

MÁSTER EN ANÁLISIS DE MALWARE Y REVERSING
MÓDULO 2. ENTORNOS DE ANÁLISIS DE MALWARE - APÉNDICES

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	2
1.1	Advertencia.....	2
1.2	Máquina virtual	2
1.3	Entorno Python.....	2
1.3.1	Pyenv	3
1.3.1.1	Pyenv virtualenv	4
2	Apéndice B: Índice de prácticas voluntarias	5
2.1	Hashes	5
2.1.1	Observa como cambia el hash	5
2.1.2	Hashes difusos.....	7
2.2	YARA	7
2.2.1	Escaneando ejecutables con 'yara'	8
2.2.2	Creando una regla YARA.....	9
2.3	Sandboxes	10
2.4	SIGMA	16
2.4.1	Búsqueda y conversión de reglas	16
2.4.2	Instalación de plugin para PowerShell y puesta en funcionamiento	16
2.5	Snort	17
2.5.1	Creación de regla para detección de dominios genéricos.....	17
2.6	Extracción de IoC	17
2.6.1	Práctica de extracción automática de IOC (I).....	18
2.6.2	Práctica de extracción automática de IOC (II).....	19
2.7	STIX	20
2.7.1	Reproducción del caso práctico	20

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

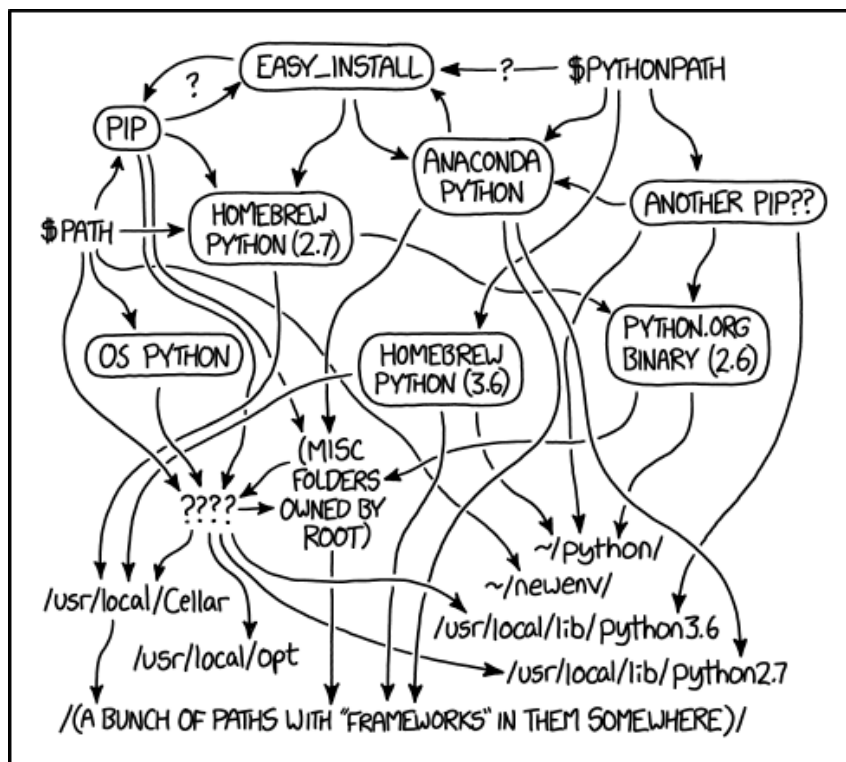
Usa cualquier hipervisor y cualquier sistema Linux (una versión actual, soportada) que te permita instalar los paquetes necesarios para hacer tanto las prácticas como las evaluaciones. Si te sientes cómodo con una distribución en particular, úsala.

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.



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

ILUSTRACIÓN 1 FUENTE: [HTTPS://XKCD.COM/1987/](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 automáticamente 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

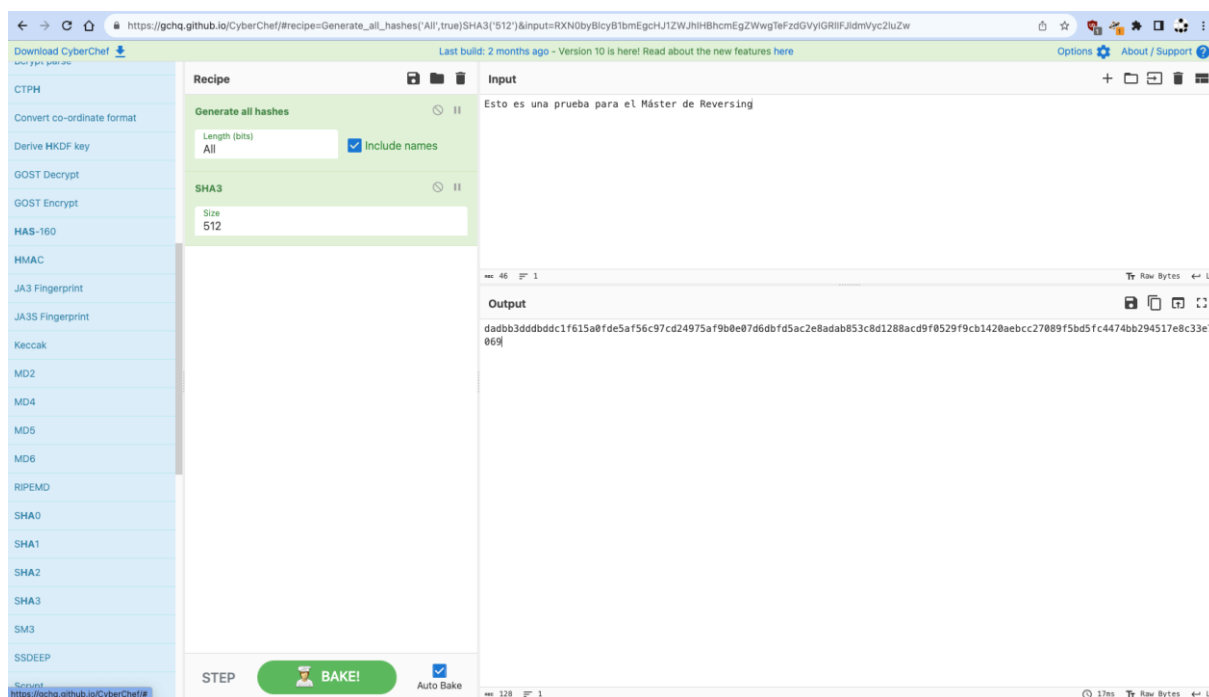
Aquí encontrarás todas las prácticas voluntarias por orden de aparición en el temario.

