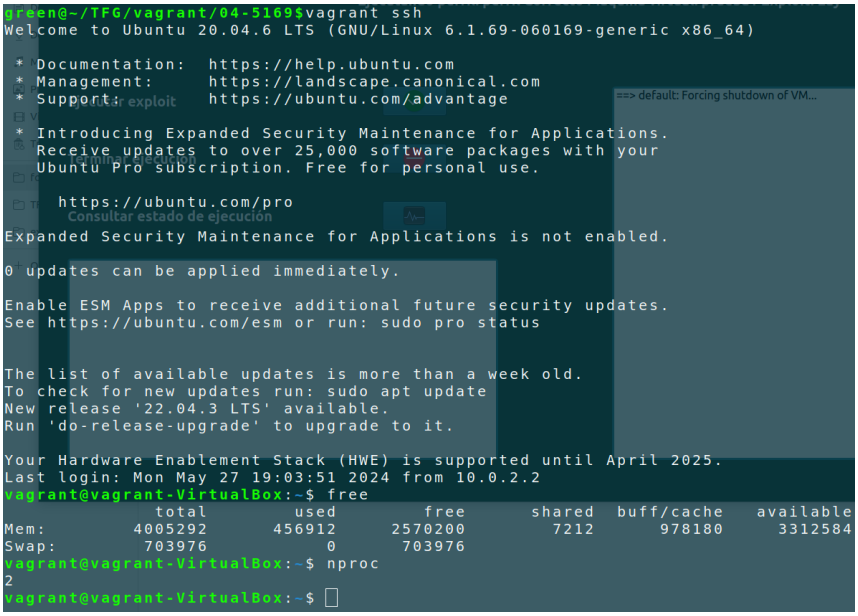


Figura 2.22: Aqui podemos observar los resultados del cambio de ajustes

- Para comenzar la ejecución pulsamos el botón de ejecutar el exploit, como podemos ver, la etiqueta de arriba de la pantalla nos muestra el perfil que está actualmente seleccionado.

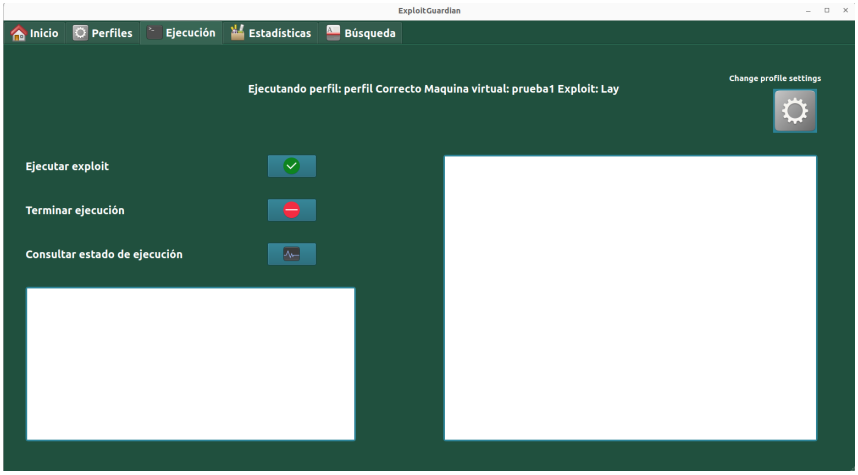


Figura 2.23: Pantalla de ejecución con el perfil seleccionado

- Finalmente, tras ejecutar podemos observar el output que aparece en la terminal de la máquina virtual y el tiempo que se ha tardado en alcanzar el flag.

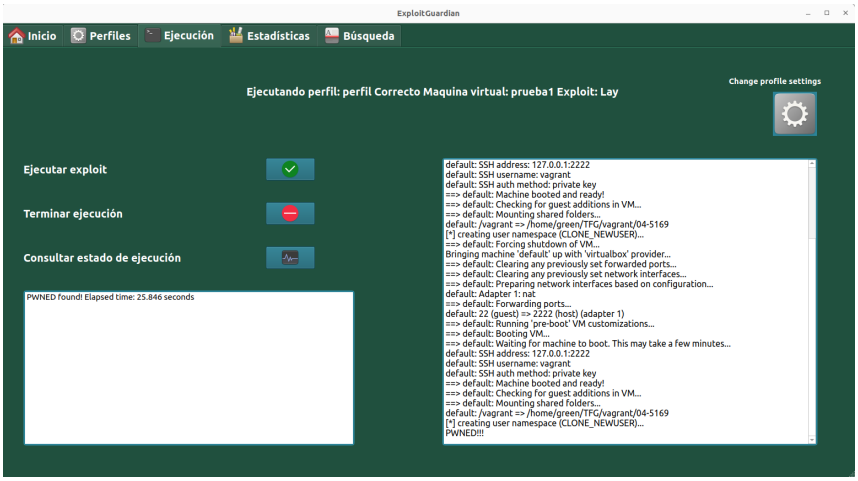


Figura 2.24: Aqui podemos observar los resultados de la ejecución

- **Navegación Pestaña de Estadísticas**
- Comenzamos seleccionando desde el combo box el perfil que deseamos visualizar. Al principio se nos mostrará un gráfico vacío.

Figura 2.25: Pestaña de estadísticas con el gráfico vacío

- Finalmente, tras pulsar el botón de mostrar, se puede visualizar el porcentaje de éxito del perfil y su total de ejecuciones.

Figura 2.26: Aqui podemos observar los resultados de todas las ejecuciones

- **Navegación Pestaña de Búsqueda**
- Como podemos observar, al principio no aparece ningún tipo de información por pantalla, ya que no hemos realizado ninguna consulta todavía. Estas consultas requieren de acceso a internet, ya que se realizan a través de la página oficial de CVE

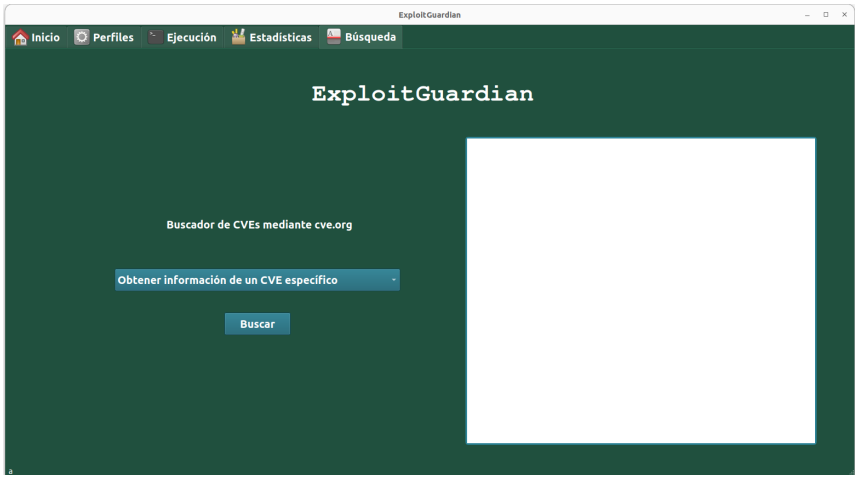


Figura 2.27: Pestaña de búesquedas vacio

- Tras introducir el ID del CVE del que queremos obtener información, encontraremos mostrada en pantalla los resultados de nuestra búsqueda.

