MÁSTER EN *ANÁLISIS DE MALWARE Y REVERSING*

Campus Internacional CIBERSEGURIDAD
ENIIT INNOVA IT BUSINESS SCHOOL
UCAM Universidad Católica de Murcia

Tabla de contenido

1. APÉNDICE A: ENTORNO DE PRUEBAS

1.1. Advertencia

En primer lugar:

NUNCA, JAMÁS, SE MANIPULA MALWARE (o algo sospechoso de serlo) EN UN SISTEMA QUE NO SEA DESECHABLE

Utiliza siempre una máquina virtual y antes de que sea una sandbox con un control de red perfecto, esa máquina virtual debe estar desconectada de Internet.

Esa frase en rojo debes grabártela tanto aquí como estudiante como en el terreno profesional. Es una regla de oro que debes respetar siempre.

1.2. Máquina virtual

Usa cualquier hipervisor y cualquier sistema Linux (una versión actual, soportada) que te permita instalar los paquetes necesarios para hacer tanto las prácticas como las evaluaciones. Si te sientes cómodo con una distribución en particular, úsala.

1.3. Entorno Python

Las distribuciones Linux casi siempre vienen con una versión de Python exclusiva. Esto representa dos problemas:

- Puede que haya paquetes Python que no estén disponibles para esa versión en particular.

- Cuando instalas muchos paquetes, debido a las diferentes necesidades, es posible que des con una conflicto de versiones. Además, polucionas el entorno con cientos de paquetes y se puede volver inmanejable.

Existen numerosas soluciones para esto. De hecho hasta el conocido XKCD lo dejó reflejado en una gran viñeta.

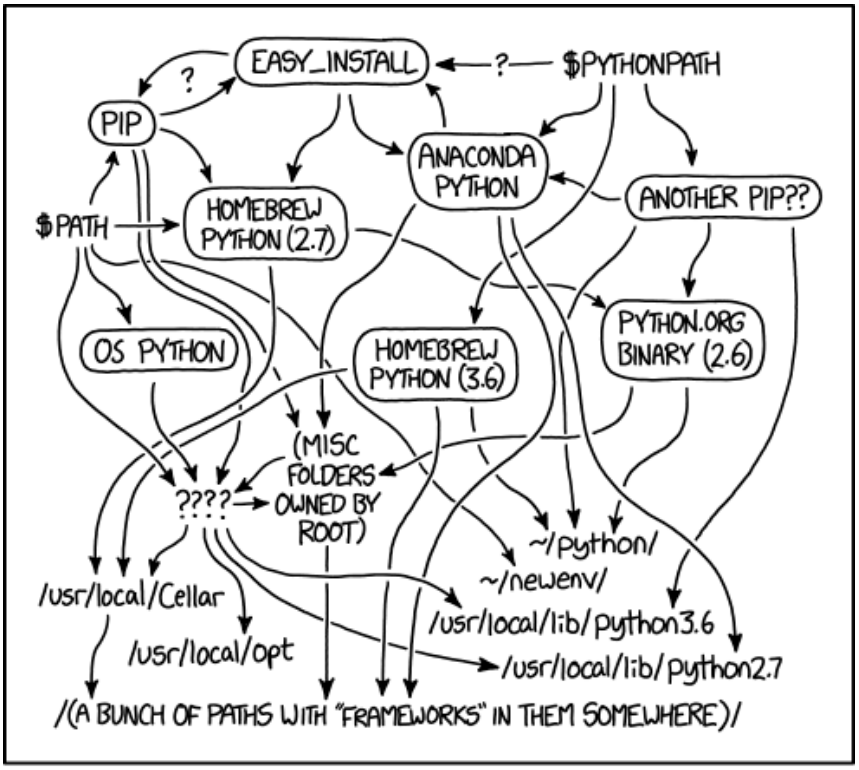


ILUSTRACIÓN 1 FUENTE: HTTPS://XKCD.COM/1987/

Voy a proponerte una solución que funciona bastante bien.

1.4. Pyenv

Pyenv es un gestor de "Pythons". Es decir, con esta herramienta consigues que en tu sistema operativo tengas todas las versiones de Python que quieras. Léete las instrucciones, merece la pena aprender cómo funciona la herramienta.

Una vez lo tengas instalado, ya puedes instalar cualquier versión de Python, un ejemplo:

$ pydev install 3.10

Eso nos instala esa versión de Python, sin influir en la que ya tenga el sistema.

En tu sesión de shell, si siguen invocando a Python verás que sigue siendo la versión del sistema y no la que acabas de instalar. Tiene sentido. El sistema necesita la versión que viene "de fábrica" para funcionar. Para usar la que tú has instalado:

pyenv shell <versión>

Eso haría que tu shell actual invoque al Python que instaló Pyenv.
También funciona con:

pyenv local <versión>

Esa es buena, porque activa ese Python en el directorio actual. Y aunque abras otra shell, si es en ese mismo directorio, se activará la misma versión. Muy útil para nuestros menesteres.

Desactivar un entorno activado por Pyenv es simple. Si estamos en un entorno activado (shell, directorio):

$ deactivate

1.1.1.1 PYENV VIRTUALENV

Pyenv nos instala diferentes versiones pero seguimos teniendo el problema de polucionar el entorno con miles de paquetes Python que podrían no tener sentido si lo tenemos mezclados y dan servicio a diferentes proyectos.

Para ello, tenemos los entornos virtuales de Python, que podemos dedicar a cada proyecto para no mezclar paquetes y tenerlo todo ordenado.

Pyenv posee un utilísimo plugin para el manejo de entornos virtuales. Disponible aquí.

Una vez instalado podemos crear todos los entornos virtuales que deseemos desde la versión Python instalada por Pyenv.

Por ejemplo, para las prácticas con YARA puedo tener un entorno virtual dedicado:

$ pyenv virtualenv <versión> yara-practicas

Si queremos activar el entorno:

$ pyenv virtualenv yara-practicas

Listo. Ya lo tenemos.

Es posible instalar las librerías Python en un solo entorno virtual. Si os diese problemas, os recomiendo un entorno virtual por cada bloque de prácticas.

Una vez tienes activado el entorno, cuando utilices un comando pip para instalar, no polucionarás otros entornos de Python, solo el entorno virtual que has creado.

Bonus

Si tienes tu proyecto en un directorio y tienes un entorno virtual Python instalado como hemos visto arriba, puedes hacer que Pyenv (si lo tienes integrado en el shell (ver sus instrucciones) detecte que esa carpeta pertenece a un directorio y una vez entres en él te active automágicamente el entorno.

Solo tienes que crear un archivo ".python-version" y dentro de él poner la cadena del nombre del entorno. En el caso de yara-practicas, sería esa cadena: "yara-practicas".