2.1. Hashes

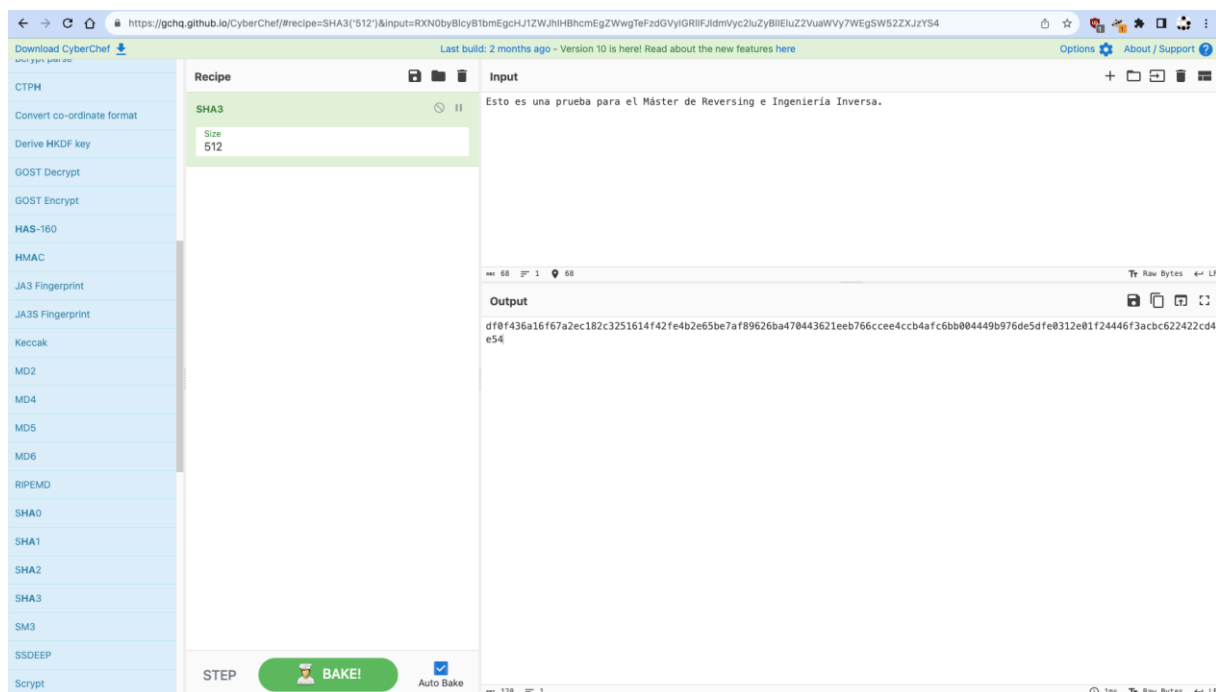
2.1.1. Observa como cambia el hash

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:



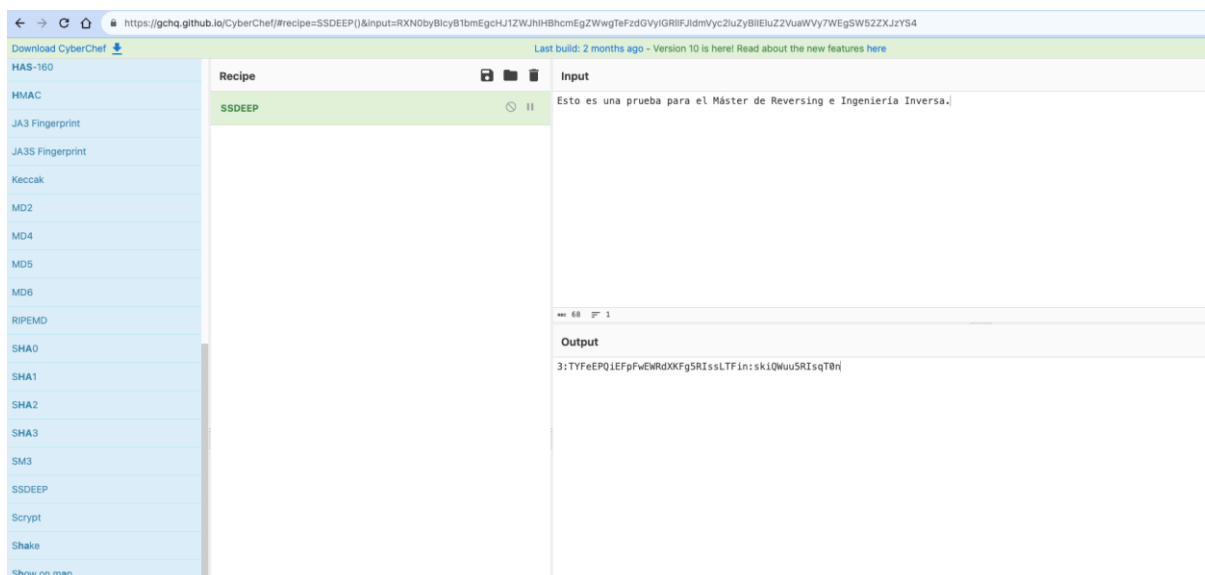
Ahora modifica ligeramente el texto de la entrada y observa como el hash ha cambiado por completo.



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:

```
yara -w -r -m rules/antidebug_antivm_index.yar muestras/0f63cb7f587b9ef434ecd6de4e65be95_994b31a9e7824e107d863bcd3784fc3fed71ab04dd32a9cc8d700010c8f771
SEM_Save [author="Malware Utikonos",original_author="naxonez",source="https://github.com/naxonez/yaraRules/blob/master/AntiDebugging.yara"] muestras/0f63cb7f587b9ef434ecd6de4e65be95_994b31a9e7824e107d863bcd3784fc3fed71ab04dd32a9cc8d700010c8f771
SEM_Init [author="Malware Utikonos",original_author="naxonez",source="https://github.com/naxonez/yaraRules/blob/master/AntiDebugging.yara"] muestras/0f63cb7f587b9ef434ecd6de4e65be95_994b31a9e7824e107d863bcd3784fc3fed71ab04dd32a9cc8d700010c8f771
anti_dbg [author="Dr",description="Checks if being debugged",version="0.2"] muestras/0f63cb7f587b9ef434ecd6de4e65be95_994b31a9e7824e107d863bcd3784fc3fed71ab04dd32a9cc8d700010c8f771
• mrev 3.10.3 © 22:01:06
```

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

Ahora vamos a ver si posee alguna rutina packer:

```
yara -w -r -m rules/packers_index.yar muestras/05f3d0906cc2485feda9596c39ce846b
IsPE32 [] muestras/05f3d0906cc2485feda9596c39ce846b
IsWindowsGUI [] muestras/05f3d0906cc2485feda9596c39ce846b
HasDebugData [author="pusher_",description="DebugData Check",date="2016-07"] muestras/05f3d0906cc2485feda9596c39ce846b
HasRichSignature [author="pusher_",description="Rich Signature Check",date="2016-07"] muestras/05f3d0906cc2485feda9596c39ce846b
VC8_Microsoft_Corporation [] muestras/05f3d0906cc2485feda9596c39ce846b
Microsoft_Visual_Cpp_8 [] muestras/05f3d0906cc2485feda9596c39ce846b
• mrev 3.10.3 © 22:01:06
```