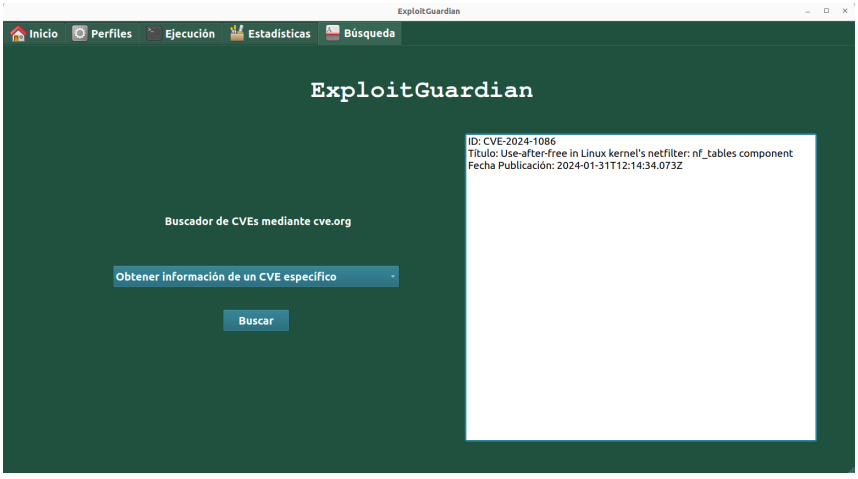


Figura 2.28: Aqui podemos observar los resultados de todas las búsquedas realizadas

Pruebas de ejecución

CVE-2024-1086

Este CVE se resume en una vulnerabilidad de `use-after-free`, es decir, usar después de liberar. Esto significa que en un programa se ha liberado memoria, pero, el atacante aún es capaz de utilizar el puntero que apuntaba a ese espacio de memoria liberado, ya que el puntero en sí no ha sido liberado. Además, siendo este exploit bastante más reciente, usando técnicas bastante interesantes y permitiendo obtener un control total del sistema víctima, resulta bastante interesante de estudiar. Este exploit tiene una serie de requerimientos que cumplimentar para ser posible su uso, uno de los cuales es el uso de una serie de kernels que no tengan parcheada esta vulnerabilidad como `6.1.69` que es el utilizado en esta prueba. El autor no lo menciona, pero si se utiliza en Ubuntu 22.04 el exploit fallará, por lo que se recomienda utilizar Ubuntu 20.04.

Como exploit utilizaremos este POC (`proof of concept`), que utiliza innovadoras técnicas para realizar un escalado de privilegios sin credenciales. Lo único necesario para este exploit es el archivo binario que está compilado en el repositorio del autor, aunque también existe la posibilidad de ejecutarlo sin binario como el mismo describe. Manipulando el sistema de direcciones del kernel de Linux, resulta útil virtualizar con más memoria RAM para asegurar el éxito del exploit. Como ya se ha mencionado, las técnicas de este exploit son bastante avanzadas y están todas recogidas en el blogspot[8] del autor. Para utilizar este exploit en nuestro programa se ha diseñado este script de lanzado rápido.

```
#!/bin/bash


cd /home/green/TFG/vagrant/04-5169 || exit

vagrant up
vagrant ssh -c 'echo cat /etc/flag.txt | ./exploit; exit'
```

Figura 2.29: Script de lanzado rápido para el CVE-2024-1086

Como se puede apreciar, primero se cambia al directorio en el que se encuentran los componentes Vagrant, después se enciende la máquina y se establece una conexión `SSH`. Ya que a partir de ese momento seguir escribiendo comandos no los ejecutaría en la máquina virtual, le pasamos el flag `-c` al comando `SSH` para que ejecute el exploit y de seguido lea el flag en `/etc/` y estando el flag protegida contra lectura sin permisos de root. Finalmente, si el resultado del comando `cat` contiene `PWNED` significará que la ejecución ha sido un éxito.

Probaremos este exploit utilizando 2000 MiB de RAM para evaluar hasta qué cantidad de memoria puede resultar viable la ejecución.

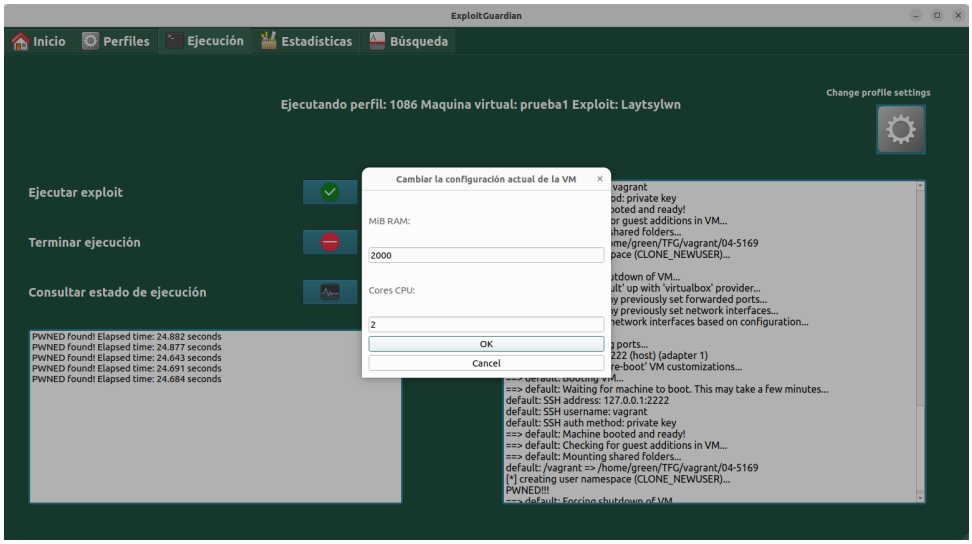


Figura 2.30: Configuración de los primeros cinco tests

Realizando una batería de cinco tests obtenemos éxito en todos ellos.