2. APÉNDICE B: ÍNDICE DE PRÁCTICAS VOLUNTARIAS

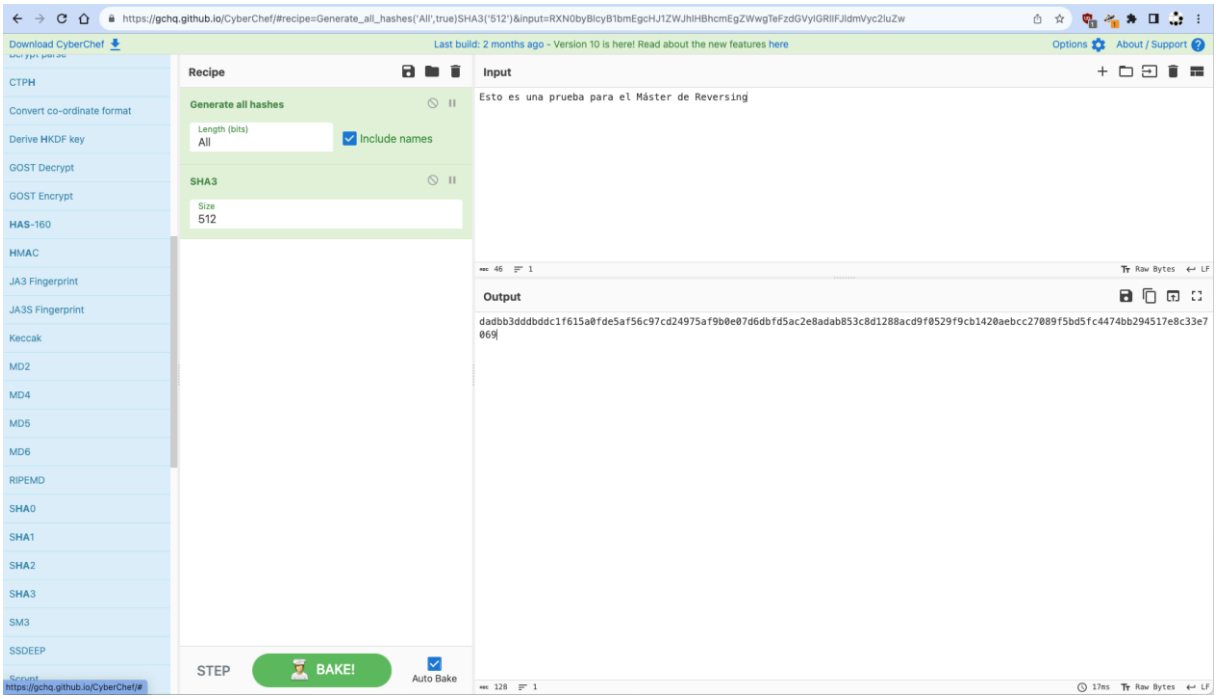
Aquí encontrarás todas las prácticas voluntarias por orden de aparición en el temario.

2.1. Hashes

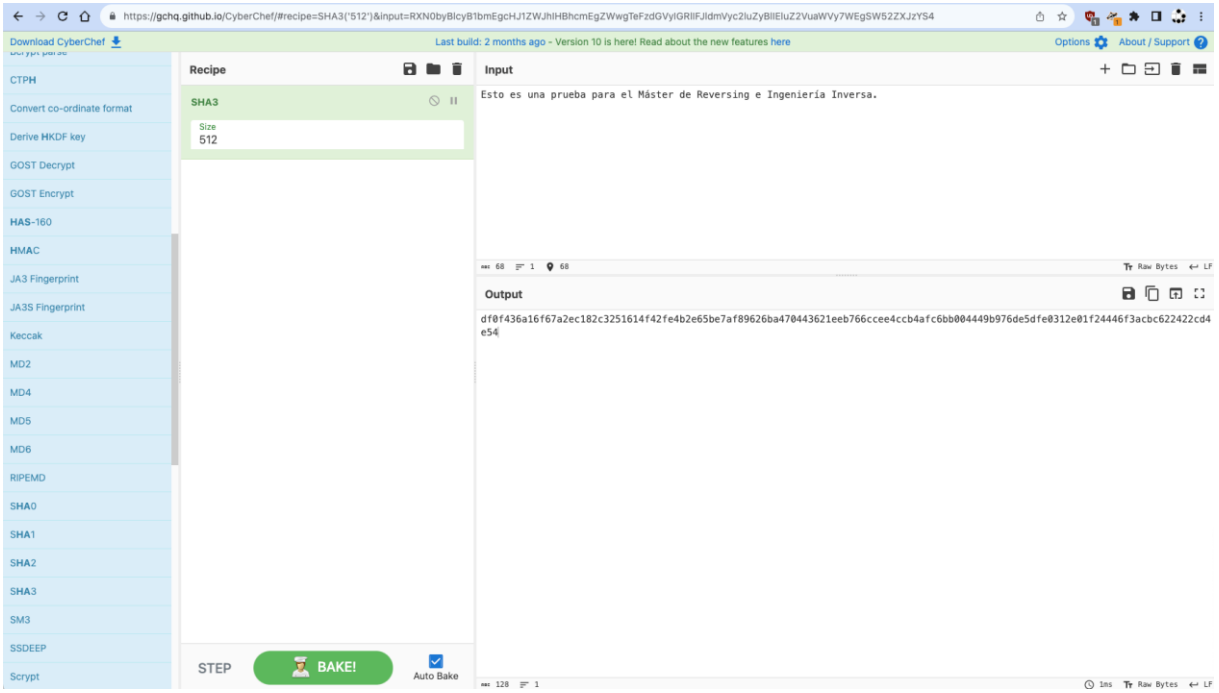
2.1.1. Observa como cambia el hash

Calcular el hash de un archivo es trivial. Tan solo se necesitan las herramientas adecuadas y, afortunadamente, están disponibles para cualquier sistema operativo e incluso en versión online.

Utilizando la herramienta online Cyberchef, pon cualquier texto y añade la receta “SHA3”. Observarás como genera el SHA3 del texto que hayas puesto:



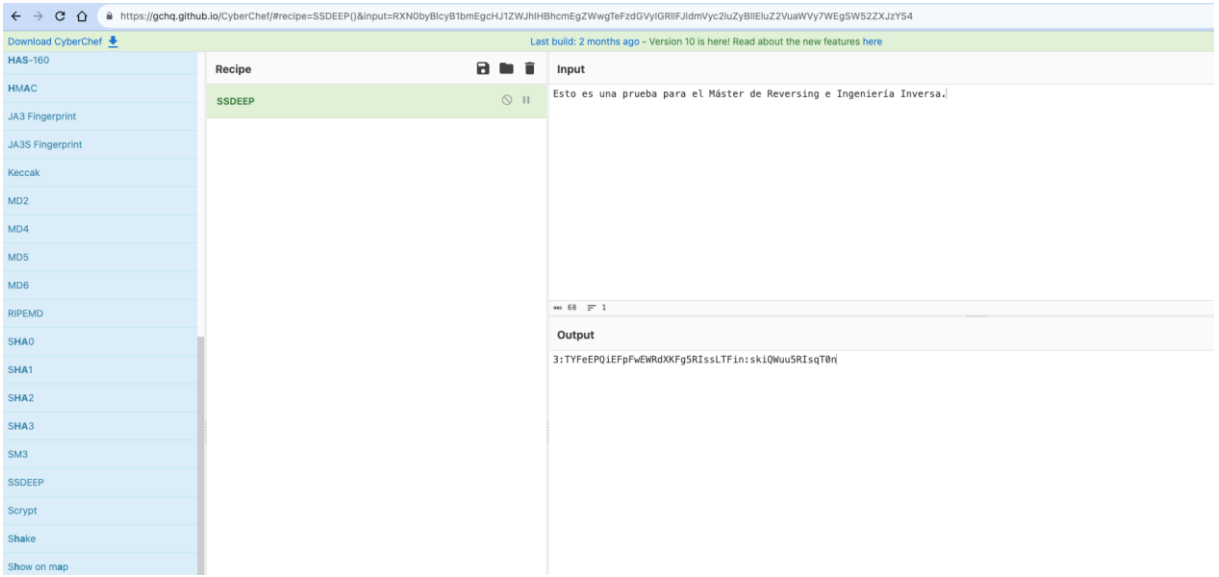
Ahora modifica ligeramente el texto de la entrada y observa como el hash ha cambiado por completo.



Quédate con esto: Una modificación, incluso mínima, puede hacer que el resultado del hash sea completamente diferente.

2.1.2. Hashes difusos

Aprovecha el entorno de pruebas de CyberChef y cambia la receta de SHA3 a SSDEEP. Introduce un texto de entrada y fíjate en el hash (anótalo, crea otra pestaña, etc.).



Cambia la entrada alterando algunas palabras y observa el resultado. Verás que los hashes son diferentes, pero no totalmente. Solo han cambiado algunas partes de este.

2.2. YARA

La utilidad de línea de comandos 'yara' está disponible en cualquier distribución Linux actual a fecha de revisión de este material. Consulta la utilidad de gestión de paquetes de tu distribución. No obstante, como ejemplo, en Debian o Ubuntu:

```
$ sudo apt install yara
```

Si quieres, en Python, puedes instalar el binding de YARA así:

```
$ pip install yara-python
```

Para instalar reglas YARA:

```
$ git clone https://github.com/Yara-Rules/rules
```

2.2.1. Escaneando ejecutables con 'yara'

En el apartado de sandboxes hay varios recursos para descargar malware. Utiliza una o varias muestras de ahí. Recuerda:

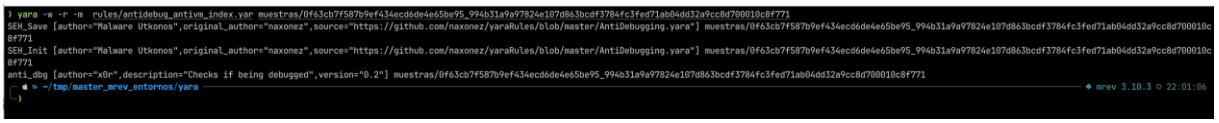
Ojo, SIEMPRE, TODO, USANDO MÁQUINAS VIRTUALES!!!!!

Jamás, nunca, lo hagas sobre tu sistema. Te infectarás sí o sí. Estás avisado.

Con 'yara' instalado (apt install yara) y las reglas libres disponibles en github (https://github.com/Yara-Rules/rules) vas a escanear las muestras que te has bajado.

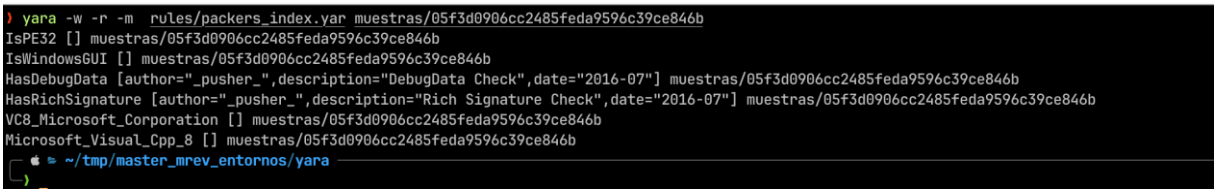
Fíjate que todo se hace en estático, sin ejecutar el malware.

Dentro de las reglas YARA tenemos un catálogo para detectar funcionalidad. Por ejemplo, vamos a pasar reglas YARA para funciones de detección de máquinas virtuales:



Bingo. Ahí tenemos unas pocas técnicas usadas por esa muestra para detectar entornos de análisis.

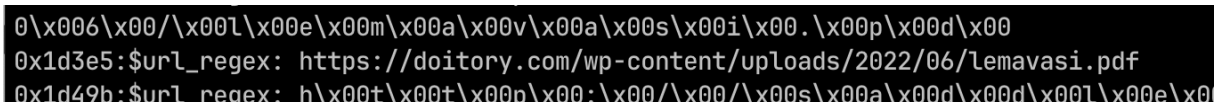
Ahora vamos a ver si posee alguna rutina packer:



No lo parece. Fíjate que las reglas nos hablan sobre qué tipo de ejecutable es y el compilador usado para crear la muestra. Algo útil cuando pasemos a un análisis manual.

Tenemos también algunas reglas misceláneas, por ejemplo esta sencilla de mi buen amigo Antonio Sánchez, que detecta patrones de uso de "http|s" dentro de binarios.

Si queremos ver la cadena que ha resultado detectada, podemos añadir la bandera "-s" y ser testigos de resultados muy curiosos:



Lo dicho, bájate unas muestras en una carpeta y lanza 'yara' con varios grupos de reglas para acostumbrarte a ver distintos resultados. Juega con las opciones de la línea de comandos.

2.2.2. Creando una regla YARA

El segundo ejercicio propuesto es, como no, la creación de una regla YARA básica.

Como va a depender mucho de las muestras que te hayas descargado, te reto a crear una regla básica que descarte las muestras dependiendo del tamaño y simplemente detecte si es un ejecutable Windows.

Para el tamaño es sencillo, de todas las muestras descargadas, lístalas por tamaño y haz la media.

Para el ejecutable Windows hay varias formas: módulos o "magia".

Fíjate en el orden por peso de mis muestras y la detección acorde:

2.3. Sandboxes

Para practicar con las sandboxes disponibles puedes usarlas directamente, no es necesario que instales nada ya que interaccionaremos a través de su interfaz web.

Recomendaciones y recursos

Te recomiendo usar al menos tres sandboxes de las comentadas en la sección correspondiente para que veas como suelen diferir sus resultados.

Ojo, SIEMPRE, TODO, USANDO MÁQUINAS VIRTUALES!!!!!

