Análisis del Malware Tema 2

Sandboxes

La idea de las sandbox es poder analizar dinámicamente el malware en un entorno aislado. Estas Herramientas registran toda la actividad del malware y generan un reporte. Para esto también se puede usar un KVM (Kernel Virtual Machine)que nos permite arrancar una máquina virtual para analizar automáticamente la actividad de un malware, generar un reporte y recuperar la máquina a una snapshot anterior.

Cuckoo Sandbox

Inicia varias máquinas virtuales conectadas a una red virtual para analizar el comportamiento de un malware. Lo primero que se debe hacer tras instalar Cuckoo es configurar la aplicación, para ello vamos a la carpeta conf:

- cuckoo.conf: contiene la config de la VM, permite elegir entre KVM y virtualbox
- virtualbox.conf: configuración que usará el virtualbox como la plataforma y la ip
- kvm.conf: lo mismo que virtualbox.conf pero para kvm
- reporting.conf: Para definir el formato del reporte

Para inicializar cuckoo primero se debe instalar el agente en la máquina virtual (agent/agent.py). Finalmente se debe realizar una snapshot con el agente arrancado, de forma que cuckoo pueda restaurar la máquina virtual tras cada análisis. Para iniciar cuckoo se debe ejecutar el script cuckoo.py en el host. Tras eso envías el archivo que se quiere analizar con submit.py, que se encuentra en la carpeta utils. Se pueden seleccionar los formatos con el flag -package.

Los resultados del análisis se guardan en la carpeta storage/analysis:

- files: Archivos creados cuando se ejecutó el malware
- logs: Actividad del malware como llamadas a librerías del sistema dll
- reports: reporte con el formati anteriormente mencionado

Cuckoo tiene algunas limitaciones ya que al volverse muy popular algunos malwares traen contramedidas como sistemas anti análisis. Lo que hacen estos sistemas es:

- Revisar la resolución de pantalla
- Revisar el número de núcleos disponibles
- Revisar el historial del navegador (Suele estar vacío en las máquinas virtuales)

Debido a esto cuckoo está en constante evolución, al igual que el malware.

REMnux

Kit de herramientras para hacer ingeniería inversa y revisar software malicioso. Provee de una colección de herramientras creadas por la comunidad que se pueden usar para el análisis del

malware. También ofrece imágenes de docker con herramientas de análisis de malware populares.

FlareVM Sandbox

Colección de scripts que permiten montar y mantener un ambiente de ingeniería inversa en una máquina virtual. Depende de 2 principales tecnologías:

- Chocolatey: Sistema de gestión de paquetes para windows donde un paquete es un archivo zip que contiene un script de instalación de PowerShell que descarga y configura herramientas específicas.
- Boxstarter: Usa los paquetes de chocolatey para automatizar la instalación de software y crear ambientes repetibles automatizados.

Contramedidas del Malware

El malware puede tener numerosas medidas contra la ingeniería inversa:

- Ofuscación
- Ocultar información de configuración
- Encriptación de la comunicación de red
- Codificación de datos.

Ofuscación

Transformar el programa dejando las funcionalidades intactas para dificultar el entendimiento de su funcionamiento. Normalmente la ofuscación se usa para:

- Proteger propiedad intelectual: para evitar el plagio
- Hacer los antivirus menos efectivos: evita la detección de malware por firma digital
- Ralentizar el análisis de malware: Para mantener el código del malware más tiempo en funcionamiento.

Codificación de datos

Cifrado simple

Hay muchos cifrados simples que aplican operaciones de bit sobre los registros de datos:

- ADD y SUB: Añade o elimina caracteres del set de caracteres
- ROL y ROR: Rota hacia la derecha o la izquierda los bots de ciertos sets de caracteres
- ROT: Intecambia 13 veces cada caracter del alfabeto
- XOR

Algoritmos standar de ciptografía

La forma más fácil de descubrir el algoritmo es usando uno estándar:

- Mirar los strings e imports para detectar el cifrado de APIs
- Detectar el uso de constantes mágicas de encriptado.
- Analizar la entropía de un archivo para detectar partes que han sido comprimidas o cifradas.