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.

```
yara -w -r -m rules/urls/url.yar muestras/
url [author="Antonio S. <asanchez@plutec.net>"] muestras//02fe727b3f15dbb90cbddd22df1736f1
url [author="Antonio S. <asanchez@plutec.net>"] muestras//5e2a16779719c4ef435bcd39c28b2c6
url [author="Antonio S. <asanchez@plutec.net>"] muestras//1eafc019569f95287aef28ad34df1595
url [author="Antonio S. <asanchez@plutec.net>"] muestras//0ec1a5a678f7c3270e1f64119242defa
• mrev 3.10.3 © 22:01:06
```

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

```
0\x006\x00/\x00l\x00e\x00m\x00a\x00v\x00a\x00s\x00i\x00.\x00p\x00d\x00
0x1d3e5:$url_regex: https://doitory.com/wp-content/uploads/2022/06/lemavasi.pdf
0x1d49h:$url_regex: h\x00t\x00t\x00n\x00:\x00/\x00/\x00s\x00a\x00d\x00d\x00l\x00e\x00
\x000\x006\x00/\x00N\x00e\x00u\x00f\x00r\x00t\x00_\x00N\x00o\x0j\x00f\x00e\x00r\x00t\x00_\x00A\x00r\x00h\x00i\x00t\x00e\x00
\x00p\x00s\x00k\x00o\x00_\x00I\x00z\x00d\x00a\x00n\x00j\x00e\x00.\x00p\x00d\x00
3:$url_regex: https://hogeorgia.com/wp-content/uploads/2022/06/Neufert_Nojfert_Arhitektonsko_Projektovanjesrpsko_Izdanje.pdf
c:$url_regex: h\x00t\x00t\x00p\x00s\x00:\x00/\x00/\x00l\x00o\x00q\x00u\x00a\x00t\x00i\x00c\x00s\x00.\x00c\x00o\x00m\x00/\x00w\x00
\x000\x006\x00/\x00P\x00a\x00p\x00y\x00r\x00u\x00s\x00A\x00u\x00t\x00r\x00v\x006\x003\x00G\x00e\x00r\x00m\x00a\x00n\x00
\x000\x006\x00/\x00P\x00a\x00p\x00y\x00r\x00u\x00s\x00A\x00u\x00t\x00r\x00v\x006\x003\x00G\x00e\x00r\x00m\x00a\x00n\x00
3:$url_regex: h\x00t\x00t\x00p\x00s\x00:\x00/\x00/\x00w\x00w\x00w\x00.\x00v\x00x\x00c\x00.\x00p\x00l\x00/\x00w\x00p\x00-\x00c\x00o\x0
\x00_\x00l\x00n\x00d\x00e\x00r\x00w\x00n\x00r\x00l\x00d\x00_\x00l\x00_\x00E\x00u\x00l\x00l\x00l\x00_\x00M\x00n\x00r\x00i\x00e\x00_\x00

```

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:

```
l l -s muestras
total 35040
-rw-rw-r-- 1 davidgarcia staff 13M 12 may 10:44 0ec1a5a678f7c3270e1f64119242defa
-rw-rw-r-- 1 davidgarcia staff 1.1M 11 may 17:46 1aef019569f95287aef28ad34df1595
-rw-rw-r-- 1 davidgarcia staff 719K 11 may 14:35 7bbc90ec584084ccfa34a1df8d65e98a
-rw-rw-r-- 1 davidgarcia staff 678K 12 may 10:08 0c87716899667648cb519c1e27fed7f
-rw-rw-r-- 1 davidgarcia staff 617K 10 may 13:17 4b7b99b2500c58e882345b76ff04b67b
-rw-rw-r-- 1 davidgarcia staff 324K 12 may 09:58 05f3d0906cc2485feda9596c39ce846b
-rw-rw-r-- 1 davidgarcia staff 322K 12 may 18:44 0f63cb7f587b9ef434ecd6de4e65be95_994b31a9a97824e107d863bcd3f3784fc3fed71ab04dd32a9cc8d700010c8f771
-rw-rw-r-- 1 davidgarcia staff 125K 10 may 13:01 5e2a16779719c4ef435bcdad39c28b2c6
-rw-rw-r-- 1 davidgarcia staff 38K 8 may 16:58 02fe727b3f15dbb90cbdd22df1736f1
-rw-rw-r-- 1 davidgarcia staff 35K 11 may 14:03 7c2dfc11baae9dc39a31073146b1f46
yara s.yara muestras
simple muestras/02fe727b3f15dbb90cbdd22df1736f1
simple muestras/7c2dfc11baae9dc39a31073146b1f46
simple muestras/05f3d0906cc2485feda9596c39ce846b
simple muestras/0f63cb7f587b9ef434ecd6de4e65be95_994b31a9a97824e107d863bcd3f3784fc3fed71ab04dd32a9cc8d700010c8f771
$ * ~ /tmp/master_mrev_entornos/yara
```

2.3. Sandboxes

Para practicar con las sandboxes disponibles puedes usarlas directamente, no es necesario que instales nada ya que interactuaremos 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/>

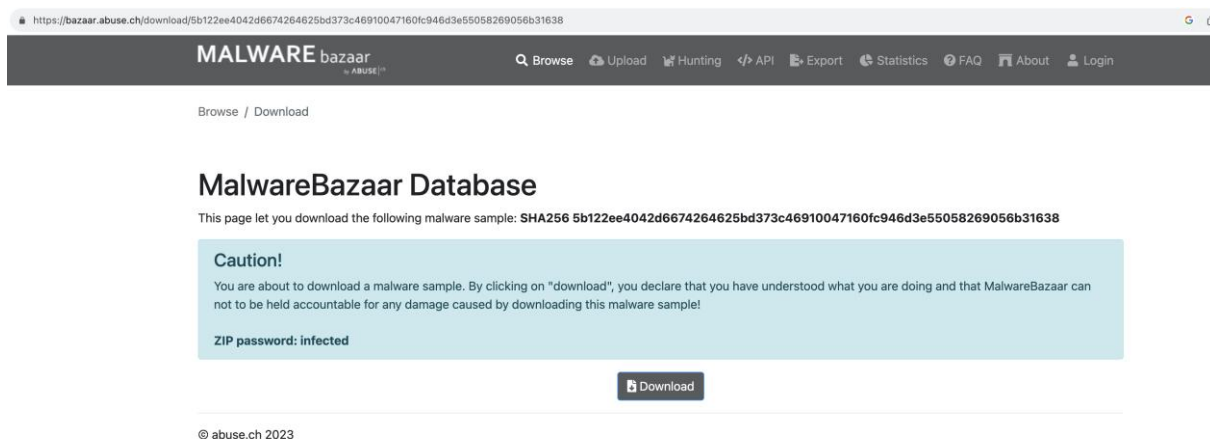
El primero es más histórico, el segundo es muy fresco. El primero no requiere cuenta, el segundo puede que sí pero es gratuita.