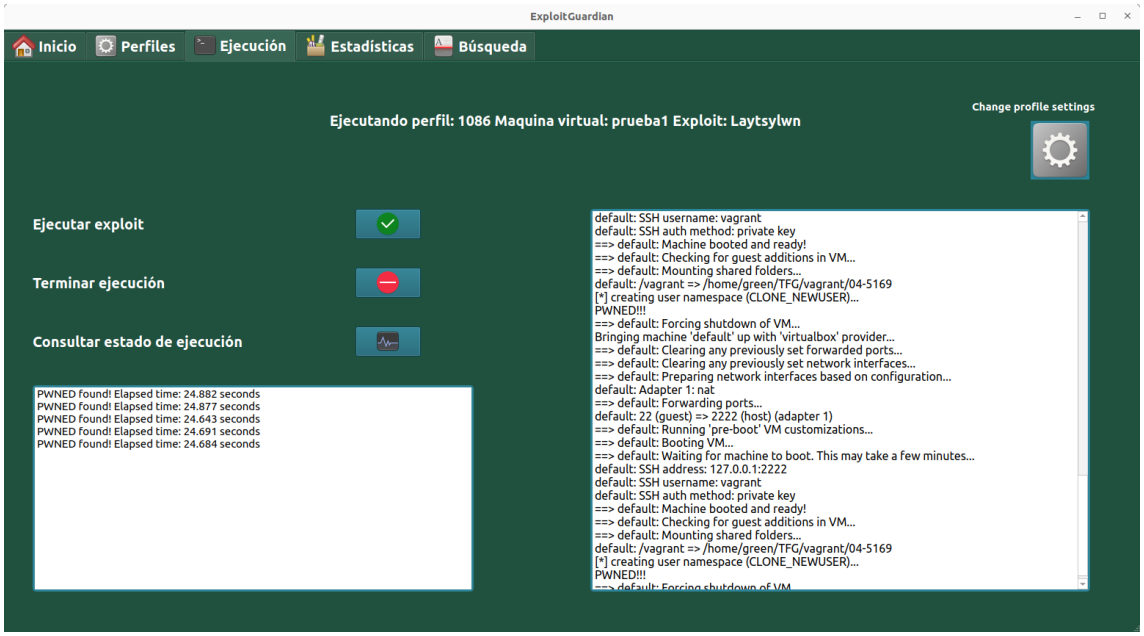


Figura 2.31: Los resultados de las 5 primeras ejecuciones

Probando con 2000 MiB de RAM, prueba ser suficiente para resultar exitosos cinco de cinco intentos. Aunque no podamos extrapolar mucho de una muestra de cinco pruebas, sí que podemos afirmar que 2000 MiB es una cantidad de memoria aceptable para este exploit.

Probando ahora con una cantidad mucho menor de RAM, se espera obtener una tasa de éxito bastante inferior a la obtenida con 2000 MiB de RAM.

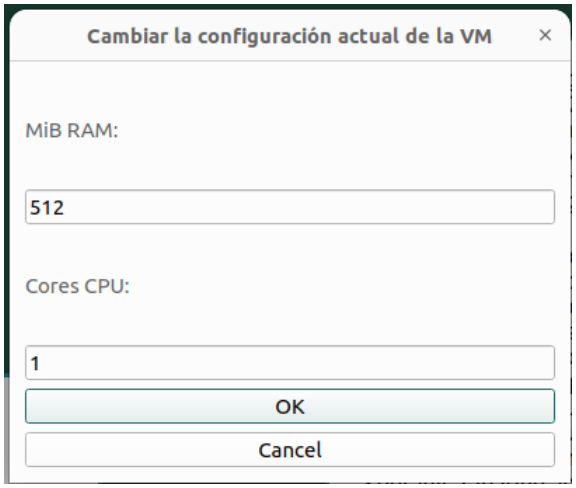


Figura 2.32: Configuración con 512 MiB de RAM

512 MiB de RAM es una cantidad muy pequeña para cualquier tipo de prueba y mucho más si es dependiente de RAM, al ser un caso extremo se espera obtener resultados bastante extremos también. Se realizarán otras cinco pruebas con esta nueva configuración.

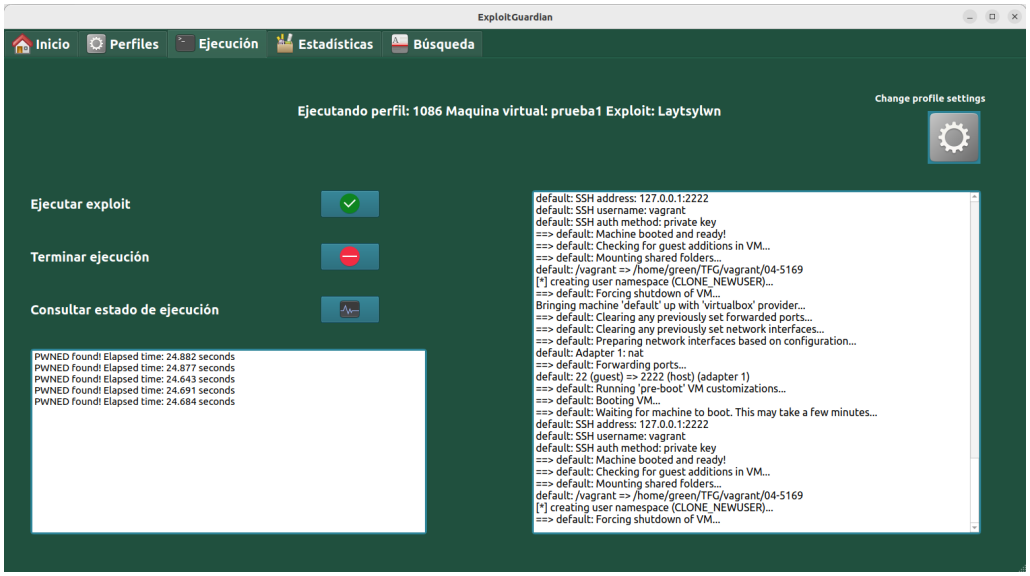


Figura 2.33: Resultados de las ejecuciones

Como podemos observar, en la pantalla de éxitos tan solo aparecen los de las ejecuciones anteriores cuando se utilizaba la configuración de mayor memoria RAM. Al quedarse la virtualización sin memoria antes de terminar el exploit el sistema se queda congelado y es necesario parar la máquina virtual, por lo que no se registra ningún intento exitoso.