Tienes dos opciones:

- Subir una muestra de malware real
- Navegar entre los informes ya creados por las detonaciones públicas

Como vamos a emplear repositorios de muestras públicos, es posible que la muestra ya esté analizada y se pierda cierta emoción. No desistas, no siempre se acierta.

Tienes muestras (muchas) de malware en estos recursos:

https://github.com/ytisf/theZoo

https://bazaar.abuse.ch/browse/

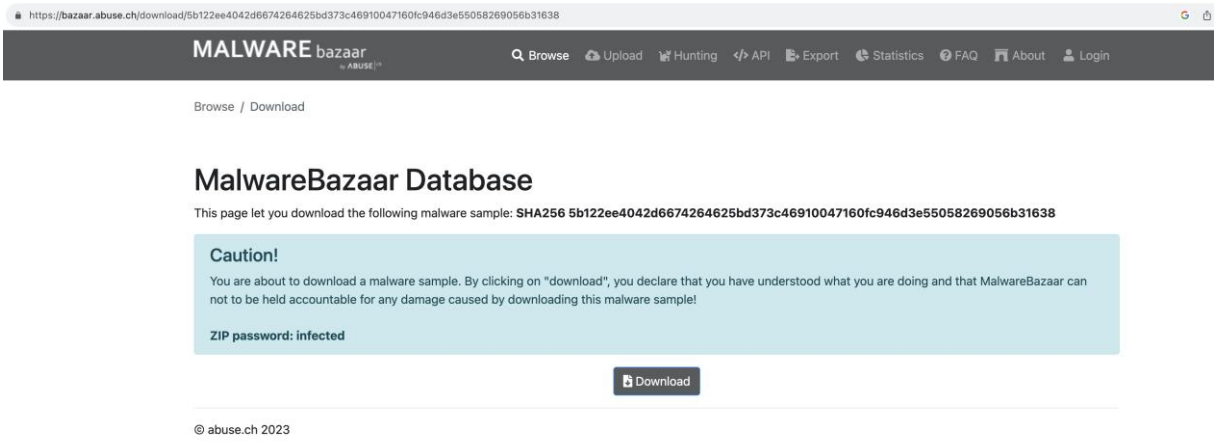
El primero es más histórico, el segundo es muy fresco. El primero no requiere cuenta, el segundo puede que sí pero es gratuita.

El ejercicio es que aprendas a subir muestras y sepas interpretar el resultado.

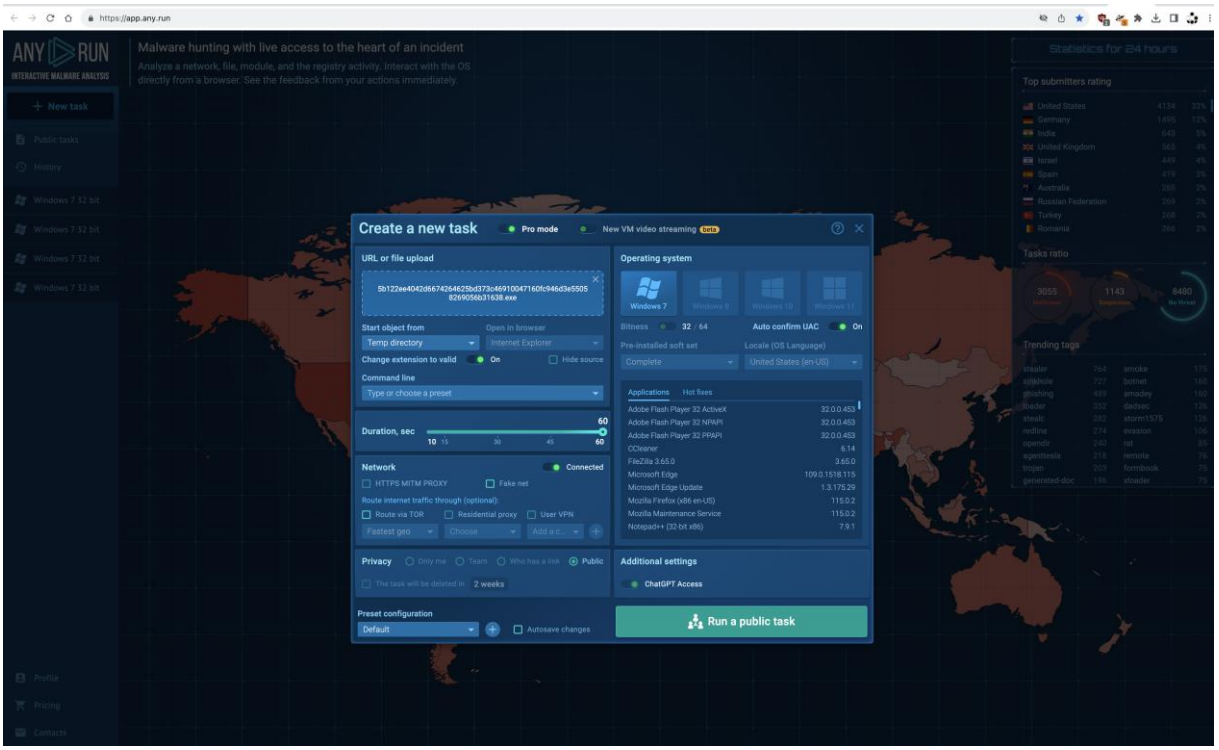
Sube al menos cinco o seis muestras. Intentan que sean frescas para ver como realiza la detonación y análisis.