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

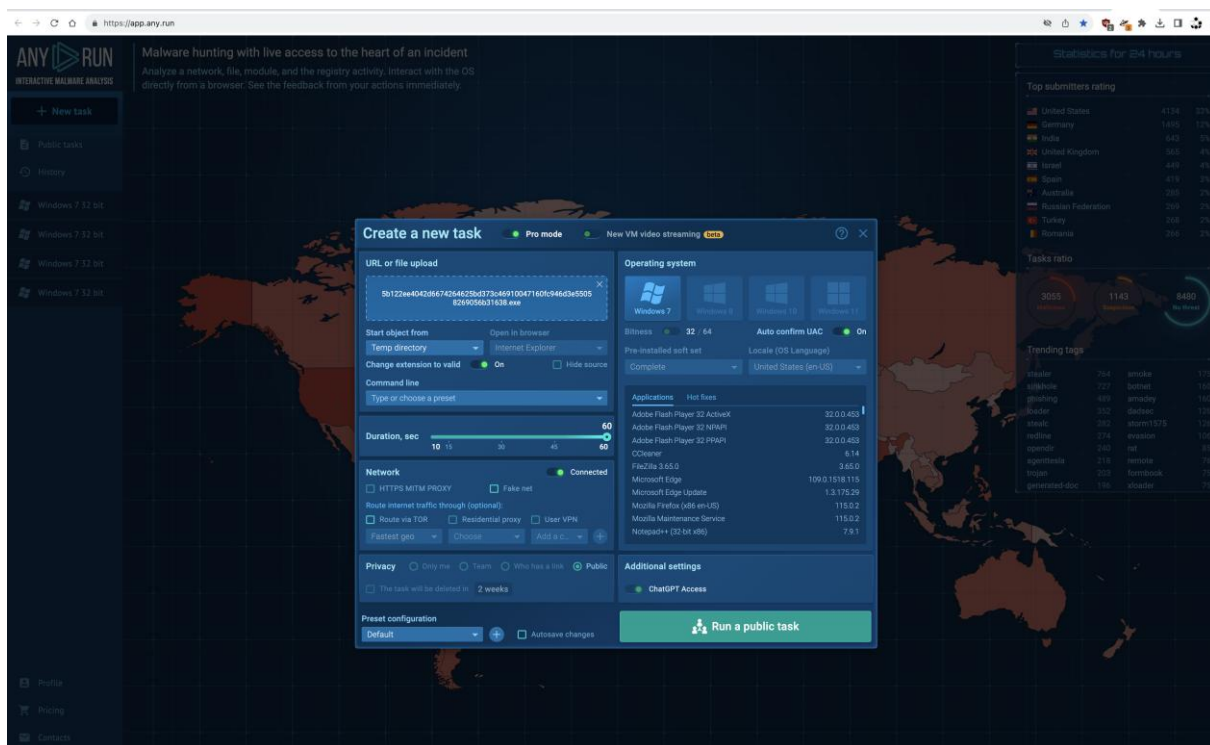
Sube al menos cinco o seis muestras. Intentan que sean frescas para ver como realiza la detonación y análisis.

Bajada y subida de una muestra como ejemplo

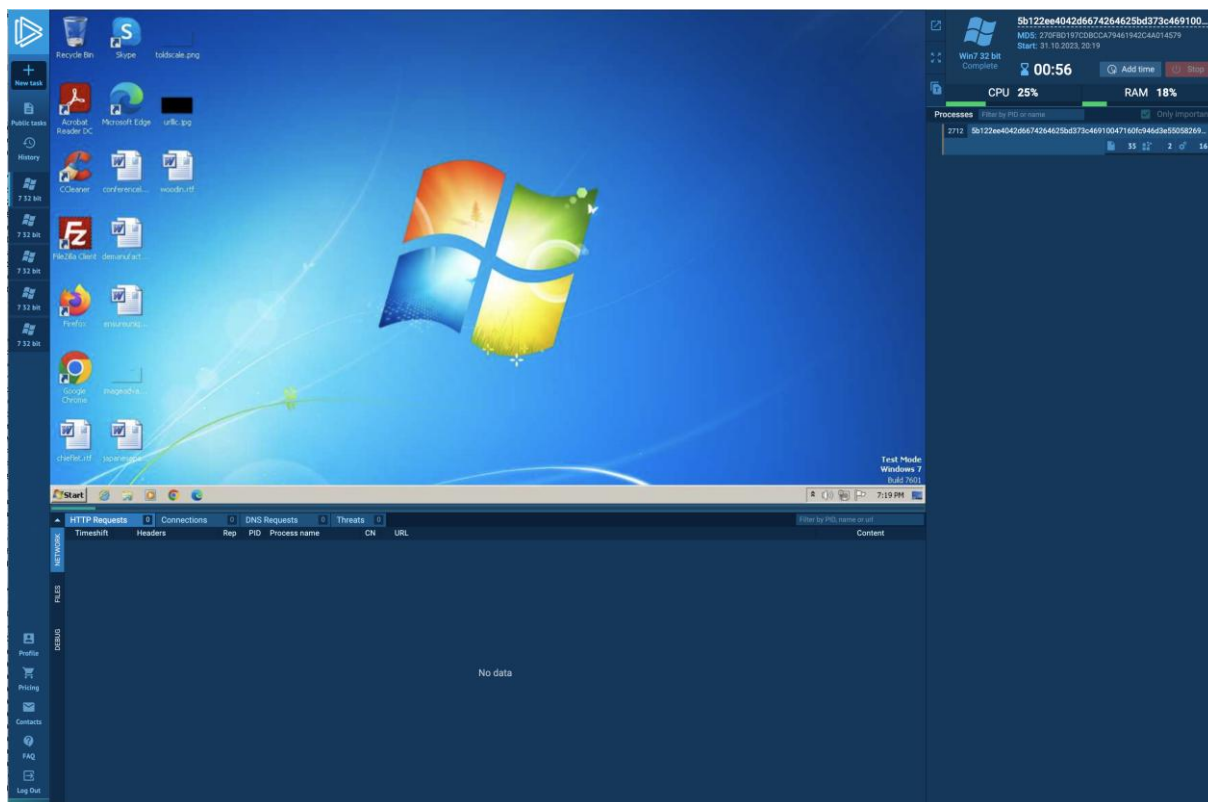
Si bajas una muestra estará en formato zip con contraseña (la clásica 'infected'). Esto es porque si te bajaras la muestra directamente saltarían las alarmas y sistemas de detección. La cruz del analista.



