INFORME DEL PROYECTO

SCRIPT PARA REGLAS YARA

Propuesta creada por
Alejandro Delgado

El proyecto se basa en la realización de un script que automatice la recopilación de repositorios públicos y descargue y compile todas las reglas yara almacenados en cada uno de ellos para su posterior prueba contra malware y detección.

ÍNDICE

INTRODUCCIÓN

Detector de Malware Utilizando Reglas Yara

El mundo digital de hoy está repleto de amenazas, y una de las más serias es el malware, software diseñado para dañar o explotar cualquier sistema informático.

Frente a esta amenaza, este proyecto tiene como objetivo desarrollar un detector de malware estático. Utilizaremos una herramienta llamada Yara, conocida por su eficacia en identificar y clasificar malware. Este proyecto no solo es un desafío técnico, sino una contribución importante a la seguridad informática.

Nuestro enfoque será crear reglas Yara que puedan identificar muestras de malware con una precisión aceptable. Para lograr esto, haremos uso de repositorios de reglas Yara de código abierto y probaremos su efectividad contra distintas muestras de malware. Al final, esperamos tener un conjunto de reglas Yara compiladas y listas para detectar amenazas digitales.

ANTECEDENTES

¿Qué son las Reglas Yara?

Yara es una herramienta que actúa como un "detector de malware" en archivos y memoria. Lo interesante de Yara es que funciona mediante reglas que definen patrones característicos del malware.

Estas reglas pueden incluir cadenas de texto, combinaciones de bytes o incluso patrones de comportamiento. Cuando Yara escanea un archivo, utiliza estas reglas para ver si alguna coincide con los patrones del archivo, indicando la presencia de malware.

Historia y Desarrollo de Yara

Yara comenzó como una herramienta interna en empresas de seguridad cibernética pero rápidamente ganó popularidad en la comunidad de seguridad debido a su versatilidad y eficacia.

A lo largo de los años, Yara se ha desarrollado y mejorado, con contribuciones de una comunidad activa de expertos en seguridad.

Aplicaciones Actuales

Hoy en día, Yara es ampliamente utilizada en la industria de la ciberseguridad. Es una herramienta esencial para analistas de seguridad, investigadores de amenazas y equipos de respuesta a incidentes.

Su capacidad para adaptarse rápidamente a nuevas amenazas la hace invaluable en un campo donde los atacantes están constantemente innovando.

DESARROLLO DEL PROYECTO

Planteamiento Inicial

Con el contexto de las reglas Yara ya desarrollado, decidimos realizar un script de Python que, conforme vaya añadiendo repositorios de Github a través de sus enlaces, realice lo siguiente:

- Debe descargar el repositorio en formato zip.
- Descomprimir el repositorio para poder acceder a las carpetas.
- Acceder a las ubicaciones dónde se encuentren los archivos “.yar” o “.yara”.
- Copiar estos archivos directamente en una carpeta llamada “yara-rules”.
- Una vez estén preparados todos los repositorios y todas los archivos yara copiados, borrar las carpetas descargadas para limpiar el área de trabajo.
- Finalmente, compilar todas las reglas yara en un único archivo que utilizaremos para matchear los distintos malware analizados.

Repositorios seleccionados

Una vez tenemos esto, accedemos a repositorios públicos donde se han subido las reglas yara para poder trabajar con ellas en nuestro script. Los repositorios seleccionados serán los siguientes:

- https://codeload.github.com/kevoreilly/CAPEv2/zip/refs/heads/master
- https://github.com/Yara-Rules/rules/archive/refs/heads/master.zip
- https://github.com/Jistrokz/TheSweeper-Rules/archive/refs/heads/main.zip
- https://github.com/Neo23x0/signature-base/archive/refs/heads/master.zip
- https://github.com/rivitna/Malware/archive/refs/heads/main.zip
- https://github.com/ditekshen/detection/archive/refs/heads/master.zip
- https://github.com/The-DFIR-Report/Yara-Rules/archive/refs/heads/main.zip
- https://github.com/reversinglabs/reversinglabs-yara-rules/archive/refs/heads/develop.zip
- https://github.com/bartblaze/Yara-rules/archive/refs/heads/master.zip
- https://github.com/Xumeiquer/yara-forensics/archive/refs/heads/master.zip