Bajada y subida de una muestra como ejemplo

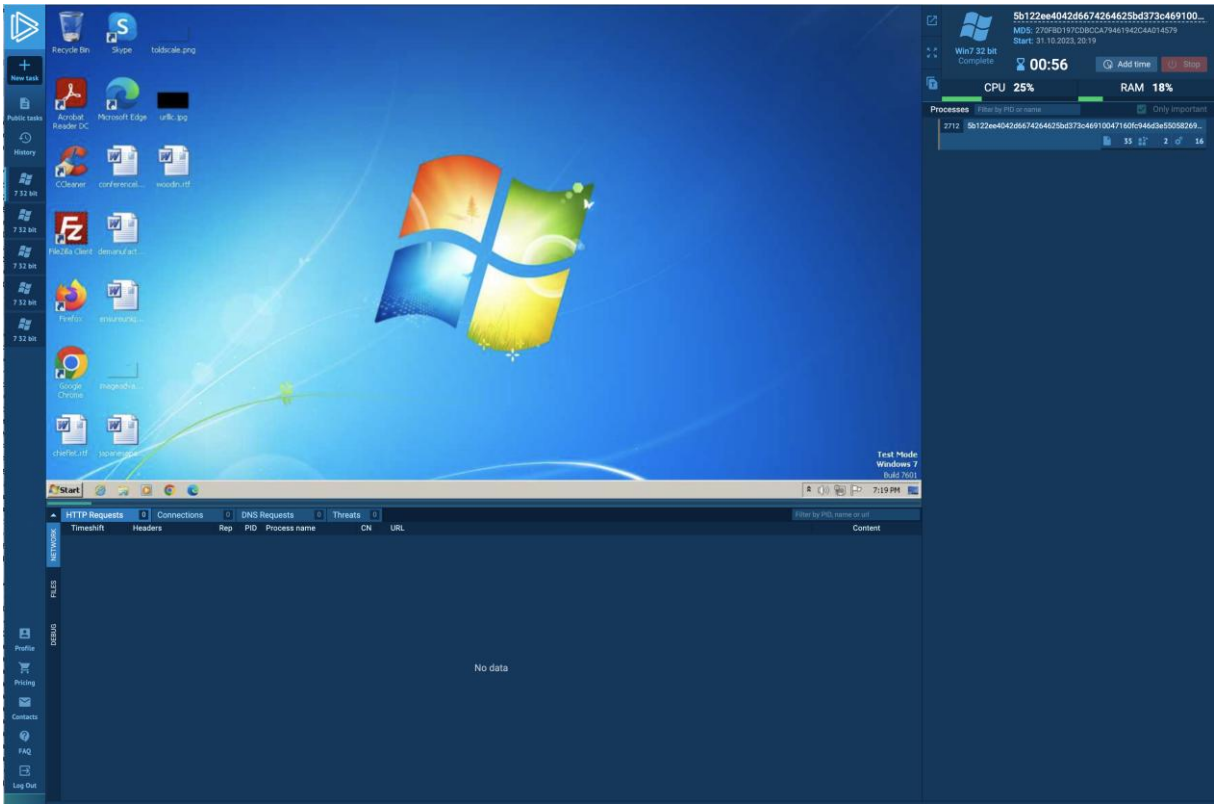
Si bajas una muestra estará en formato zip con contraseña (la clásica 'infected'). Esto es porque si te bajaras la muestra directamente saltarían las alarmas y sistemas de detección. La cruz del analista.



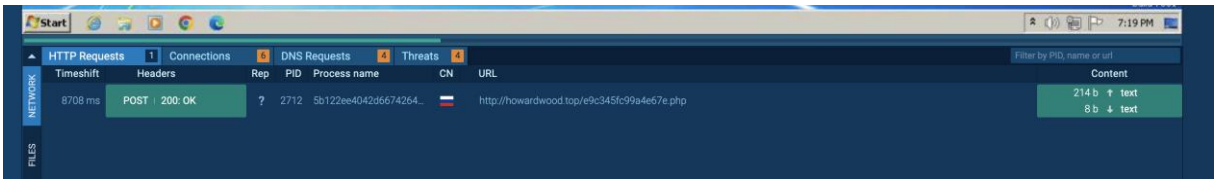
Una vez tenemos la muestra seleccionamos una sandbox y la subimos:



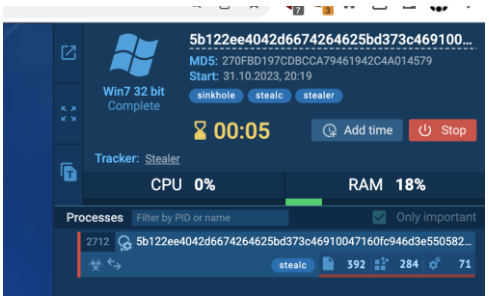
Toca esperar hasta que nos llegue el análisis:



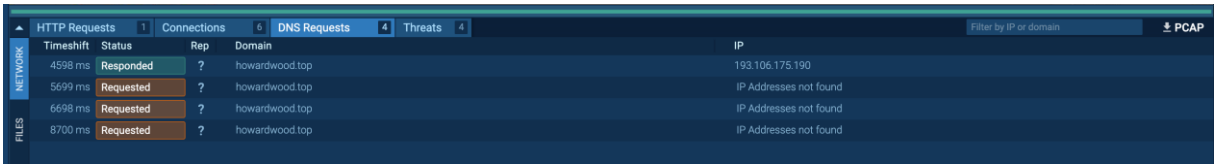
Al poco ya empieza a verse actividad de red bastante sospechosa:



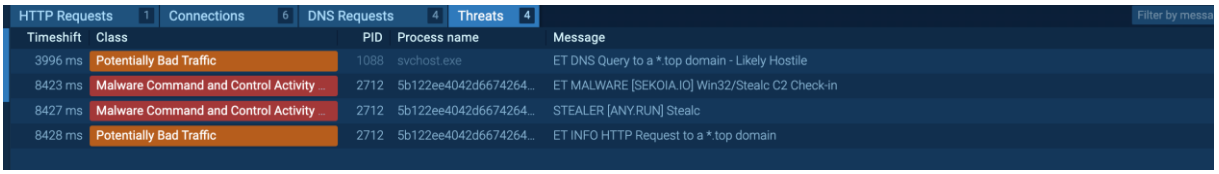
Solo tenemos 1 minuto de actividad:



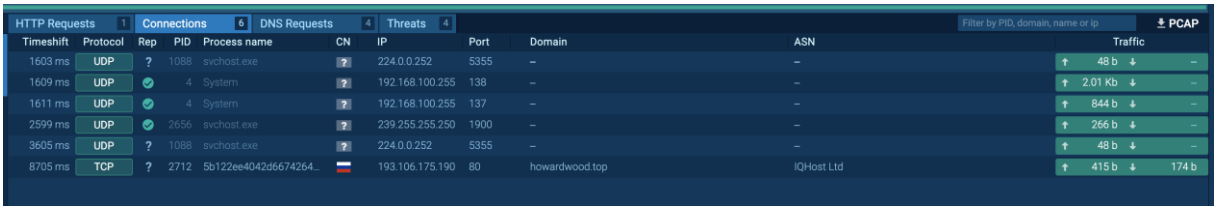
Vemos las resoluciones:



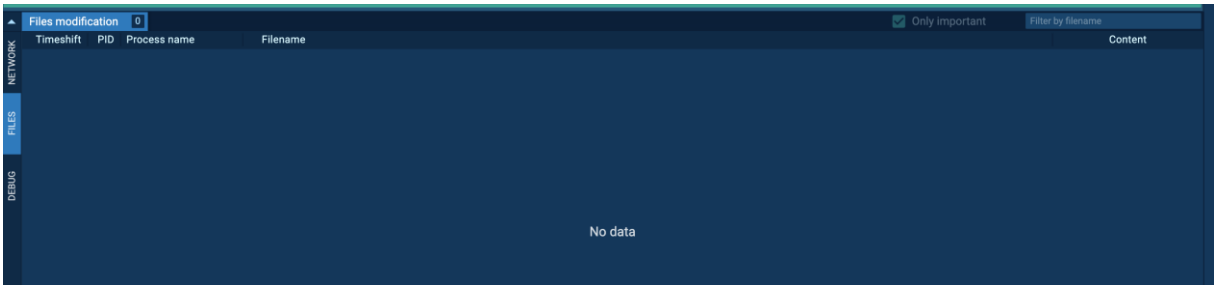
Reglas Suricata y propias de ANY.RUN levantadas durante el proceso:



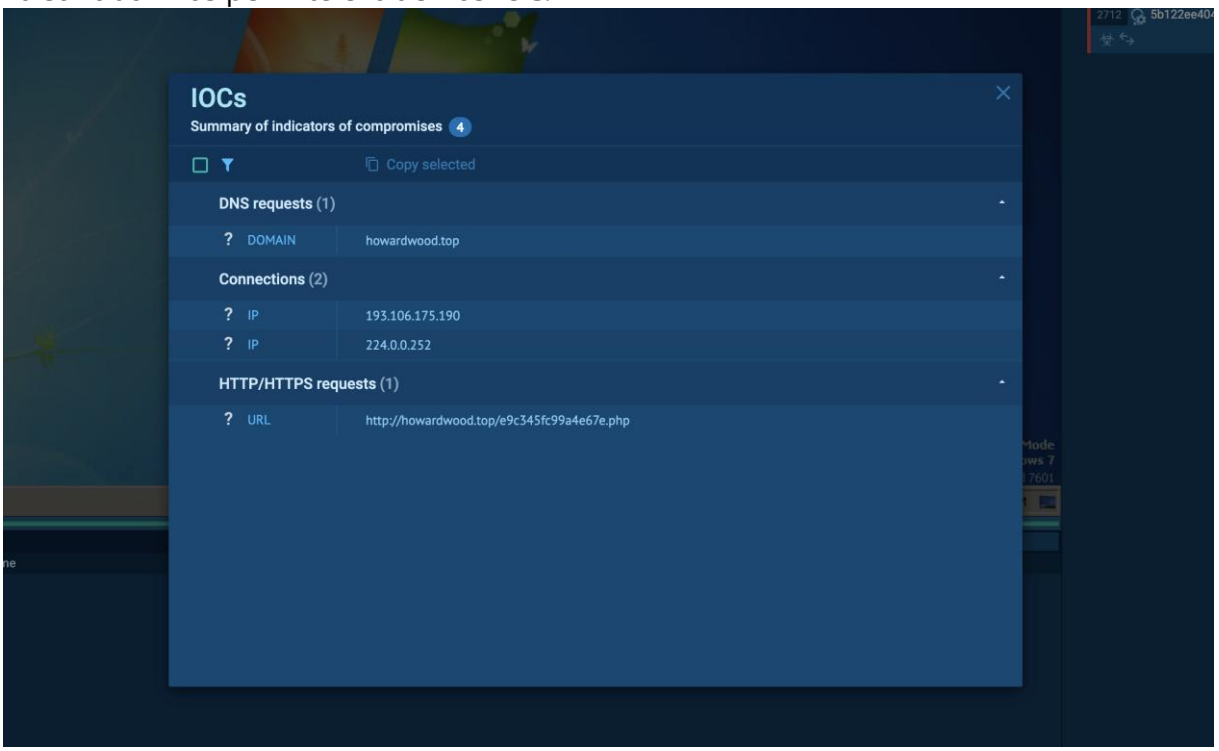
Conexiones de red:



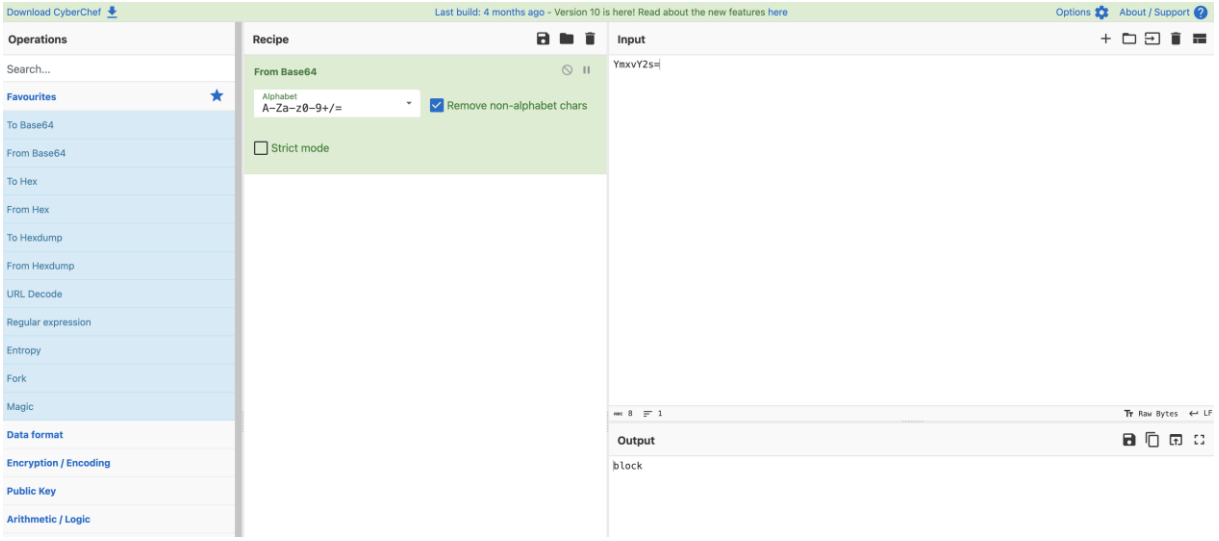
No se han creado archivos, algo curioso:



La sandbox nos permite extraer los IOC:



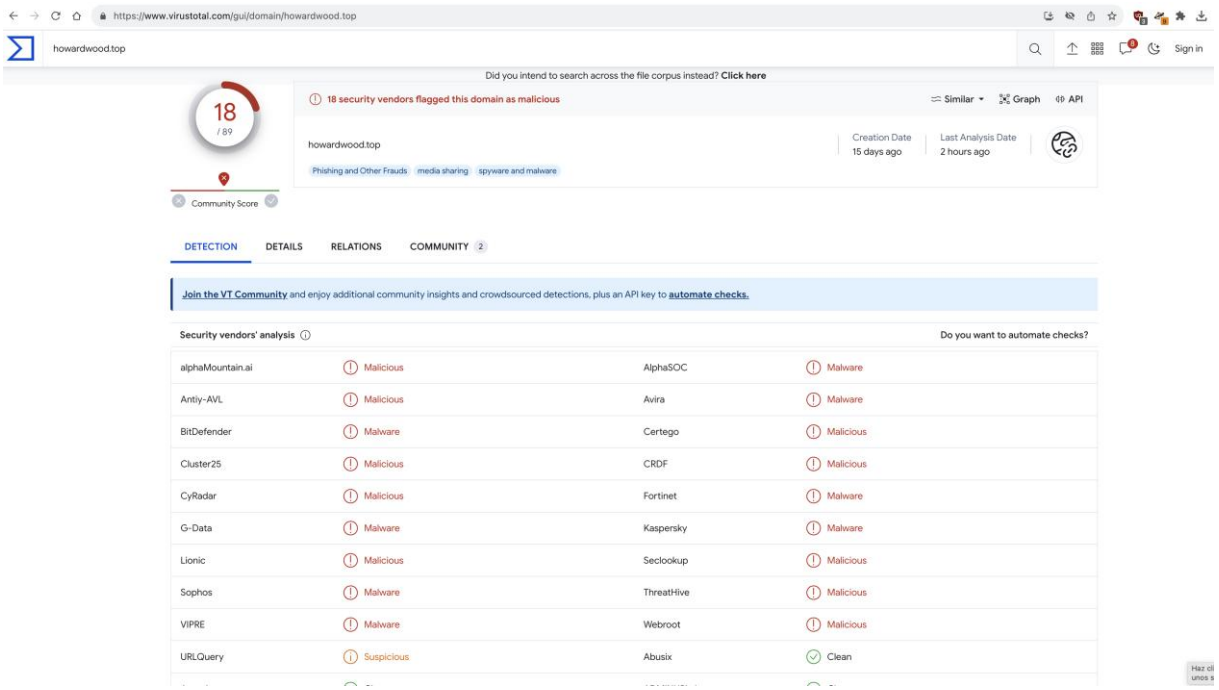
El intercambio con esa URL era prometedor, pero solo contiene una cadena trivialmente codificada en base64:



'block', curioso, ¿verdad?

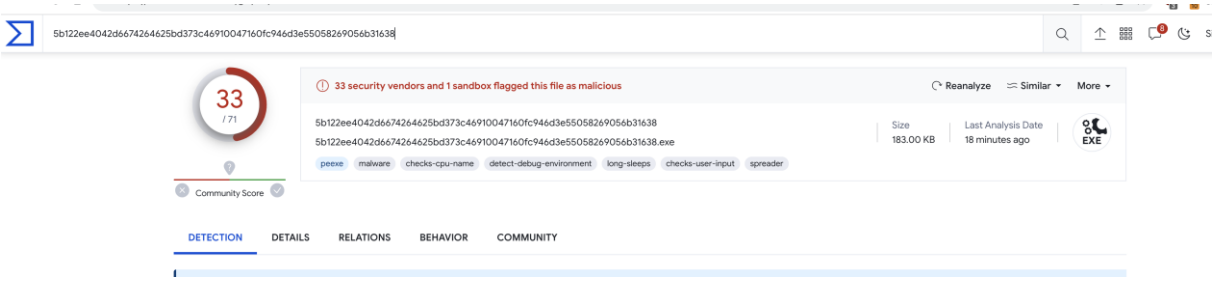
Tirando del hilo

Podemos seguir investigando. Por ejemplo, preguntando por ahí acerca del dominio empleado por el malware, quizás sus IP, etc.

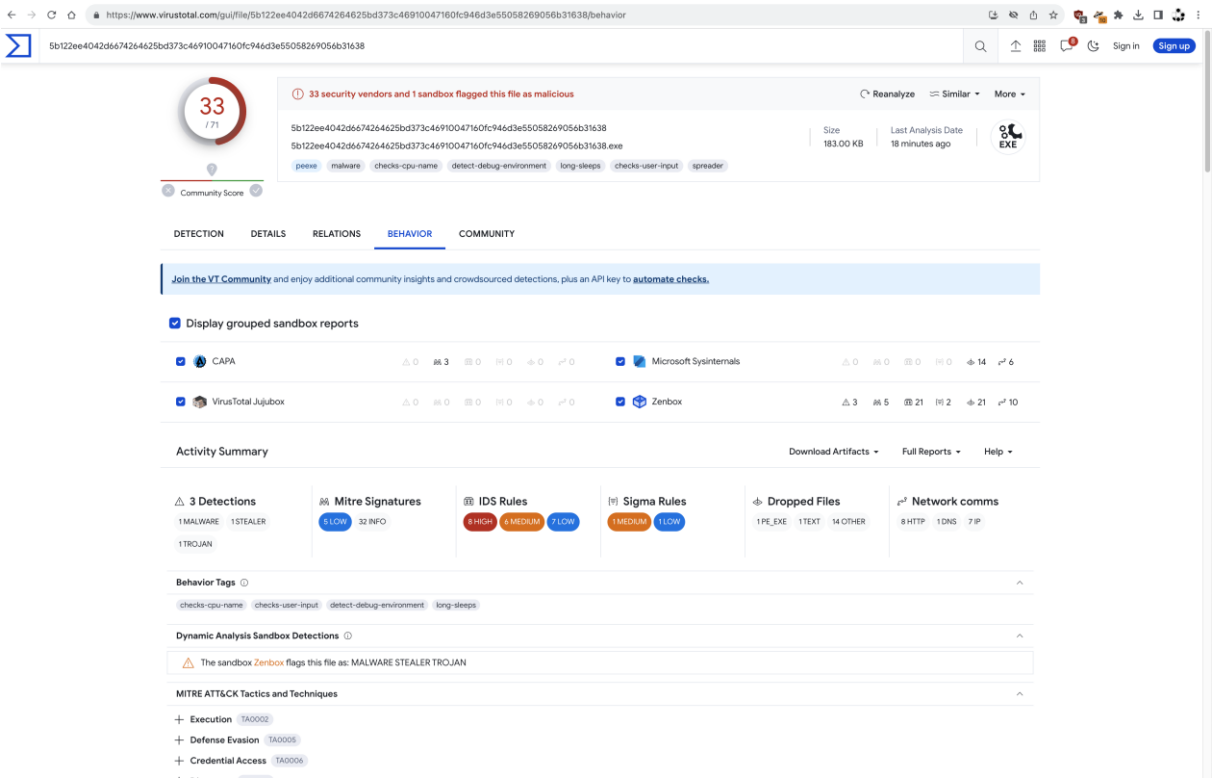


Veamos si VT dispone de información de la muestra, buscamos por el hash:

Bingo, además, fíjate en lo "fresca" que es la muestra. La analizaron hace tan solo 18 minutos:



Incluso encontramos información sobre su comportamiento en otras sandboxes:



Práctica con muestras. Ata cabos. Investiga con otros recursos. Observa como extraen información del malware y como aúnan datos de todo tipo de herramientas (muchas explicadas en el temario).

2.4. SIGMA

Instalar SIGMA para Python:

$ pip install sigma-cli

Eso nos instalará la utilidad de línea de comandos y también la librería pySigma. Por defecto, no instala backends necesarios para hacer pruebas.

Las reglas SIGMA van aparte, puedes instalarlas así:

$ git clone https://github.com/SigmaHQ/sigma

2.4.1. Búsqueda y conversión de reglas

Busca las reglas correspondientes a la detección de Qakbot y conviértelas a un formato que acepte Elasticsearch (eql)

Deberías obtener un resultado similar a este:

Respecto de las "pipelines", existe la opción en determinados plugins de no usar ninguna. Busca la opción adecuada y empleada para generar las reglas. Mira las diferencias en sintaxis cuando se usa o no una pipeline.