Observando estos resultados, puede resultar interesante probar también a realizar esta ejecución, con una cantidad considerable de memoria RAM y con todas las facilidades para obtener éxito, pero aunque sigamos manteniendo la versión de kernel `6.1.69`, en una versión de Ubuntu más actualizada, en este caso Ubuntu 22.04.

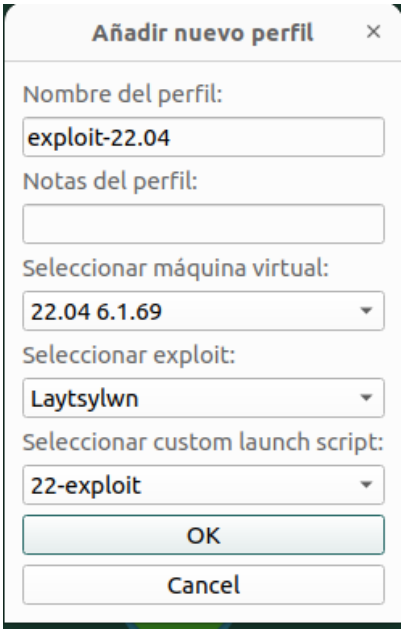


Figura 2.34: Opciones para el nuevo perfil

Creando el perfil adecuado para esta prueba, las condiciones son las mismas que en las anteriores ejecuciones excepto por la versión de Ubuntu. Además, como ya se ha mencionado, se configura el perfil para tener una cantidad de recursos físicos más que suficientes para garantizar éxito en la prueba.

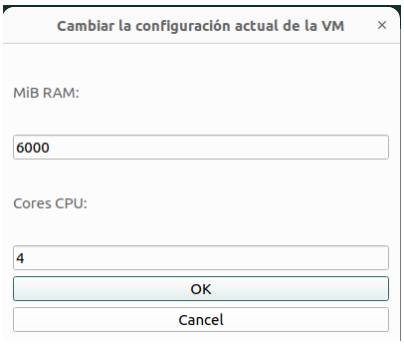


Figura 2.35: 6000 MiB RAM

Estando el perfil seleccionado y todos los detalles preparados, se procede a ejecutar la prueba.

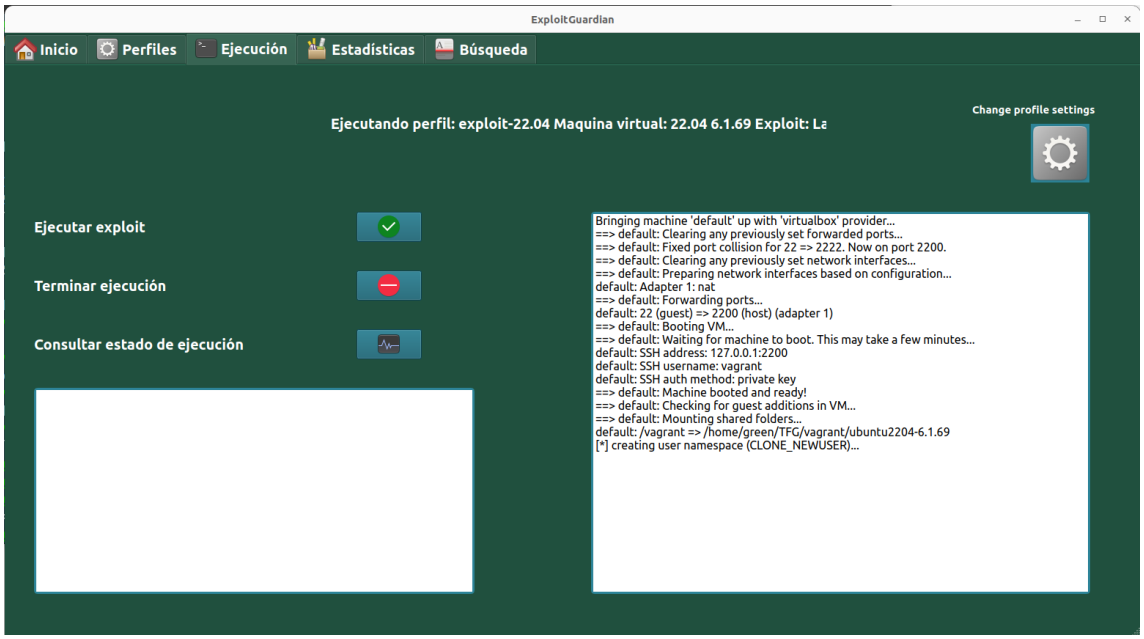


Figura 2.36: La ejecución queda bloqueada

Como podemos observar, el exploit llega al punto crítico de su ejecución en su manejo de la memoria, pero por mucho tiempo que se espere, la ejecución está paralizada significando esto que la ejecución ha sido un fracaso. Esto se debe a que esta nueva versión de Ubuntu, aun utilizando un kernel vulnerable, no esta afectada por este exploit.

CVE-2023-0386

Este CVE se resume en una vulnerabilidad que aprovecha un defecto en el subsistema de archivos OverlayFS, presente en el kernel de Linux. Esta vulnerabilidad permite el acceso no autorizado (**root access**) si se ejecuta un binario que tiene el `setuid` de `root`. Este problema surge cuando un usuario copia un archivo con capacidades de un montaje nosuid a otro montaje. Este bug de mapeo de `UID` permite a un usuario local escalar sus privilegios en el sistema. Los requerimientos son, haber compilado el binario y cambiado su dueño y `UID` a `root`. La versión del kernel de Linux utilizada es la que aparece en la página oficial de CVE, es decir, `6.2-rc6`.

Como exploit utilizaremos este POC (`proof of concept`), obtenido de esta página web[9] donde también se explica más a fondo este exploit.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>

int main(void) {
    int result;

    result = setuid(0);
    if (result != 0 ) {
      printf("could not setuid(0): %s\n", strerror(errno));
      return result;
    }

    result = setgid(0);
    if (result != 0) {
      printf("could not setguid(0): %s\n", strerror(errno));
      return result;
    }

    printf("Starting root shell...\n");
    system("/bin/bash");
    return 0;
}
```

Figura 2.37: Código en C para la explotación

Para utilizar este exploit en nuestro programa se ha diseñado este script de lanzado rápido.

Como podemos observar, es completamente idéntico al script del anterior CVE. Aun así, se ha incluido este paso de introducir un archivo de lanzado rápido personalizado, ya que cada CVE puede tener sus propias particularidades y, aunque en estos dos casos el modo de lanzado sea igual de sencillo, no será así en todos los casos.