- https://github.com/Hestat/lw-yara/archive/refs/heads/master.zip
- https://github.com/sbousseaden/YaraHunts/archive/refs/heads/master.zip
- https://github.com/kweatherman/yara4ida/archive/refs/heads/master.zip
- https://github.com/100DaysofYARA/2023/archive/refs/heads/main.zip
- https://github.com/kevthehermit/PasteHunter/archive/refs/heads/master.zip
- https://github.com/advanced-threat-research/Yara-Rules/archive/refs/heads/master.zip
- https://github.com/mikesxrs/Open-Source-YARA-rules/archive/refs/heads/master.zip
- https://github.com/CERT-Polska/mquery/archive/refs/heads/master.zip
- https://github.com/avast/yaramod/archive/refs/heads/master.zip
- https://github.com/SupportIntelligence/Icewater/archive/refs/heads/master.zip
- https://github.com/wgpsec/whohk/archive/refs/heads/master.zip
- https://github.com/1aN0rmus/Yara/archive/refs/heads/master.zip
- https://github.com/t4d/PhishingKit-Yara-Rules/archive/refs/heads/master.zip
- https://github.com/kevthehermit/YaraRules/archive/refs/heads/master.zip
- https://github.com/blacktop/docker-yara/archive/refs/heads/master.zip
- https://github.com/intezer/yara-rules/archive/refs/heads/master.zip
- https://github.com/godaddy/yara-rules/archive/refs/heads/master.zip
- https://github.com/fboldewin/YARA-rules/archive/refs/heads/master.zip
- https://github.com/tenable/yara-rules/archive/refs/heads/master.zip

Desarrollo del Script en Python

Ahora, toca desarrollar el script de python que nos permitirá automatizar la tarea de descargar reglas yara de una forma simple, sólo añadiendo la url del repositorio seleccionado, y compilarse en un único archivo que nos permita matchear con los distintos malware seleccionados.

Explicaré paso por paso lo realizado en el código para un correcto entendimiento del mismo y poder mejorarlo en un futuro si el proyecto resultara interesante.

1. Importación de Módulos

Primero, se importan los módulos necesarios: **requests** para realizar peticiones HTTP, **zipfile**, **os**, y **shutil** para manejar archivos y directorios, **glob** para buscar archivos con patrones, **yara** para compilar las reglas Yara, y **time** y **datetime** para el manejo de fechas y tiempos.

```
import requests, zipfile, os, shutil, glob, yara, time
from datetime import datetime
```

2. Definición de Funciones Auxiliares

Se definen varias funciones para facilitar tareas específicas:

- **get_current_date_formatted():** Devuelve la fecha actual en un formato específico.
- **create(folder):** Crea un directorio si no existe.
- **unzip_and_copy_yara_files(zip_file, extract_to, yara_dest_folder):** Extrae archivos .yara o .yar de un archivo ZIP y los mueve a un destino específico.
- **download(dst, path):** Descarga un archivo desde una URL dada y lo guarda en una ubicación específica.
- **check_and_compile_individual_yara_files(src_folder):** Revisa y compila individualmente archivos Yara, devolviendo una lista de archivos válidos.
- **compile_yara_rules(valid_files, compiled_file):** Compila todas las reglas Yara válidas en un solo archivo.

```
def get_current_date_formatted():
    return datetime.now().strftime("%d_%m_%Y")

def create(folder):
    if not os.path.exists(folder):
        os.mkdir(folder)
```

```
def unzip_and_copy_yara_files(zip_file, extract_to, yara_dest_folder):
    with zipfile.ZipFile(zip_file, 'r') as zip_ref:
        zip_ref.extractall(extract_to)
    for root, dirs, files in os.walk(extract_to):
        for filename in files:
            if filename.endswith('.yara') or filename.endswith('.yar'):
                src_file = os.path.join(root, filename)
                dest_file = os.path.join(yara_dest_folder, filename)
                if os.path.exists(dest_file):
                    os.remove(dest_file)
                shutil.move(src_file, dest_file)

def download(dst, path):
    r = requests.get(path)
    with open(dst, 'wb') as f:
        f.write(r.content)

def check_and_compile_individual_yara_files(src_folder):
    valid_files = []
    for filename in glob.glob(src_folder + '/*.yar*', recursive=True):
        try:
            yara.compile(filepath=filename)
            valid_files.append(filename)
        except yara.SyntaxError as e:
            pass
    return valid_files

def compile_yara_rules(valid_files, compiled_file):
    filepaths = {os.path.basename(file).split('.')[0]: file for file in valid_files}
    print(f"Compilando {len(filepaths)} archivos YARA válidos...")
    try:
        rules = yara.compile(filepaths=filepaths)
        rules.save(compiled_file)
```

```
        if os.path.exists(compiled_file):
            print(f"Archivo compilado creado exitosamente en {compiled_file}")
            return len(filepaths)
        else:
            print("Error: No se pudo crear el archivo compilado.")
            return 0
    except yara.SyntaxError as e:
        print(f"Error en la compilación final: {e}")
        return 0
```