Custom Encoding

Sistemas de codificación caseros:

- Combinar varias capas de multiples metodos de codificados
- Algoritmos completamente customizados creados por el autor del malware.

Reverse Encoding

Procedimiento estándar para encontrar que codificado se ha utilizado y deducir como descifrarlo:

1. Trazar la ejecución del programa buscando funciones de codificado y decodificado
2. Deducir cuando y como se usan estas funciones.
3. Usar el malware contra sí mismo: reprogramar las funcioanes, usarlas como son en el malware…

Usando el malware contra sí mismo

Arrancar el malware en un debugger y establecer breakpointas antes y despues del codificado o decodificado. Tiene sus problemas:

- Puede que el malware no desencripte la información que nos interesa
- No se puede deducir como hacer que el malware ejecute la función de desencriptado

Empaquetado del malware

Los diseñadores de malware suelen añadir un componente llamado run-time packer que son aplicaciones que comprimen el código como otras aplicaciones típicas como Zip o RAR. Cuando estas aplicaciones son descomprimidas con el empaquetador, la aplicación será descomprimida en la memoria de sistema en vez de en el sistema de archivos. La ventaja de esto es que el código de un malware es más pequeño y menos detectable, lo que dificulta su detección por parte de antivirus. Otra medida que se puede tomar es encriptar el malware con el empaquetador, de forma que ni los antivirus puedan desempaquetar el código.

Procedimiento

1. Descomprimir el código empaquetado

2. cargar el código empaquetado en memoria
3. Resolver las importaciones del ejecutable original
4. Transferir la ejecución al OEP (Original entry point.)

Normalmente se realizan tareas anti análisis en cada paso.

Empaquetando el Malware

En los empaquetadores, un estub es una pieza de código que contiene la rutina de decompresión o descifrado que actua de cargador y se ejecuta antes de ejecutar el malware.

Detección

Hay varios indicadores de que podemos estar tratando con un ejecutable empaquetado:

- PE Header: Diferentes tamaños entre datos y tamaño virtual
- Acceso Limitado a los strings e imports
- Entropía: tiene una entropía alta

Desempaquetando el código

Si este fuera un caso fácil, para desempaquetar el código pueden usarse métodos como XOR, AES, etc... A veces se mantiene la tabla de importación intacta de forma que el cargador de windows puede resolver los imports. Si fuera un caso difícil no hay imports y hacerlo manualmente es un dolor de cabeza.

Desempaquetado automatizado

Existen herramientas como UPX que permiten realizar los desempaquetados, pero no siempre sirven. A veces se crean scripts o herramientas propias para empaquetar, lo que hace difícil su desempaquetado. El desempaquetado automatizado no siempre funciona debido a esto.

Cuando falla el desempaquetado automatizado

La dificultad y tiempo requeridos depende de:

- El empaquetador
- Las opciones de empaquetado
- El método de desempaquetado
- La presencia de múltiples niveles de empaquetado.

En estos casos se pueden aplicar 2 métodos:

- Crear una herramienta para deshacer cada paso realizado por el empaquetador y hacer ingeniería inversa del proceso (nadie lo hace)

- Arrancar el programa empaqutado para encontrar el OEP y dimpear todo el proceso desde memoria (lo común)

El proceso

1. Encontrar el punto de entrada original (OEP): La dirección de memoria donde el programa comienza es movido en la sección empaquetada. El analista tiene que recuperar el archivo original
2. Reconstruir la tabla de direcciones importadas (IAT): Esta referencia las funciones usadas por el programa disponibles en la API de windows. Durante la ejecución, el IAT es resuelto dinámicamente por el malware y lo que hace mucho más difícil trazar el código.