2.4.2. Instalación de plugin para PowerShell y puesta en funcionamiento

¿Qué sucede si intentas transformar una regla cualquiera (puedes emplear las de Qakbot) a PowerShell?

Busca como añadir nueva funcionalidad a 'sigma' a través de su sistema de plugins para obtener una conversión a PowerShell. **Ojo a los mensajes de diagnóstico de 'sigma'**.

Deberías obtener una salida similar a esta:

2.5. Snort

2.5.1. Creación de regla para detección de dominios genéricos

El pcap se pondrá a disposición de la clase.

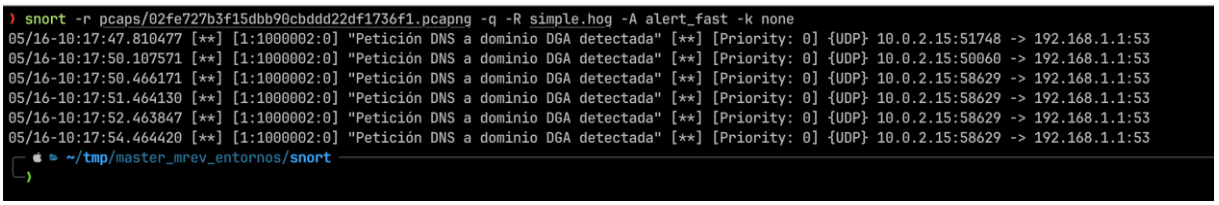
Crea **UNA SOLA** regla para:

- Detectar peticiones DNS salientes
- Que detecten los dominios DGA grabados en el pcap

Consejos:

- Piensa, qué atributo de regla te ayuda a detectar determinados campos por medio de una expresión.
- Snort descarta paquetes cuya checksum sea errónea. Emplea la opción "-k none" para evitarlo.

La salida de tu regla debería proporcionarte idéntico número de alertas y descripción (salvo que cambies el 'msg', que no afecta):



2.6. Extracción de IoC

Los IOC pueden extraerse a mano, pero es laborioso hacerlo. También se pueden y suelen extraer con expresiones regulares, de forma genérica con grep y similares herramientas de línea de comandos.

Existen numerosas utilidades y librerías para diversos lenguajes de programación. Por ejemplo, para Python:

```
$ pip install iocextract
```

Ojo, porque la extracción de IOC suele dar falsos positivos. Conviene revisar la lista de aquello que se ha extraído usando un automatismo.

2.6.1. Práctica de extracción automática de IOC (I)

Vamos a usar la siguiente URL donde la Unit42 de la empresa Palo Alto publica un artículo acerca de una puerta trasera y la vincula a cierto grupo APT.

https://unit42.paloaltonetworks.com/rare-possible-gelsemium-attack-targets-se-asia/

Usa la herramienta de línea de comando que proporciona 'iocextractor' para extraer los IOC que enumera el artículo.

La utilidad de línea de comando necesita que primero descargues la página web. Puedes usar cualquier herramienta tipo 'wget' o 'curl'.

Ejemplo:

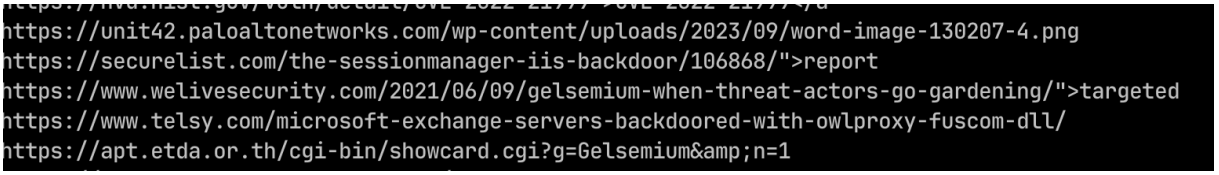
```
~/tmp/master_mrev_entornos
❯ curl https://unit42.paloaltonetworks.com/rare-possible-gelsemium-attack-targets-se-asia/ > index.html
```

La página es un html común:

Ahora, sobre el archivo descargado, lanzamos la herramienta de extracción:

En el uso básico extraerá todos los elementos que cree haber encontrado en la página y que detecta como IOCs.

Mucho cuidado con los automatismo de este tipo porque no son completamente fiables (debido a la complejidad de interpretar información ambigua). Por ejemplo, la extracción en este caso produce artefactos:



Es decir, ha extraído bien pero se lleva consigo parte del html.

Además, si os fijáis, contiene URL que no son maliciosas pero que por contra podían proporcionarnos información adicional.

Posibles puntos de mejora

- ¿Qué estrategias seguirías para automatizar la tarea de descarga de informes?
- ¿Cómo podrías conseguir eliminar los artefactos que 'iocextract' es incapaz de eliminar?

2.6.2. Práctica de extracción automática de IOC (II)

De nada nos sirve tener una lista aglutinada con todos los IOC. Si la pasamos a otro componente de la cadena de ingesta de IOCs, tendrá que volver a parsearlos y extraerlos de nuevo para clasificarlos.
En un escenario real la ingesta se produce ya clasificada. Es decir, tendremos un listado de IP, otro de dominios, otro de URL, de hashes, etc.

Investiga que opciones tenemos disponibles para extraer ciertos IOCs y obtén un archivo por cada uno de ellos (si quieres, ignora IPv6).

El resultado final debe quedarte aproximadamente así:

Posibles puntos de mejora

- No produzcas resultados duplicados.
- Utiliza solo herramientas de línea de comandos disponibles en cualquier distribución Linux.
- Recuerda, el espíritu de la asignatura es: **AUTOMATIZACIÓN**.

2.7. STIX

Para instalar la librería STIX en Python:

$ pip install stix2

2.7.1. Reproducción del caso práctico

Para esta práctica, como introducción a STIX, se recomienda **encarecidamente** seguir el caso práctico elaborado en el capítulo dedicado a STIX.

Reproducir dicho caso te proporcionará una toma de contacto o rodaje para familiarizarte con STIX y afrontar la evaluación.

Presta atención a los campos obligatorios, prescinde esta vez de los campos opcionales si lo ves necesario.

Ve examinando el grafo que estás construyendo en el visualizador web de STIX:

https://oasis-open.github.io/cti-stix-visualization/

Ve montando el bundle STIX, agregando cada nodo o elemento que vayas construyendo. Un ejemplo:

```python
from stix2 import Bundle

elements = []


bundle = Bundle(elements)

print(bundle.serialize(pretty=True))

```

Por cada elemento que agregues, ponlo en 'elements' y se agregará a bundle. Así:

Recuerda que debes importar cada tipo de elemento o clase que necesites desde la librería stix2.