Guía de Laboratorio - Análisis Forense de Memoria RAM

Taller Informática Forense y IR

1. Introducción y Objetivos

El objetivo de esta práctica es ejecutar un análisis forense digital sobre un volcado de memoria RAM (memory dump) para identificar indicadores de compromiso (IoCs) y reconstruir las acciones del usuario.

Al finalizar esta guía, el estudiante será capaz de:

1. **Identificar el entorno:** Determinar la versión del sistema operativo y arquitectura.
2. **Analizar procesos:** Distinguir entre procesos legítimos y anómalos, comprendiendo la jerarquía de ejecución.
3. **Recuperar artefactos volátiles:** Extraer comandos de consola y reconstruir datos visuales que nunca fueron guardados en disco.
4. **Comprender la estructura de memoria:** Diferenciar entre listas de procesos activas y escaneos de estructuras en el pool de memoria.

2. Preparación del Entorno de Trabajo

Para garantizar la reproducibilidad del análisis, se deben definir las variables de entorno y rutas de acceso de manera agnóstica al sistema operativo del analista.

Requisitos Previos:

- Intérprete Python 3 instalado.
- Framework Volatility 3 configurado.
- Archivo de evidencia: `MemoryDump_Lab1.raw`

Convención de Rutas:

En esta guía utilizaremos los siguientes marcadores de posición. El estudiante debe sustituirlos por las rutas reales en su estación de trabajo:

- `<RUTA_VOLATILITY>`: Directorio donde reside el script `vol.py`.
- `<RUTA_DUMP>`: Ruta absoluta o relativa al archivo de evidencia `.raw`.

Inicialización (Ejemplo en Linux/macOS):

```
cd <RUTA_VOLATILITY>
source venv/bin/activate
# Verificación de ejecución
python3 vol.py -h
```

3. Fase 1: Identificación del Perfil del Sistema (`windows.info`)

Antes de interpretar cualquier estructura de datos, es imperativo conocer la versión del Kernel y la arquitectura del sistema operativo objetivo. Volatility 3 utiliza tablas de símbolos intermedios para mapear las estructuras de memoria correctamente.

Comando:

```
python3 vol.py -f <RUTA_DUMP> windows.info
```

Explicación :

El plugin `windows.info` localiza el **KDBG (Kernel Debugger Data Block)**. Esta estructura es mantenida por el kernel de Windows para fines de depuración y contiene referencias vitales como la lista de procesos cargados (`PsLoadedModuleList`) y la versión del sistema (`NtBuildNumber`).

Análisis de Resultados:

- **SystemTime:** Registre la fecha y hora del sistema (UTC). Este será el "Punto Cero" para la línea de tiempo forense.
- **Layer/Architecture:** Confirme si el sistema es x86 o x64 para entender cómo se direccionan los punteros de memoria.

4. Fase 2: Análisis de Ejecución y Procesos

En esta fase, determinaremos qué programas estaban activos en el momento de la captura. Utilizaremos dos enfoques complementarios para validar la integridad de la lista de procesos.

4.1. Jerarquía de Procesos (`windows.pstree`)

Este plugin visualiza la relación padre-hijo entre procesos, lo cual es fundamental para entender el origen de una ejecución (ej. un `cmd.exe` lanzado por `explorer.exe` es normal; uno lanzado por `svchost.exe` es sospechoso).

Comando:

```
python3 vol.py -f <RUTA_DUMP> windows.pstree
```

Hallazgos Esperados:

Identifique la siguiente cadena de ejecución anómala:

1. `cmd.exe`**:** Indicativo de ejecución de comandos por consola.
2. `mspaint.exe`**:** Indicativo de manipulación de imágenes.
3. `WinRAR.exe`**:** Indicativo de compresión de archivos. En contextos forenses, el uso de archivadores suele preceder a la exfiltración de datos (Data Staging).

4.2. Validación Cruzada (`windows.psscan`)

Mientras `pstree` y `pslist` recorren la lista doblemente enlazada de procesos activos (`ActiveProcessLinks`), un malware avanzado podría desenlazarse de esta lista para ocultarse (técnica DKOM - Direct Kernel Object Manipulation).

Comando:

```
python3 vol.py -f <RUTA_DUMP> windows.psscan
```

Explicación :

El plugin `psscan` no confía en las listas del kernel. En su lugar, realiza un "Pool Scanning", buscando firmas de bytes que coincidan con la estructura `_EPROCESS` en toda la memoria física. Si un proceso aparece en `psscan` pero no en `pstree`, es un fuerte indicador de ocultamiento (Rootkit). En este laboratorio, ambos resultados coinciden, descartando técnicas de ocultamiento avanzadas para estos procesos.