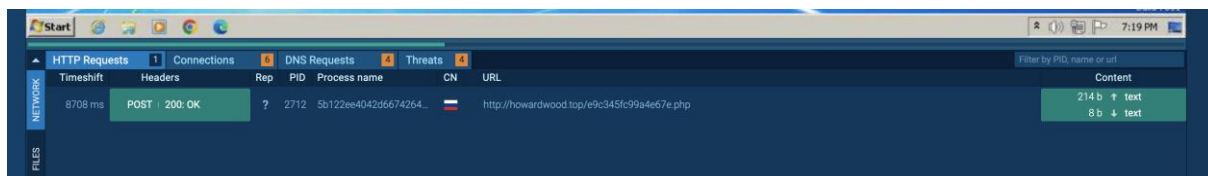
Una vez tenemos la muestra seleccionamos una sandbox y la subimos:



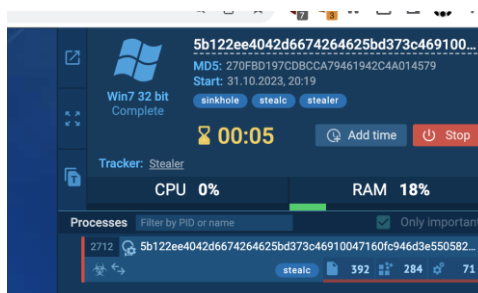
Toca esperar hasta que nos llegue el análisis:



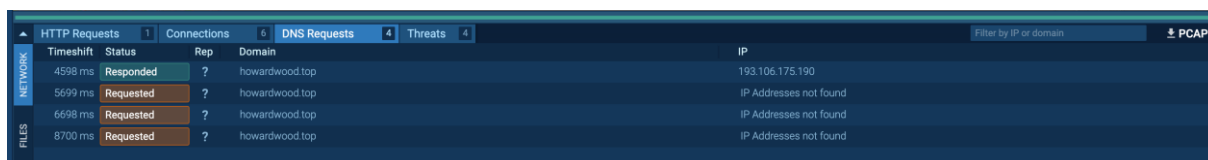
Al poco ya empieza a verse actividad de red bastante sospechosa:



Solo tenemos 1 minuto de actividad:



Vemos las resoluciones:



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

HTTP Requests					1	Connections	6	DNS Requests	4	Threats	4	Filter by message
Timeshift	Class	PID	Process name	Message								
3996 ms	Potentially Bad Traffic	1088	svchost.exe	ET DNS Query to a *.top domain - Likely Hostile								
8423 ms	Malware Command and Control Activity	2712	5b122ee4042d6674264...	ET MALWARE [SEKQIA.IO] Win32/Stealc C2 Check-in								
8427 ms	Malware Command and Control Activity	2712	5b122ee4042d6674264...	STEALER [ANY.RUN] Stealc								
8428 ms	Potentially Bad Traffic	2712	5b122ee4042d6674264...	ET INFO HTTP Request to a *.top domain								

Conexiones de red:

HTTP Requests											Connections		DNS Requests		Threats		Filter by PID, domain, name or ip		PCAP
TimeShift	Protocol	Rep	PID	Process name	CN	IP	Port	Domain	ASN	Traffic									
1603 ms	UDP	?	1088	svchost.exe	?	224.0.0.252	5355	-	-	↑ 48 b ↓	-								
1609 ms	UDP	✓	4	System	?	192.168.100.255	138	-	-	↑ 2.01 Kb ↓	-								
1611 ms	UDP	✓	4	System	?	192.168.100.255	137	-	-	↑ 844 b ↓	-								
2599 ms	UDP	✓	2656	svchost.exe	?	239.255.255.250	1900	-	-	↑ 266 b ↓	-								
3605 ms	UDP	?	1088	svchost.exe	?	224.0.0.252	5355	-	-	↑ 48 b ↓	-								
8705 ms	TCP	?	2712	5b122ee4042d6674264...	🇵🇱	193.106.175.190	80	howardwood.top	IQ-Host Ltd	↑ 415 b ↓	174 b								

No se han creado archivos, algo curioso:

<div> <div>Files modification</div> <div>0</div> </div> <div> <div>Only important</div> <div>Filter by filename</div> </div>	Timeshift	PID	Process name	Filename	Content
	No data				

La sandbox nos permite extraer los IOC:

IOCs ✕

Summary of indicators of compromises 4

☐ ▼ 📄 Copy selected

DNS requests (1) ⌵

? DOMAIN	howardwood.top
----------	----------------

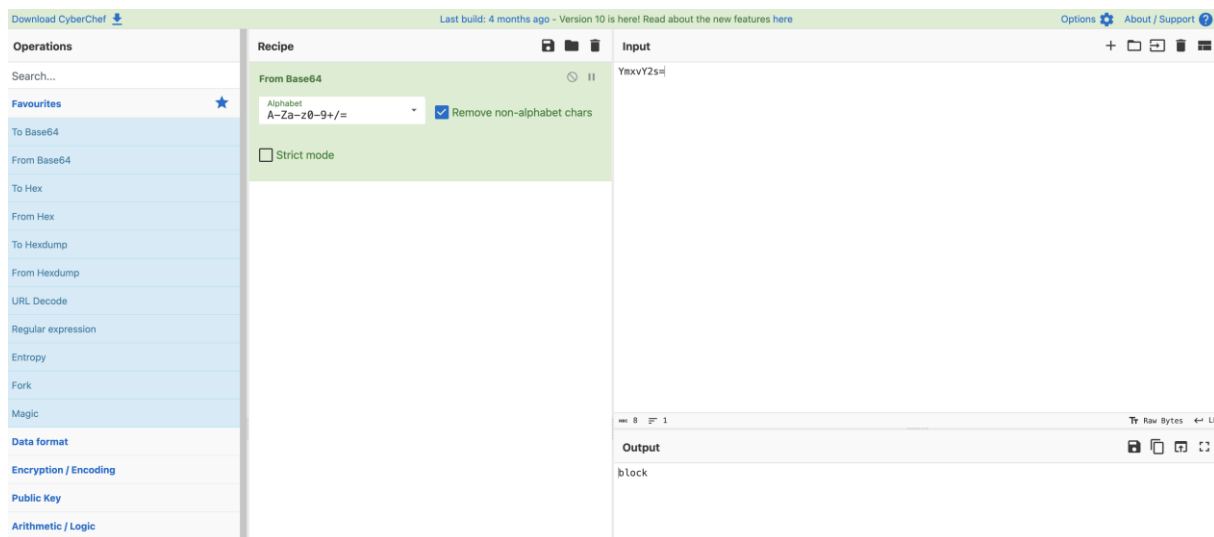
Connections (2) ⌵

? IP	193.106.175.190
? IP	224.0.0.252

HTTP/HTTPS requests (1) ⌵

? URL	http://howardwood.top/e9c345fc99a4e67e.php
-------	--

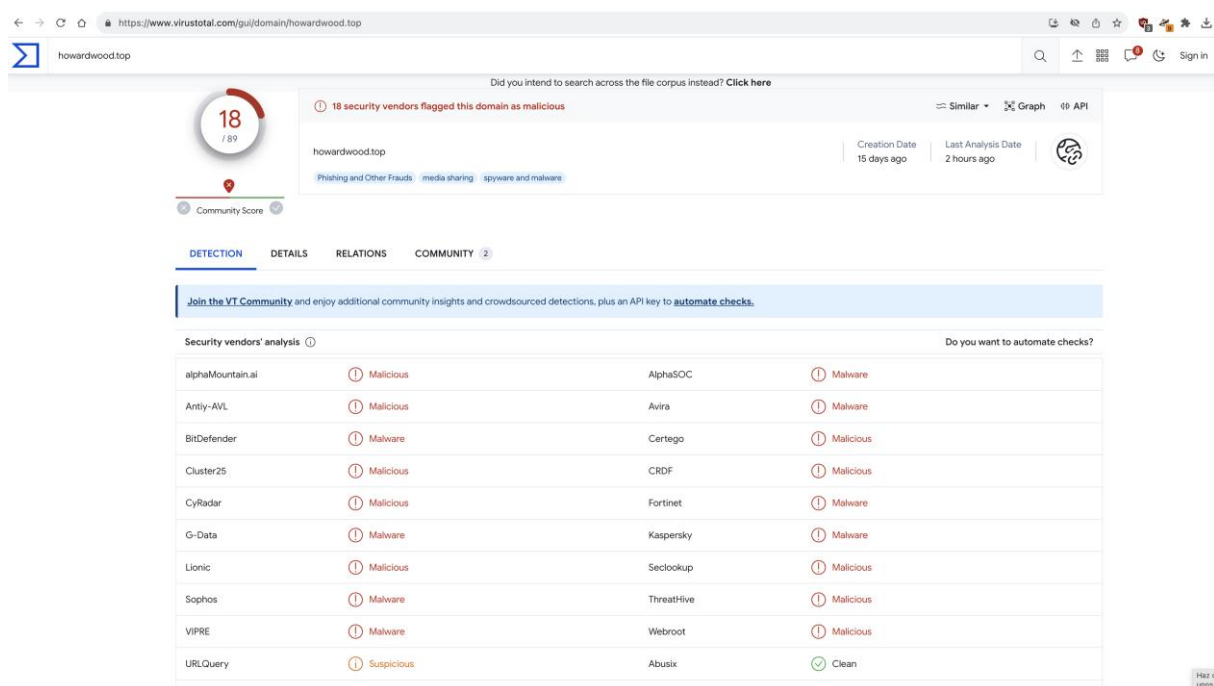
El intercambio con esa URL era prometedor, pero solo contiene una cadena trivialmente codificada en base64:



'b'lock', curioso, ¿verdad?

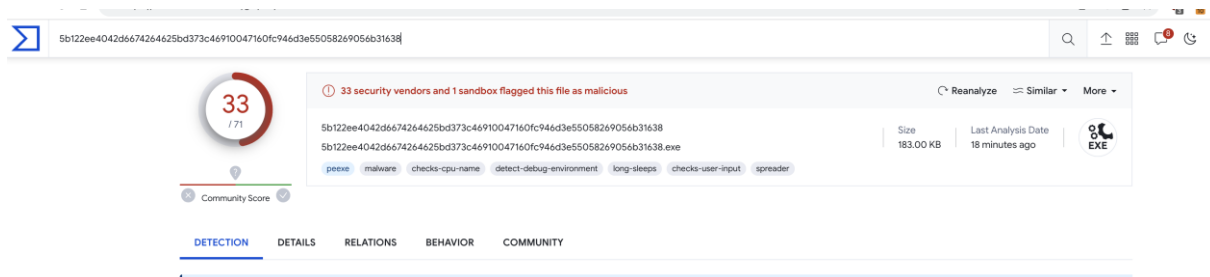
Tirando del hilo

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

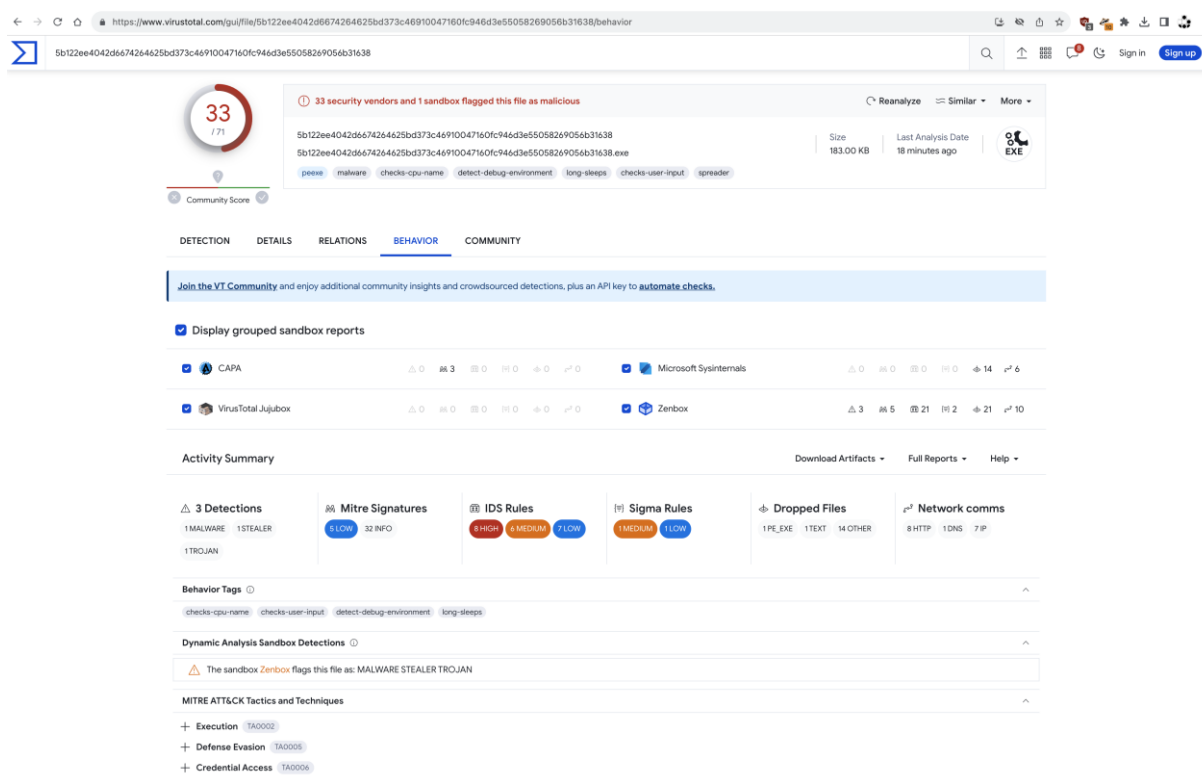


Veamos si VT dispone de información de la muestra, buscamos por el hash:

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 (eq1)