```
#!/bin/bash


cd /home/green/TFG/vagrant/ubuntu2023-0386 || exit

vagrant up
vagrant ssh -c 'echo cat /etc/flag | ./setuid; exit'
```

Figura 2.38: Script de lanzado rápido para el CVE-2023-0386

Probaremos este exploit utilizando la configuración por defecto que son 1024 MiB de RAM para evaluar el tiempo medio de ejecución de este exploit en esta máquina virtual en concreto y si cambiar el número de núcleos CPU o la cantidad de memoria RAM afecta en el tiempo necesario para la ejecución.

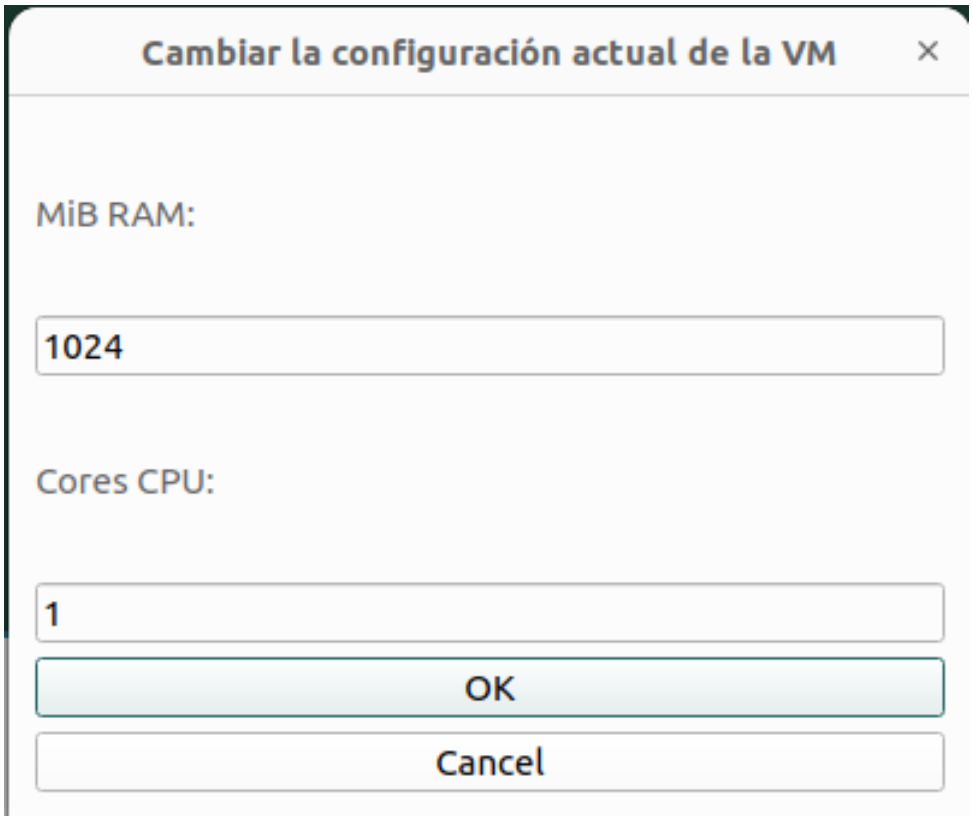


Figura 2.39: Configuración para los primeros cuatro tests

Realizando una batería de cuatro tests obtenemos éxito en todos ellos y un tiempo de alrededor de 39 segundos.

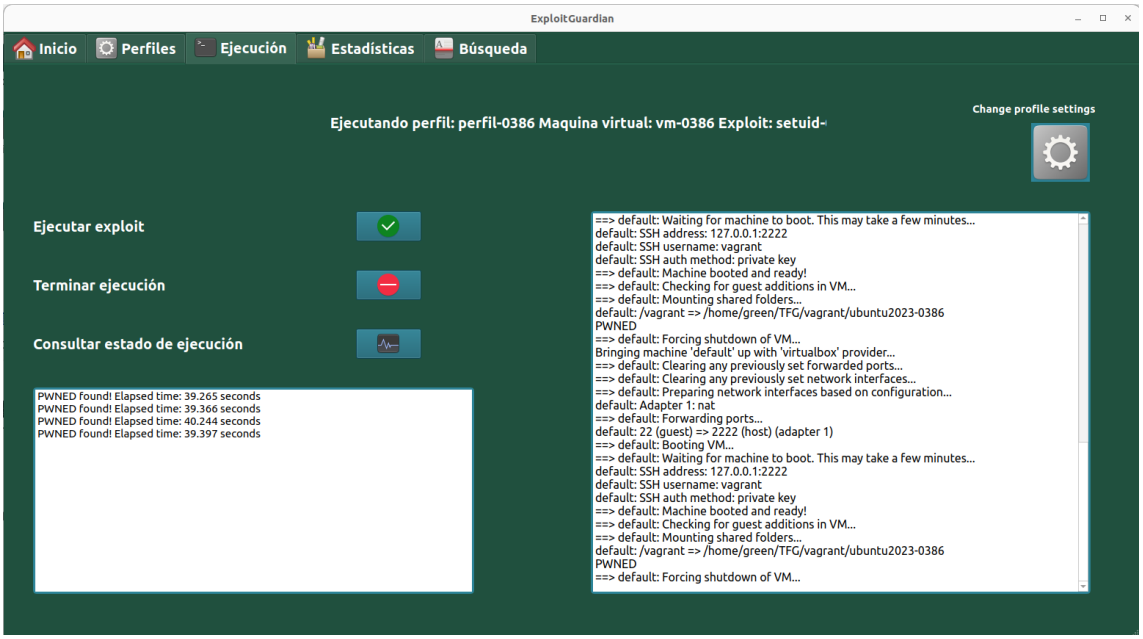


Figura 2.40: Los resultados de las 4 primeras ejecuciones

Esto era de esperar, ya que este exploit es muy estable y no tiene ningún requisito especial a la hora de ejecutarse.

Probando con 4096 MiB de RAM se espera que todos los exploits sigan siendo exitosos, por lo que solo se completará una ejecución más, pero resulta una incógnita si el tiempo hasta alcanzar el flag se verá afectado.

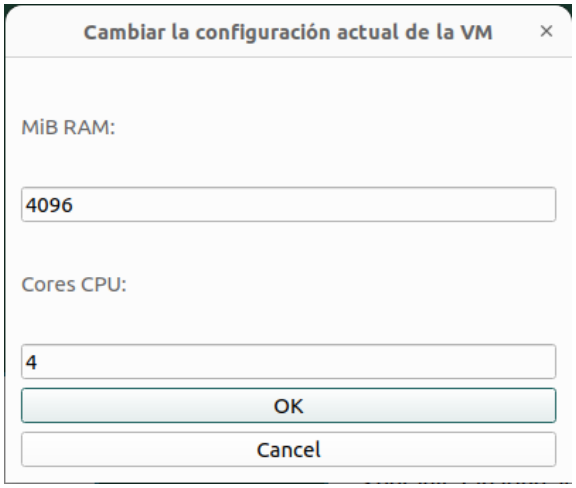


Figura 2.41: Configuración con 4096 MiB de RAM y cuatro núcleos

Realizamos el cambio en la configuración con el que, ya que la cantidad de recursos asignada a la virtualización es bastante mayor, se espera mayor rapidez en la ejecución.

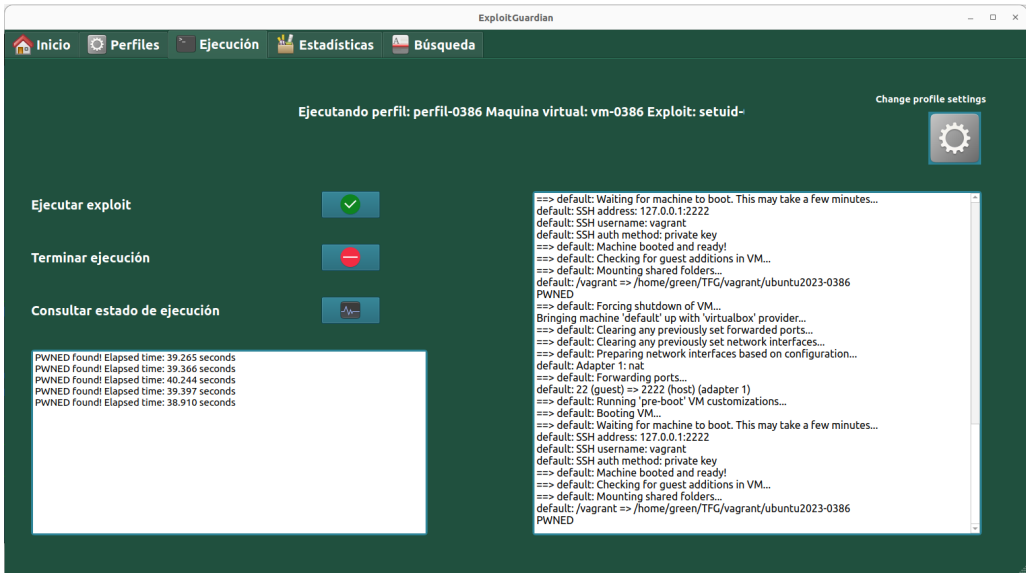


Figura 2.42: Resultados de la última ejecución

Como podemos observar, en la pantalla de éxitos la nueva ejecución no presenta ningún tipo de mejora razonable en comparación con haber cuadriplicado su memoria asignada y número de núcleos CPU así que se intuye que no hay espacio de mejora en la eficiencia, por lo menos, a través de cambios en la Vagrantfile.

Se procede a repetir la misma prueba que en el anterior exploit, en este caso, lo único especificado son las versiones del kernel vulnerables, entre las cuales se encuentra `6.2-rc6` por lo que procederemos a probar también a realizar esta ejecución, en una versión de Ubuntu más actualizada, en este caso también Ubuntu 22.04.

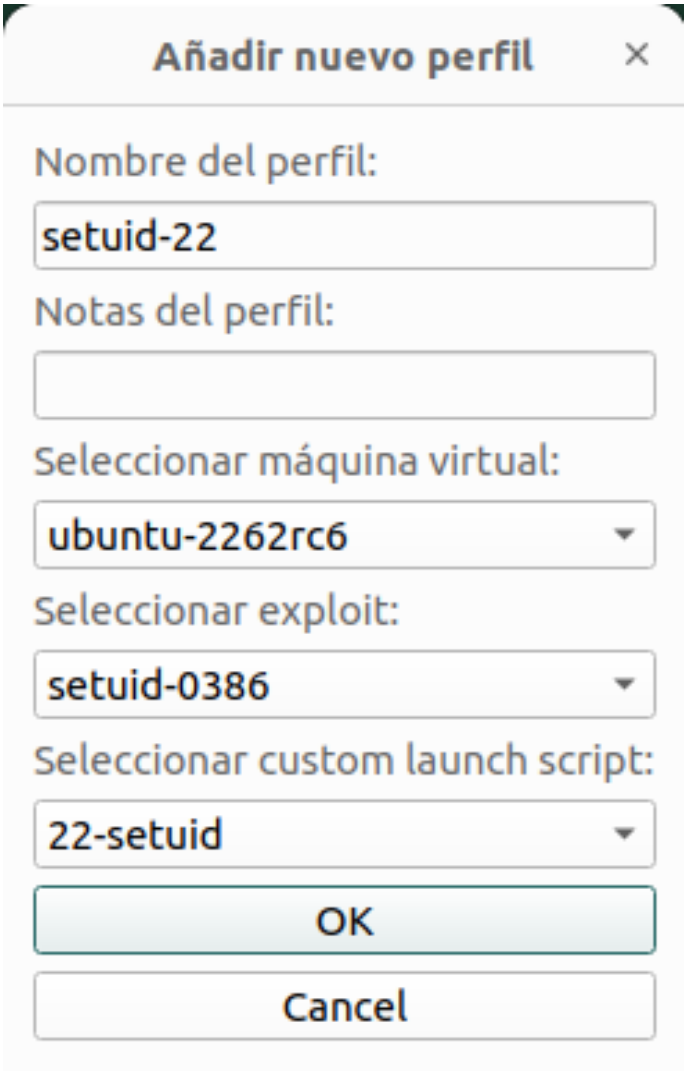


Figura 2.43: Opciones para el nuevo perfil

Creando el perfil adecuado para esta prueba, las condiciones son las mismas que en las anteriores ejecuciones excepto por la versión de Ubuntu.

Estando el perfil seleccionado y todos los detalles preparados, se procede a ejecutar la prueba.

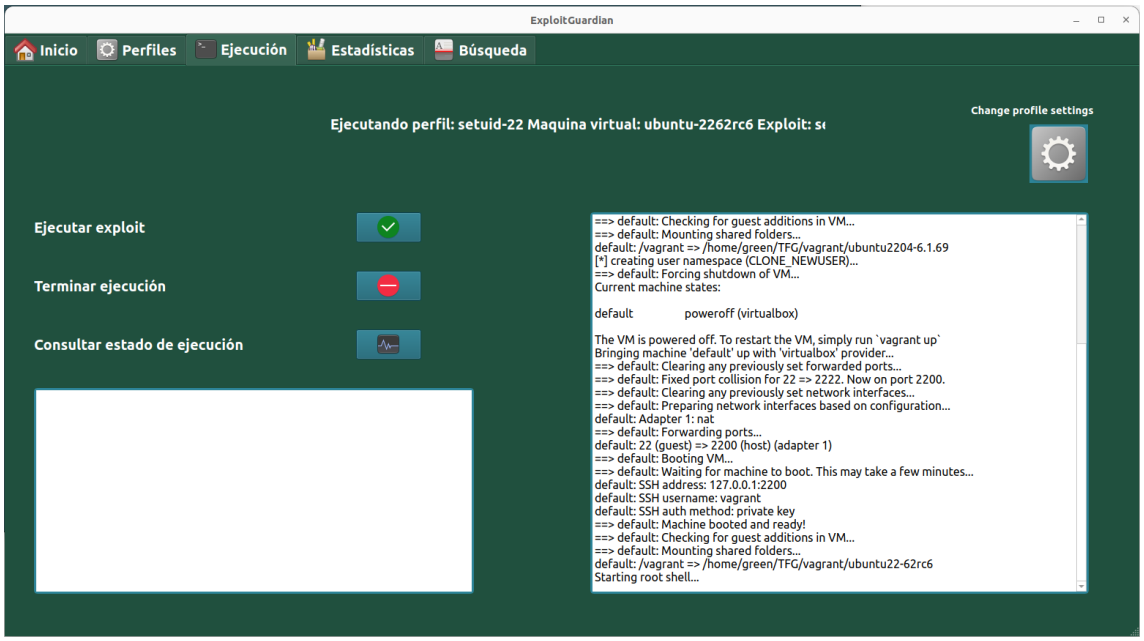


Figura 2.44: La ejecución queda bloqueada

Como podemos observar, el exploit llega al intentar crear el `root shell`, pero también queda congelado en ese punto, es decir, la ejecución ha sido un fracaso. Ocurriendo que aunque la versión del kernel utilizado en tanto este CVE como en el anterior sean vulnerables al exploit, pero actualizando la versión de Ubuntu al dejar de funcionar podemos deducir que los desarrolladores de Ubuntu introducen también parches retrocompatibles para kernels antiguos para asegurar la integridad de estos sistemas que requieren de usar kernels en específico. Esta deducción quizás no resulta la más emocionante, pero sirve de ejemplo de que clase de información se podría obtener utilizando un sistema de pruebas como el aquí presentado.

Capítulo 3

Análisis de impacto

3.1. Impacto a Nivel Personal

Posibles Beneficios:

- **Desarrollo de habilidades:** Individuos interesados por el campo de la informática y, específicamente por el campo de la ciberseguridad, pueden verse beneficiados por el uso de esta herramienta que, de relativa sencillez a la hora de uso, puede servir como una herramienta para una introducción a este mundo en las personas que este campo despierte curiosidad. Sirviendo como campo de pruebas y análisis para aspirantes a profesionales en ciberseguridad, puede ayudar a entender mejor los fenómenos de las vulnerabilidades informáticas y exploits. Evidentemente, esta herramienta puede jugar un gran papel no solo a la hora de formar nuevos profesionales sino también a la hora de facilitar el trabajo de profesionales haciendo labores de análisis mucho más sencilla
- **Conciencia de seguridad:** El poder presenciar las vulnerabilidades y exploits pueden resultar en una concienciación acerca de lo peligrosos y comunes que son estos fenómenos, ayudando a que el público general tome más conciencia de estos problemas y tome pasos extra para protegerse contra ellos.

Posifbles Efectos Adversos:

- **Fines no éticos:** Al igual que un individuo u organización tiene la posibilidad de usar una herramienta asi para fines educativos y con objetivos que no perjudiquen a ninguna parte, también puede haber individuos u organizaciones que, con fines ilícitos o perjudiciales, usasen esta herramienta para desarrollar sus propios vectores de ataque con los cuales realmente se podría causar algún daño. Este problema corresponde al ya existente de *White Hat Hackers vs Black Hat Hackers*, es decir que ya existe este fenómeno en el contexto de la ciberseguridad en el que la mayoría de los actores utilizan estas herramientas e investigan con fines positivos y lícitos (*White Hat Hackers*) pero existe un pequeño porcentaje que se dedica al beneficio

de manera ilícita e inmoral a través de estos exploits que podrían usar esta herramienta para fines no éticos. Aunque esta posibilidad existe, no hay que olvidar que el porcentaje de individuos que pueden beneficiarse de esta herramienta para fines no autorizados, es minúscula y el bien que una herramienta así puede brindar sobrepasa con creces estas desventajas.

3.2. Impacto a Nivel Empresarial

Posibles Beneficios:

- **Mejora de la seguridad informática:**
  - **Prevención Proactiva:** La herramienta permite realizar todo tipo de pruebas de penetración dentro de un entorno seguro y controlado, lo que ayuda a las empresas a adoptar un enfoque proactivo en lugar de reactivo frente a las vulnerabilidades. Es decir, les brinda a las empresas la capacidad de identificar y corregir fallos antes de que sean explotados maliciosamente.
  - **Personalización de Seguridad:** Las empresas pueden configurar la herramienta para realizar simulaciones de ataques personalizados o escenarios de amenazas particulares, lo que les permite fortalecer sus defensas contra los ataques dirigidos que sean más relevantes para su situación en específico.
- **Reducción de costos:**
  - **Menores Costos de Recuperación:** Al detectar problemas de seguridad antes de que causen daños, las empresas pueden evitar los costosos procesos de recuperación y las pérdidas de datos que suelen acompañar a las brechas de seguridad.
  - **Optimización de Recursos:** La herramienta ayuda a optimizar los recursos de seguridad al permitir que los equipos de seguridad se concentren en áreas de mayor riesgo identificadas durante las pruebas, en lugar de dispersar esfuerzos en menos áreas críticas.
  - **Cumplimiento Regulatorio:** Al mantener una postura de seguridad robusta y demostrable, las empresas pueden evitar multas y sanciones asociadas con el incumplimiento de normativas de protección de datos.