5. Fase 3: Análisis de Línea de Comandos (`windows.cmdline`)

Una vez identificado el proceso sospechoso (`WinRAR.exe`), es necesario determinar sus argumentos de ejecución para entender su propósito.

Comando:

```
python3 vol.py -f <RUTA_DUMP> windows.cmdline
```

Explicación :

Cuando se inicia un proceso en Windows, sus argumentos se almacenan en el **PEB (Process Environment Block)**, una estructura de datos en modo usuario. Este plugin extrae y decodifica estas cadenas Unicode.

Análisis Forense:

Localice el PID asociado a WinRAR.exe.

- **Argumento:** `... "C:\Users\Alissa Simpson\Documents\Important.rar"`
- **Conclusión:** Se confirma la creación de un archivo comprimido con nombre sensible ("Important"), corroborando la hipótesis de preparación para exfiltración.

6. Fase 4: Búsqueda de Artefactos de Archivos (`windows.filescan`)

Para validar la existencia del archivo `Important.rar` en la memoria (incluso si no se escribió en disco o si el disco no está disponible), utilizamos el escaneo de objetos tipo archivo.

Comando:

```
python3 vol.py -f <RUTA_DUMP> windows.filescan | grep "Important.rar"
```

(Nota: En Windows, puede requerir `findstr` en lugar de `grep`, o simplemente volcar la salida a un archivo de texto y buscar manualmente).

Explicación :

El plugin `filescan` recorre la memoria buscando estructuras `_FILE_OBJECT`. Esto permite ver archivos que estaban abiertos o mapeados en memoria en el momento del volcado, proporcionando la ruta completa y los permisos de acceso.

7. Fase 5: Análisis de Memoria Gráfica (`windows.memmap`)

Dado que detectamos `mspaint.exe`, existe la posibilidad de recuperar la imagen visualizada volcando su espacio de memoria asignado.

7.1. Volcado de Memoria (Memory Dump)

Extraeremos el espacio de direccionamiento virtual del proceso Paint.

Comando:

```
# Sustituya <PID_MSPAINT> por el ID numérico observado en la Fase 2 (ej. 2424)
python3 vol.py -f <RUTA_DUMP> windows.memmap --pid <PID_MSPAINT> --dump
```

Resultado: Se generará un archivo `pid.<PID>.dmp`.

7.2. Reconstrucción Visual (GIMP)

Los volcados de memoria carecen de encabezados de formato de imagen (headers). Se trata de datos "crudos" (Raw Data).

Procedimiento en GIMP:

1. **Archivo > Abrir:** Seleccione el archivo `.dmp`.
2. **Tipo de Imagen:** Seleccione "Raw Image Data".
3. **Configuración de Decodificación:**
   - **Image Type:** RGB Alpha (32-bit). Windows 7 suele utilizar 4 canales para gráficos en memoria.
   - **Width (Ancho):** Este es el parámetro crítico. Pruebe resoluciones estándar (1280, 1024, 1920) o ajuste el deslizador hasta que patrones visuales (como líneas verticales) se alineen coherentemente.
   - **Offset:** Desplace el inicio de lectura para omitir las secciones de código del programa y llegar al heap donde se almacenan los datos de la imagen (bitmap).

8. Fase 6: Análisis de Cadenas y Esteganografía (`strings`)

En ocasiones, los plugins de historial de consola (`windows.consoles`) fallan debido a diferencias en la estructura interna de `conhost.exe` entre versiones de Windows. La técnica de "Búsqueda de Cadenas" es el método de respaldo universal.

Comando:

```
strings -e l <RUTA_DUMP> | grep "Zmxh"
```

- `-e l`**:** Filtra cadenas en formato Little-Endian (UTF-16LE), que es la codificación nativa de Windows.
- `Zmxh`**:** Es la representación en Base64 de la palabra "flag".

Decodificación:

El análisis forense requiere decodificar los hallazgos ofuscados.

```
echo "<CADENA_ENCONTRADA>" | base64 -d
```

Concepto Técnico:

El comando `strings` escanea el archivo binario buscando secuencias de caracteres imprimibles consecutivos. Al no depender de estructuras del sistema operativo, es capaz de encontrar datos residuales en memoria no asignada o liberada.

9. Conclusión del Análisis

El análisis del volcado de memoria ha permitido corroborar la siguiente cadena de eventos:

1. El usuario ejecutó comandos de consola.
2. Se utilizó software de edición gráfica (`mspaint`) para manipular evidencia visual.
3. Se empaquetó información sensible mediante `WinRAR` en un archivo `Important.rar`.
4. Se identificaron cadenas codificadas en Base64 residentes en memoria.

Este procedimiento demuestra la volatilidad de la evidencia digital y la importancia de la adquisición de memoria RAM como paso prioritario en la respuesta a incidentes.