Deberías obtener un resultado similar a este:

```
any where Image:"\\regsvr32.exe" and (CommandLine like- ("% /s", "% -s*)) and CommandLine:"* calc"
any where ((ParentImage like- ("%\\cmd.exe", "%\\cscript.exe", "%\\curl.exe", "%\\mshta.exe", "%\\powershell.exe", "%\\pwsh.exe", "%\\wscript.exe")) and Image:"\\rundll32.exe" and (CommandLine like- ("%\\ProgramData\\*", "%\\Users\\Publi
s\\*", "%\\AppData\\Local\\Temp\\*", "%\\AppData\\Roaming\\*")) and CommandLine:"*.dll")
any where ((ParentImage like- ("%\\cmd.exe", "%\\cscript.exe", "%\\curl.exe", "%\\mshta.exe", "%\\powershell.exe", "%\\pwsh.exe", "%\\wscript.exe")) and Image:"\\rundll32.exe" and (CommandLine like- ("%\\ProgramData\\*", "%\\Users\\Publi
s\\*", "%\\AppData\\Local\\Temp\\*", "%\\AppData\\Roaming\\*")) and (CommandLine like- ("%\\*", "%bind", "%DrawThemeIcon", "%G610", "%G610", "%jhbvgyftr", "%kjhbkjvydrt", "%LS88", "%Notd", "%H115", "%next", "%Nkn", "%print", "%qqq", "%
veeq", "%R332", "%test", "%time", "%updt", "%vips", "%Wind", "%WISO", "%X555", "%XLS9", "%XAutopen", "%X888"))
any where ((ParentImage like- ("%\\cmd.exe", "%\\cscript.exe", "%\\curl.exe", "%\\mshta.exe", "%\\powershell.exe", "%\\pwsh.exe", "%\\wscript.exe")) and Image:"\\rundll32.exe" and (CommandLine like- ("%\\ProgramData\\*", "%\\Users\\Publi
s\\*", "%\\AppData\\Local\\Temp\\*", "%\\AppData\\Roaming\\*")) and (not CommandLine:"*.dll")
any where Image:"\\bootuninstall.exe" or (Hashes like- ("%\\IPHASH=E772C81507231104F88C398743E68E5*", "%SHA256=423A9D1D418E2DC38EAB99F0F3121D2072472D0426260283A6388622C05180*", "%SHA256=559CAE635F0D070A5289482EF436831D48B1A5A0F51750836F3
260749291D086*", "%SHA256=B55EB5481F770DE5A08FA6E9D953DAEBC28800DF9461144B16D6A2817CC5071*", "%SHA256=FAB48536AA37C4ABC8BE97AB9C1F86CB338A3923D423FDC2859E9D6A3FABEAB*"))
```

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:

```
Get-WinEvent -LogName "None" | Read-WinEvent | Where-Object {($_.Image.EndsWith("regsvr32.exe")) -and ($_.CommandLine.Contains(" /s") -or $_.CommandLine.Contains(" -s")) -and $_.CommandLine.EndsWith(" calc")}
```

```
Get-WinEvent -LogName "None" | Read-WinEvent | Where-Object {((ParentImage.EndsWith("cmd.exe") -or ParentImage.EndsWith("powershell.exe") -or ParentImage.EndsWith("vbscript.exe") -or ParentImage.EndsWith("wscript.exe") -or ParentImage.EndsWith("rundll32.exe") -or ParentImage.EndsWith("ProgramData") -or ParentImage.EndsWith("UsersPublic") -or ParentImage.EndsWith("AppDataLocalTemp") -or ParentImage.EndsWith("AppDataRoaming")) -and ($_.CommandLine.Contains("dl"))}
```

```
Get-WinEvent -LogName "None" | Read-WinEvent | Where-Object {((ParentImage.EndsWith("cmd.exe") -or ParentImage.EndsWith("powershell.exe") -or ParentImage.EndsWith("vbscript.exe") -or ParentImage.EndsWith("wscript.exe") -or ParentImage.EndsWith("rundll32.exe") -or ParentImage.EndsWith("ProgramData") -or ParentImage.EndsWith("UsersPublic") -or ParentImage.EndsWith("AppDataLocalTemp") -or ParentImage.EndsWith("AppDataRoaming")) -and ($_.CommandLine.EndsWith("bind") -or $_.CommandLine.EndsWith("DrawThemeIcon") -or $_.CommandLine.EndsWith("GSI0") -or $_.CommandLine.EndsWith("G170") -or $_.CommandLine.EndsWith("jhhvgyfr") -or $_.CommandLine.EndsWith("kjhbkjvdyfr") -or $_.CommandLine.EndsWith("L588") -or $_.CommandLine.EndsWith("MofD") -or $_.CommandLine.EndsWith("W115") -or $_.CommandLine.EndsWith("next") -or $_.CommandLine.EndsWith("H1ar") -or $_.CommandLine.EndsWith("print") -or $_.CommandLine.EndsWith("qqq") -or $_.CommandLine.EndsWith("qqq") -or $_.CommandLine.EndsWith("q332") -or $_.CommandLine.EndsWith("Test") -or $_.CommandLine.EndsWith("Time") -or $_.CommandLine.EndsWith("Upd") -or $_.CommandLine.EndsWith("vips") -or $_.CommandLine.EndsWith("winr") -or $_.CommandLine.EndsWith("wsg") -or $_.CommandLine.EndsWith("X555") -or $_.CommandLine.EndsWith("X155") -or $_.CommandLine.EndsWith("xlrtopen") -or $_.CommandLine.EndsWith("X588"))}
```

```
Get-WinEvent -LogName "None" | Read-WinEvent | Where-Object {((ParentImage.EndsWith("cmd.exe") -or ParentImage.EndsWith("powershell.exe") -or ParentImage.EndsWith("vbscript.exe") -or ParentImage.EndsWith("wscript.exe") -or ParentImage.EndsWith("rundll32.exe") -or ParentImage.EndsWith("ProgramData") -or ParentImage.EndsWith("UsersPublic") -or ParentImage.EndsWith("AppDataLocalTemp") -or ParentImage.EndsWith("AppDataRoaming")) -and ($_.CommandLine.Contains("dl"))}
```

```
Get-WinEvent -LogName "None" | Read-WinEvent | Where-Object {($_.Image.EndsWith("QotUninstall.exe") -or ($_.Hashes.Contains("DHPASH:772C815072311D4F8C33907A3EABE5") -or $_.Hashes.Contains("SHA256:423A9D13D410E2DC38EAB99F0F31210207247200426260203A63888220C05180") -or $_.Hashes.Contains("SHA256:559CAE35F00970A52894632F43A031048B1A5A0F5176083AF3280749291D0B6") -or $_.Hashes.Contains("SHA256:855EB5481F770DE9AD0FAE9D95304AEB2800D0F9641348014ED42817CC5071") -or $_.Hashes.Contains("SHA256:FAB84853AAA37C4ABC8BE97AB9C1F84C33863923D423FDC2859EB9D63FA8EAD"))}
```