Encontrando el OEP: Saltos

Buscar por instrucciones de salto (JMP) al final de cada sección de código:

- Salto del stub de desempaquetado al OEP
- Muy visible en Stubs de desempaquetado simples

El OEP puede estar en diferentes secciones de un stub de desempaquetado:

- Establecer breakpoints en otras secciones
- La ejecución de código se parará en el momento en que cambies de sección

Encontrando el OEP: PUSHA/POPA

- PUSHA es utilizado para guardar registros de estado al principio del stub
- POPA se usa para restaurar el estado del registro al final del stub
- Se pone un breakpoint de memoria en uno de los registros de valor del stack
- Se romperá la ejecución del código cuando POPA cambie el valor
- Normalmente este código suele estar cerca del OEP.

Encontrando el OEP: Llamadas a la API

Bloquear la tabla de imports nos acerca al final del stub:

- Breakpoint en LoadLibrary/GetProcAdress
- Buscar cerca

Buscar llamadas API de Windows como GetModuleHandle o GetCommandLine:

- Rompe una de estas funciones
- OEP suele estar encima del pariente.

Bloqueando Imports

- El stup de empaquetado actua como un proxy que intercepta cada llamada a la API. La llamada

se traduce de la API al stub y ejecutada por una payload
- Se debe evitar hacerlo a mano.

Antianálisis

Técnicas usadas por diseñadores de malware para evitar que este sea detectado usando desensamblado, Debug, Máquinas virtuales o antivirus.

Anti-Desensamblado

Muchos desensambladores son lineares (Procesan una instrucción a la vez) y orientados al flujo (Examina cada instrucción y construyen una lista de localizaciones para desensamblar considerando saltos, llamads, etc..)

El malware se aprovehca de la heurística del desemsamblado, explotando las cosas que da por hecho un ensamblador. El malware introduce código para confundir el análisis de frame del stack.

Anti-Debugging

El malware puede usar el API de Windows para detectar si está siendo debuggeado:

- isDebuggerPressent: revisa el ambiente de bloque de proceso (PEB)
- CheckRemoteDebuggerPresent: Busca por un proceso en la máquina local
- NtQueryInformationProcess: Puede ser usado para revisar si se está ejecutando un debugger
- OutputDebugString: Envía un string al debugger para mostrarlo.

El malware puede hacer revisiones manuales:

- Revisa el flag BeingDebugged en 0x02 en el PEB
- Revisa el flag ProcessHeap en 0x18 en el PEB
- Revisa el flag NTGlobal localizado en 0x68 en el PEB.

El malware también puede buscar por residuos del sistema:

- Claves de Registro:
- Buscar en el sistema por archivos o directorios especificos
- Buscar en el proceso actual una lista para detectar debuggers populares
- Usar FindWindow para localizar un debugger

El malware puede buscar su propio código para detectar interrupts (INT 3, breakpoint o 0xCC). Esto puede ser evitado usando breakpoints de hardware usando registros (D0-D3), pero el malware también puede detectarlos. También puede detectar el debugger ya que cuando está en debug corre más lento. El malware puede contar Ticks usando QueryPerformanceCounter, GetTickCount o la instrucción rdtsc.

Anti-Máquina Virtual

El malware puede usar herramientas como scoopyNG para detectar que está en una máquina virtual. También puede detectar que está en una máquina virtual ya que hay instrucciones con semántica diferente en una máquina virtual que en hardware real. Por suerte gran parte del código anti máquina virtual puede ser parcheado fácilmente, pero el malware puede aprovechar exploits en la máquina para escapar al host.

Anti-Antivirus

Un antivirus realiza la siguientes operaciones detectables:

- Antes de la ejecución: Revisa hashes y firmas, usa un escaneo de emulador y análisis de código heurístico.
- Durante la ejecución: Pone anzuelos en las librerías del sistema, monitoriza el kernel, la red y el sistema de archivos.

El Malware puede usar las mismas estrategias que se usa para el anti-análisis par contrarrestar un antivirus:

- Revisa por procesos de antivirus o si ha sido instalado en una máquina virtual
- Examina las librerías para detectar los anzuelos
- Se usa ofuscación de código para dificultar la detección
- Usa empaquetadores para cambiar su firma hash.

Detección y Eliminación

From:
https://knoppia.net/ - **Knoppia**

Permanent link:
https://knoppia.net/doku.php?id=mwr:tema2&rev=1729524903

Last update: **2024/10/21 15:35**