Posibles Efectos Adversos:

- **Riesgo de Mal Uso Interno:** Existe la posibilidad de que empleados con acceso a la herramienta puedan usarla para propósitos no autorizados, como obtener acceso no consentido a información sensible o sistemas críticos. Este riesgo se magnifica si no hay controles adecuados sobre quién puede usar la herramienta y cómo.
- **Explotación Externa:** Si la herramienta no está adecuadamente asegurada, podría ser explotada por agentes externos que logren acceder a ella.

Esto podría permitir a un atacante potenciar las capacidades de la herramienta para realizar ataques contra la propia empresa o terceros.

3.3. Impacto a Nivel Económico

Posibles beneficios

1. **Prevención de Ransomware:**
   - **Evitando Costos de Rescate:** Los ataques de ransomware, donde los datos críticos son secuestrados y se exige un rescate para su liberación, pueden costar a las empresas millones en pagos de rescate. La herramienta permite identificar y mitigar las vulnerabilidades que los ransomware podrían explotar, reduciendo significativamente la posibilidad de estos ataques.
   - **Mantenimiento de la Operatividad:** Al evitar ataques de ransomware, las empresas pueden garantizar la continuidad operacional sin interrupciones, evitando costos indirectos significativos asociados con el tiempo de inactividad del sistema y la pérdida de productividad.
2. **Minimización del Impacto de Brechas de Seguridad:**
   - **Reducción de Costos Legales y de Cumplimiento:** Las brechas de seguridad pueden llevar a costosas batallas legales y sanciones por incumplimiento de regulaciones de protección de datos como GDPR[10] en la Unión Europea. Al asegurar proactivamente los sistemas, la herramienta disminuye la probabilidad de tales eventos.
   - **Protección Contra la Pérdida de Datos Sensibles:** La pérdida o el robo de información sensible puede resultar en graves consecuencias financieras, desde la pérdida directa de propiedad intelectual hasta costos asociados con la notificación de brechas y medidas de reparación para los afectados.
3. **Costos de Implementación Versus Ahorros a Largo Plazo**
   - **Análisis Costo-Beneficio:** Aunque la implementación de la herramienta requiere una inversión inicial en software, hardware y capacitación, el retorno sobre la inversión puede ser rápido dado los altos costos asociados con los incidentes de ciberseguridad. Estudios indican que el costo promedio de una brecha de datos puede superar los millones, dependiendo de la escala y el alcance del ataque. En comparación, la inversión en seguridad proactiva es significativamente menor.

Efectos Adversos:

- **Inversión inicial alta**: El desarrollo y mantenimiento de la herramienta requiere una inversión significativa, lo que podría ser una barrera para pequeñas empresas.

Capítulo 4

Conclusiones y trabajo futuro

4.1. Conclusiones

El trabajo realizado demuestra que hay beneficios significativos al utilizar una herramienta como la descrita. Por ejemplo, la capacidad de ajustar rápidamente la cantidad de memoria RAM asignada es crucial, especialmente cuando se evalúa el exploit base utilizado en este proyecto. Este exploit en particular tiene una fuerte dependencia de la memoria del dispositivo para garantizar el éxito de su ejecución, por lo que la posibilidad de modificar estos valores de forma rápida es extremadamente beneficiosa. Esto no solo permite una verificación efectiva de la vulnerabilidad en diferentes condiciones de memoria, sino que también facilita la medición y comparación inmediata de los resultados.