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):

```
snort -r pcaps/02fe727b3f15dbb90cbddd22df1736f1.pcapng -q -R simple.hog -A alert_fast -k none
```

```
05/16-10:17:47.810477 [**] [1:1000002:0] "Petición DNS a dominio DGA detectada" [**] [Priority: 0] {UDP} 10.0.2.15:51748 -> 192.168.1.1:53
```

```
05/16-10:17:50.107571 [**] [1:1000002:0] "Petición DNS a dominio DGA detectada" [**] [Priority: 0] {UDP} 10.0.2.15:50060 -> 192.168.1.1:53
```

```
05/16-10:17:50.466171 [**] [1:1000002:0] "Petición DNS a dominio DGA detectada" [**] [Priority: 0] {UDP} 10.0.2.15:58629 -> 192.168.1.1:53
```

```
05/16-10:17:51.464130 [**] [1:1000002:0] "Petición DNS a dominio DGA detectada" [**] [Priority: 0] {UDP} 10.0.2.15:58629 -> 192.168.1.1:53
```

```
05/16-10:17:52.463847 [**] [1:1000002:0] "Petición DNS a dominio DGA detectada" [**] [Priority: 0] {UDP} 10.0.2.15:58629 -> 192.168.1.1:53
```

```
05/16-10:17:54.464420 [**] [1:1000002:0] "Petición DNS a dominio DGA detectada" [**] [Priority: 0] {UDP} 10.0.2.15:58629 -> 192.168.1.1:53
```

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:

```
1 <!doctype html>  
2 <html lang="en-US">  
3 <head>  
4 <meta charset="utf-8">  
5 <meta http-equiv="x-ua-compatible" content="ie=edge">  
6 <meta name="viewport" content="width=device-width, initial-scale=1">  
7 <link rel="apple-touch-icon" sizes="180x180" href="https://unit42.paloaltonetworks.com/wp-content/themes/unit42-v5/favicon/icon-Unit42-180x180.png">  
8 <link rel="icon" type="image/png" sizes="32x32" href="https://unit42.paloaltonetworks.com/wp-content/themes/unit42-v5/favicon/icon-Unit42-32x32.png">  
9 <link rel="icon" type="image/png" sizes="16x16" href="https://unit42.paloaltonetworks.com/wp-content/themes/unit42-v5/favicon/icon-Unit42-16x16.png">  
10 <link rel="manifest" href="https://unit42.paloaltonetworks.com/wp-content/themes/unit42-v5/favicon/site.webmanifest">  
11 <link rel="mask-icon" href="https://unit42.paloaltonetworks.com/wp-content/themes/unit42-v5/favicon/safari-pinned-tab.svg" color="#000000">  
12 <meta name="msapplication-TileColor" content="#000000">  
13 <meta name="theme-color" content="#000000">  
14 <script type="text/javascript">  
15 var main_site_url = 'https://www.paloaltonetworks.com';  
16 var maindomain_lang = 'https://www.paloaltonetworks.com';  
17 function getParameterByName(name, url) {  
18   if(url == null){  
19     url = window.location.href;  
20   }  
21   name = name.replace(/[\\\/]/g, '\\&');  
22   var regex = new RegExp('[?&]' + name + '=[^&#]*')&#160;);  
23   results = regex.exec(url);  
24   if (!results) return null;  
25   if (results[2]) return '&#160;';  
26   return decodeURIComponent(results[2].replace(/\+/g, ' '));  
27 }  
28 var container_q = getParameterByName('container');  
29 var d_lang = 'en';  
30 if(container_q != '' &#160;&#160;container_q != null){  
31   sessionStorage.setItem('container', container_q);  
32   location.href = 'https://unit42.paloaltonetworks.com/rare-possible-gelsemium-attack-targets-se-asia';  
33 }  
34 </script>
```

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

```
~/tmp/master_mrev_entornos  
> iocextract -i index.html
```

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:

```
https://www.msc.gov/vch/detail/ove-2022-21777-ove-2022-21777-0  
https://unit42.paloaltonetworks.com/wp-content/uploads/2023/09/word-image-130207-4.png  
https://securelist.com/the-sessionmanager-iis-backdoor/106868/">report  
https://www.welivesecurity.com/2021/06/09/gelsemium-when-threat-actors-go-gardening/">targeted  
https://www.telsy.com/microsoft-exchange-servers-backdoored-with-owlproxy-fuscom-dll/  
https://apt.etda.or.th/cgi-bin/showcard.cgi?g=Gelsemium&n=1
```

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:

```
) head *.txt
==> emails.txt <==

==> hashes.txt <==
17392669a04f17fda068d18ae5850d135f3912d08b4e2eee81fce915849887b3
181feef51991b162bdf5d49bb7fd368d9ec2b535475b88bc197d70d73eef886
24eb9c77448dda2d7cfecc60c804a378e89cbd450fbf7f4db875eb131cd4510a
29cc79a451f73bac43dbe9455d2184770beae69f4e6bc2d824abd2cfbedf53f1
2f3abc59739b248ee26a575700eef93b18bd2029eb9f8123598ffdd81fa54d8b
314f698ae04b8ffa41c2e28a001a5735
3268f269371a81dbdce8c4eedffd8817c1ec2eadec9ba4ab043cb779c2f8a5d2
3be95477e1d9f3877b4355cff3fbcdd3589bb7f6349fd4ba6451e1e9d32b7fa6
4dcde3fd7f0ab80bc34b924ecaa640165ee49aa1a22179b3f580b2f74705dd9
527063cb9da5eac2e4b290019eaa5edd47ff3807fec74efa0f1b7ddf5a1b271

==> ip.txt <==
27.124.26[. ]83
27.124.26[. ]86

==> url.txt <==
27.124.26[. ]83.\r\nEarthWorm\r\nEarthWorm
27.124.26[. ]83</span>.</p>
27.124.26[. ]83</span></li>
27.124.26[. ]86
27.124.26[. ]86</span>
27.124.26[. ]86</span></li>
http://browsehappy.com/">upgrade
http://www.w3.org/2000/svg
https%3A%2F%2Funit42
https://api.w.org/
```

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:

```
1 from stix2 import Bundle
2
3 elements = []
4
5
6 bundle = Bundle(elements)
7
8 print(bundle.serialize(pretty=True))
9
```

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

```
from stix2 import Bundle, Malware
malware = Malware(is_family=False)
elements = [malware]
bundle = Bundle(elements)
9 print(bundle.serialize(pretty=True))

python template.py
{
  "type": "bundle",
  "id": "bundle--66ec2ec5-d80a-4cb4-8aaa-461676a3ac24",
  "objects": [
    {
      "type": "malware",
      "spec_version": "2.1",
      "id": "malware--cdf4e395-6269-4288-a727-dab9fd2bddff",
      "created": "2023-10-29T08:54:53.060888Z",
      "modified": "2023-10-29T08:54:53.060888Z",
      "is_family": false
    }
  ]
}
```

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