3. Configuración inicial y Creación de Directorios

El script comienza imprimiendo información inicial y estableciendo las rutas de trabajo. Crea directorios para almacenar las reglas Yara y el archivo compilado.

```
print('{:_<50}'.format(""))
print("\n{:^50}".format("--- REGLAS YARA ---"))
print('{:_<50}'.format(""))
print("\n{:^50}".format("Creado por Alejandro Delgado"))
print('{:_<50}'.format(""))
print("\n\nComenzando a ejecutar el script...")

start_time = time.time()
root = os.path.dirname(os.path.abspath(__file__))

current_date = get_current_date_formatted()
rules_folder = os.path.join(root, f"rules_{current_date}")
yara_rules_folder = os.path.join(rules_folder, "yara-rules")
compiled_rules_file = os.path.join(rules_folder, "compiled_yara_rules")

create(rules_folder)
create(yara_rules_folder)
```

4. Definición de Repositorios

Se define una lista de URLs (**repositories**) que corresponden a diferentes repositorios de reglas Yara en GitHub.

```
repositories = [
    'https://codeload.github.com/kevoreilly/CAPEv2/zip/refs/heads/master',
    'https://github.com/Yara-Rules/rules/archive/refs/heads/master.zip',
    'https://github.com/Jistrokz/TheSweeper-Rules/archive/refs/heads/main.zip',
    'https://github.com/Neo23x0/signature-base/archive/refs/heads/master.zip',
    'https://github.com/rivitna/Malware/archive/refs/heads/main.zip',
    'https://github.com/ditekshen/detection/archive/refs/heads/master.zip',
    'https://github.com/The-DFIR-Report/Yara-Rules/archive/refs/heads/main.zip',
    'https://github.com/reversinglabs/reversinglabs-yara-rules/archive/refs/heads/develop.zip',
    'https://github.com/bartblaze/Yara-rules/archive/refs/heads/master.zip',
    'https://github.com/Xumeiquer/yara-forensics/archive/refs/heads/master.zip',
    'https://github.com/Hestat/lw-yara/archive/refs/heads/master.zip',
    'https://github.com/sbousseaden/YaraHunts/archive/refs/heads/master.zip',
    'https://github.com/kweatherman/yara4ida/archive/refs/heads/master.zip',
    'https://github.com/100DaysofYARA/2023/archive/refs/heads/main.zip',
    'https://github.com/kevthehermit/PasteHunter/archive/refs/heads/master.zip',
    'https://github.com/advanced-threat-research/Yara-Rules/archive/refs/heads/master.zip',
    'https://github.com/mikesxrs/Open-Source-YARA-rules/archive/refs/heads/master.zip',
    'https://github.com/CERT-Polska/mquery/archive/refs/heads/master.zip',
    'https://github.com/avast/yaramod/archive/refs/heads/master.zip',
    'https://github.com/SupportIntelligence/Icewater/archive/refs/heads/master.zip',
    'https://github.com/wgpsec/whohk/archive/refs/heads/master.zip',
    'https://github.com/1aN0rmus/Yara/archive/refs/heads/master.zip',
    'https://github.com/t4d/PhishingKit-Yara-Rules/archive/refs/heads/master.zip',
    'https://github.com/kevthehermit/YaraRules/archive/refs/heads/master.zip',
    'https://github.com/blacktop/docker-yara/archive/refs/heads/master.zip',
    'https://github.com/intezer/yara-rules/archive/refs/heads/master.zip',
    'https://github.com/godaddy/yara-rules/archive/refs/heads/master.zip',
    'https://github.com/fboldewin/YARA-rules/archive/refs/heads/master.zip',
    'https://github.com/tenable/yara-rules/archive/refs/heads/master.zip'
]
```

5. Procesamiento de Repositorios

Para cada repositorio en la lista:

- Se determina el nombre del repositorio.
- Se descarga el archivo ZIP del repositorio.
- Se extraen los archivos Yara y se copian al directorio destinado para ellos.
- Se limpia el área de trabajo eliminando los archivos temporales y los directorios.

```
repo_count = 0

print("Procesando cada repositorio...")
for repo in repositories:
    repo_name = repo.split('/')[-1].split('.')[0]
    temp_folder = os.path.join(root, repo_name)

    zip_filename = os.path.join(root, f"{repo_name}.zip")
    download(zip_filename, repo)
    unzip_and_copy_yara_files(zip_filename, temp_folder, yara_rules_folder)
    shutil.rmtree(temp_folder, ignore_errors=True)
    os.remove(zip_filename)
    repo_count += 1
```

6. Compilación de Reglas Yara

Después de procesar todos los repositorios:

- Se revisan y compilan individualmente los archivos Yara para asegurar que son válidos.
- Se compilan todas las reglas Yara válidas en un solo archivo.
- Se imprime información sobre el proceso de compilación y el resultado.

```
valid_files = check_and_compile_individual_yara_files(yara_rules_folder)
compiled_count = compile_yara_rules(valid_files, compiled_rules_file)

end_time = time.time()
elapsed_time = end_time - start_time

print("\nSe ha ejecutado correctamente.")
print(f"Nº de repositorios analizados: {repo_count}")
print(f"Nº de reglas Yara compiladas: {compiled_count}")
print(f"Tiempo transcurrido: {elapsed_time:.2f} segundos.")
print("\nGracias por utilizar mi programa")
```

7. Finalización del Script

El script finaliza mostrando un resumen del proceso, incluyendo la cantidad de repositorios procesados, el número de reglas compiladas y el tiempo total transcurrido.

PROBLEMAS DURANTE LA IMPLEMENTACIÓN

Desafíos encontrados y solución

Durante el desarrollo del proyecto encontré diferentes problemas que me costaron resolver y que me han supuesto un desafío el saber interpretarlos y resolverlos con la idea de crear un script que pudiera mejorar la automatización de recopilación de reglas yara y su posterior compilación.

PROBLEMA. CARPETAS CON EL MISMO NOMBRE:

Al descargar tantos repositorios e intentar automatizar su descarga, me encontraba con el problema que al crear los distintos directorios se producción problemas ya que se les otorgaba el mismo nombre y fallaba el script.

SOLUCIÓN PROPUESTA:

Para evitar la auto asignación de nombres, lo que planteé fue crear carpetas temporales con los nombres de las carpetas de donde se estaban extrayendo los archivos e ir descargando ahí cada uno de los archivos que me fuera encontrando. Una vez descargados y enviados a su carpeta, estas carpetas serían borradas para limpiar el área de trabajo y evitar posibles colisiones a futuro en otras compilaciones.

PROBLEMA. FALLOS EN LA COMPILACIÓN DE ARCHIVOS:

Al descargar archivos de múltiples repositorios, algunos de ellos tenían problemas de sintaxis y me daba error de compilación, lo que detenía el script y no permitía su funcionamiento normal.

SOLUCIÓN PROPUESTA:

Para esto había dos vías: o arreglar a mano aquellos archivos que tuvieran este error o evitarlos en la compilación. Ya que la primera era totalmente inviable para un correcto desarrollo, tomé la segunda opción y evité estos archivos que producción estos errores a la hora de compilar.

Para ello, realizo un primer análisis en el que visualizo aquellos archivos que pueden generar problemas y los descarto para que, a la hora de compilar todas las reglas yara en un único archivo, no generen problemas.

PROBLEMA. DUPLICIDAD DE ARCHIVOS YARA:

Al descargar los archivos e intentar moverlos a la carpeta específicamente creada para contenerlos se duplicaban los archivos, es decir, estaban tanto en la carpeta creada para este motivo como en la carpeta raíz.

SOLUCIÓN PROPUESTA:

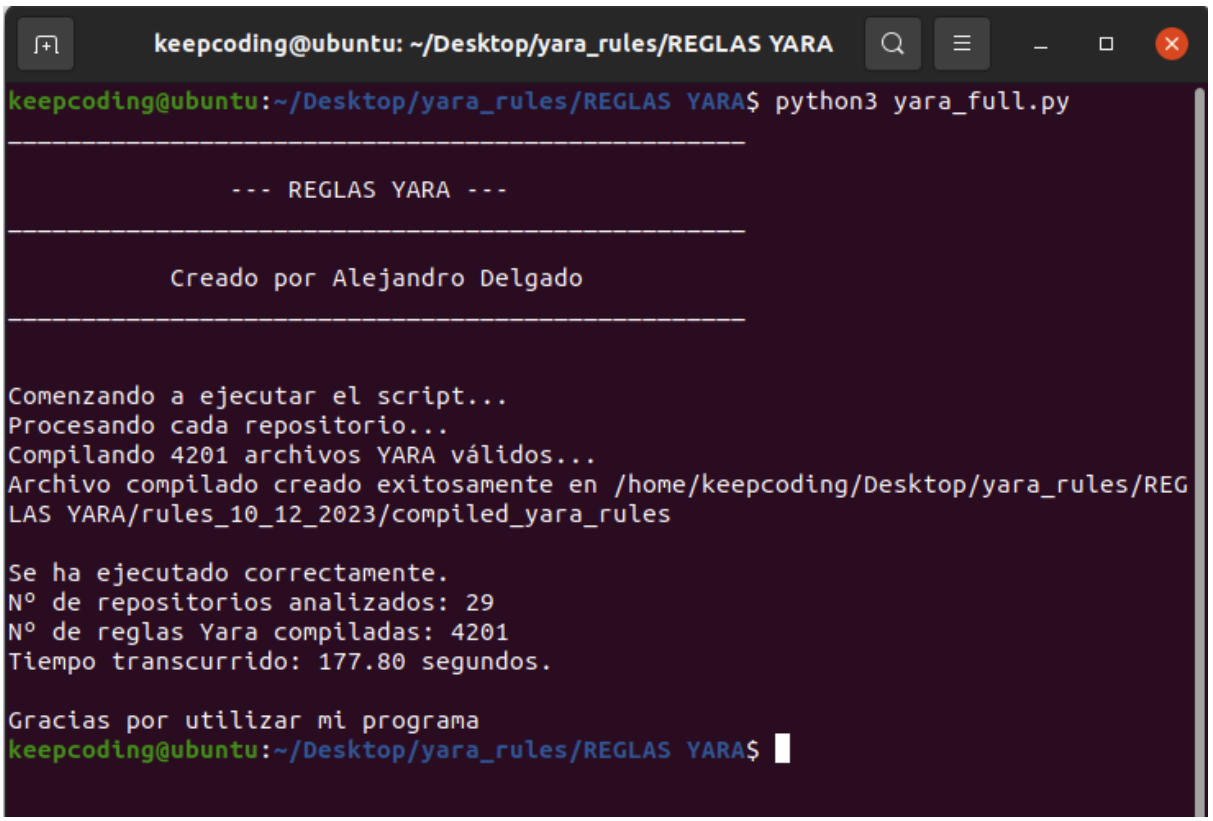
Para evitar este problema, cambié el orden de como se hacían las cosas para no descargar primero los archivos y posteriormente crear la carpeta y moverlos. En su lugar, cree en primera instancia la carpeta donde se iba a almacenar todos los archivos yara descargados y, después, identificaría esta carpeta y los descargaría directamente en ella, evitando la duplicidad de los archivos.

PRUEBAS Y RESULTADOS

Ejecutando el script de automatización de reglas yara.

Cuando ejecutamos el script funcionará de la siguiente forma:

- Se crea una carpeta llamada rules que contendrá la fecha de cuando se ha ejecutado el script para tenerlo como recordatorio.
- Procesará cada repositorio.
- Descargará los archivos yara en la carpeta indicada.
- Se realizará un primer análisis de los archivos yara descargados para evitar problemas en la posterior compilación de las mismas.
- Se compilarán todos los archivos YARA válidos.
- Se mostrará por la terminal la siguiente información:
    - Nº de repositorios analizados.
    - Nº de reglas Yara compiladas.
    - Tiempo transcurrido.



Pruebas con Malware

Se ha realizado la prueba con los siguientes malware:

- emotet.exe
- emotet_unpack.exe
- WannaCry.exe
- win.exe

Todos han sido satisfactorios y han matcheado con las reglas yara compiladas. A continuación, muestro capturas de pantalla con las pruebas ejecutadas:

```
keepcoding@ubuntu:~/Desktop/yara_rules/REGLAS YARA/rules_10_12_2023$ yara -C compiled_yara_rules emotet.exe
misc_upx_packed_binary emotet.exe
UPXv20MarkusLaszloReiser emotet.exe
UPXV200V290MarkusOberhumerLaszloMolnarJohnReiser emotet.exe
UPX290LZMAMarkusOberhumerLaszloMolnarJohnReiser emotet.exe
Failed_Checksum emotet.exe
_dUP_v2x_Patcher__wwwdiablo2oo2cjbnet_ emotet.exe
_UPX_290_LZMA_ emotet.exe
```

```
keepcoding@ubuntu:~/Desktop/yara_rules/REGLAS YARA/rules_10_12_2023$ yara -C compiled_yara_rules emotet_unpack.exe
XorExample1 emotet_unpack.exe
Failed_Checksum emotet_unpack.exe
_dUP_v2x_Patcher__wwwdiablo2oo2cjbnet_ emotet_unpack.exe
_MSVCpp_v8_procedure_1_recognized__h_ emotet_unpack.exe
_MASMTASM__sig4_h_ emotet_unpack.exe
_Microsoft_Visual_Cpp_80_ emotet_unpack.exe
_Microsoft_Visual_Cpp_60__80_ emotet_unpack.exe
_Microsoft_Visual_Cpp_8_ emotet_unpack.exe
_MASMTASM__sig1h_ emotet_unpack.exe
```

```
keepcoding@ubuntu:~/Desktop/yara_rules/REGLAS YARA/rules_10_12_2023$ yara -C compiled_yara_rules WannaCry.exe
warning: rule "RegExpExample1": too many matches for $re4, results for this rule may be incorrect
Win32_Ransomware_WannaCry WannaCry.exe
dox WannaCry.exe
Wanna_Cry_Ransomware_Generic WannaCry.exe
Failed_Checksum WannaCry.exe
_dUP_v2x_Patcher__wwwdiablo2oo2cjbnet_ WannaCry.exe
_Microsoft_Visual_Cpp_ WannaCry.exe
_Armadillo_v171_ WannaCry.exe
_Microsoft_Visual_Cpp_v60_ WannaCry.exe
```

```
keepcoding@ubuntu:~/Desktop/yara_rules/REGLAS YARA/rules_10_12_2023$ yara -C compiled_yara_rules win.exe
warning: rule "RegExpExample1": too many matches for $re4, results for this rule may be incorrect
warning: rule "domain": too many matches for $domain_regex, results for this rule may be incorrect
dox win.exe
Failed_Checksum win.exe
_dUP_v2x_Patcher__wwwdiablo2oo2cjbnet_ win.exe
_yodas_Protector_v1033_dllocx__Ashkbiz_Danehkar_h_ win.exe
_Possibly_PCX_graphics_format_ win.exe
_MPEG_Layer_IIIII_music_file_ win.exe
_Microsoft_Visual_Cpp_v60_ win.exe
```

Por lo tanto podemos decir que la ejecución del script ha sido satisfactorio y que posiblemente sea una buena herramienta que ayude en el análisis de malware a futuro, evitando mayores incidentes o repercusiones.

CONCLUSIÓN DEL PROYECTO

Al concluir este proyecto, nos encontramos con un sistema de detección de malware notablemente efectivo, apoyado en reglas Yara. La prueba exitosa de nuestro sistema con cuatro diferentes muestras de malware nos ha confirmado que funciona. Esta hazaña no sólo valida la calidad de las reglas compiladas, sino también la eficiencia de nuestro enfoque automatizado mediante un script en Python.

Aspectos Destacados

- **Efectividad Comprobada:** El éxito en la detección de todas las muestras de malware probadas es un testimonio de la eficacia de nuestro sistema.
- **Automatización y Eficiencia:** El proceso automatizado para recopilar y compilar reglas Yara aportó una gran eficiencia al proyecto.

Áreas para Mejoras Futuras

- **Expansión de la Base de Datos de Reglas:** Incorporar más fuentes para mejorar la cobertura y diversidad de las reglas.
- **Actualización Continua:** Mantener las reglas actualizadas frente al cambiante panorama de amenazas digitales.
- **Validación y Pruebas Ampliadas:** Realizar pruebas más extensas para asegurar la robustez del sistema en diversos escenarios.
- **Integración con Otras Herramientas:** Potenciar la utilidad del sistema mediante su integración con otras plataformas de seguridad.

Implicaciones en la Seguridad Informática

Este proyecto no solo demuestra un progreso significativo en la detección de malware, sino que también enfatiza la importancia de la colaboración y el intercambio en la comunidad de seguridad informática. Con este trabajo, contribuimos a fortalecer las defensas contra las amenazas digitales, resaltando el valor de soluciones innovadoras y colaborativas en el ámbito de la ciberseguridad.