Además, el uso de esta herramienta permite una iteración más rápida durante las pruebas, lo que reduce significativamente el tiempo necesario para obtener resultados concluyentes. La flexibilidad para ajustar los parámetros en tiempo real proporciona una ventaja considerable en el ámbito de la seguridad informática, donde el tiempo y la precisión son esenciales.

4.2. Trabajo Futuro

Mirando hacia el futuro, esta herramienta, aunque actualmente tiene un uso limitado, presenta un potencial a la hora de implementar más funcionalidades. La inclusión de nuevas funcionalidades, como la integración con otras herramientas de análisis de seguridad, podría ampliar significativamente su aplicabilidad. Además, mejorar la interfaz de usuario para facilitar aún más a los técnicos la configuración y el análisis de los resultados puede convertirla en un recurso indispensable para la evaluación de vulnerabilidades.

Un enfoque sencillo de implementar y de gran utilidad sería la automatización de pruebas. Implementar funcionalidades que permitan la ejecución de pruebas automatizadas basadas en parámetros predefinidos podría ahorrar considerablemente más tiempo y esfuerzo, permitiendo a los investigadores concentrarse en el análisis de los resultados más que en la configuración de las pruebas. Se

podría explorar la creación de plantillas personalizables que los usuarios puedan ajustar según sus necesidades específicas y que el programa ejecute por ejemplo, cien iteraciones de un perfil en específico.

La capacidad de generar informes detallados y análisis estadísticos avanzados sería otra adición valiosa. Estos informes podrían incluir métricas clave de rendimiento, como puede ser el consumo de memoria y cpu de la máquina virtual, que son esenciales para entender el comportamiento de los exploits y la efectividad de las medidas de mitigación de vulnerabilidades empleadas.

Además, incorporar algún tipo de integración con plataformas de comunidades de código abierto podría enriquecer el desarrollo de la herramienta, y de la información de la que dispone dicha aplicación.

Bibliografía

[1] D. Palmer. «Ataque sobre la NHS por Wannacry». (2018), dirección: https://www.zdnet.com/article/this-is-how-much-the-wannacry-ransomware-attack-cost-the-nhs/ (visitado 20-04-2024).

[2] K. Wood. «Ataque sobre Colonial Pipeline». (2023), dirección: https://www.law.georgetown.edu/environmental-law-review/blog/cybersecurity-policy-responses-to-the-colonial-pipeline-ransomware-attack/ (visitado 20-04-2024).

[3] «Página web de Python». (2024), dirección: https://www.python.org/.

[4] «Página web de la herramienta Vagrant». (2024), dirección: https://www.hashicorp.com/conferences/hashidays?product_intent=vagrant.

[5] «Página web de la librería Qt». (2024), dirección: https://www.qt.io/.

[6] «Página web de la base de datos PostgreSQL». (2024), dirección: https://www.postgresql.org/.

[7] «Página web de CVE». (2024), dirección: https://www.cve.org/.

[8] «Blog del autor del POC de CVE-2024-1086 con explicación detallada». (2024), dirección: https://pwning.tech/nftables/.

[9] «Página web de Security Labs acerca de CVE-2023-0396 con POC». (2023), dirección: https://securitylabs.datadoghq.com/articles/overlayfs-cve-2023-0386/.

[10] «Página web oficial de la UE sobre la GDPR». (2024), dirección: https://gdpr-info.eu/.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Mon Jun 03 11:16:33 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)