Cómo se ejecuta un malware: conceptos y técnicas

"Threat Hunting de Sliver C2"

El análisis de malware implica entender cómo el software malicioso se ejecuta en un sistema, cuáles son sus comportamientos y técnicas de evasión, y cómo se puede descompilar o analizar para obtener más detalles.

Los "stages" (etapas) de los payloads generalmente se refieren al proceso gradual en el que un atacante ejecuta código malicioso dentro de un sistema comprometido. Los payloads pueden estar diseñados en etapas para evitar detección, incrementar su efectividad o reducir el tamaño inicial del código malicioso.

Sliver es un framework de comando y control (C2) de código abierto desarrollado por Bishop Fox, diseñado para ser utilizado en ejercicios de Red Teaming y actividades de simulación de amenazas. Fue creado como una alternativa moderna a otros frameworks como Cobalt Strike y Metasploit, con características que permiten ejecutar ataques avanzados y simular a actores de amenazas reales.

Stage 0: Initial Payload Delivery

- **Objetivo:** Entregar el payload inicial.
- **Descripción:** Esta es la fase en la que el atacante entrega el primer código malicioso al objetivo. Este código suele ser pequeño, indetectable y diseñado para preparar la ejecución del siguiente paso. Puede entregarse mediante correo electrónico, exploits en la web, vulnerabilidades de software, etc.
- **Ejemplo:** Un archivo adjunto en un correo phishing con macros maliciosas o un enlace que lleva a la descarga de un archivo ejecutable.

Stage 1: Dropper/Loader

- **Objetivo:** Descargar o cargar el payload real.
- **Descripción:** En esta etapa, el payload inicial (dropper) descarga o extrae de manera discreta el payload principal (también llamado "second-stage payload") desde un servidor remoto o extrae un código embebido. Aquí, el malware ya puede establecer cierta persistencia mínima.
- **Ejemplo:** El dropper descarga un archivo más pesado o más peligroso desde un servidor de comando y control (C2) o extrae el código encriptado que ya estaba dentro del dropper.

Stage 2: Persistence & Execution

- **Objetivo:** Establecer persistencia en el sistema.
- **Descripción:** El payload de segunda etapa asegura que el malware se ejecute cada vez que el sistema se inicie o que el atacante mantenga acceso continuo. Puede modificar claves de registro, crear servicios, o añadir tareas programadas para asegurarse de que persiste tras reinicios o desconexiones.
- **Ejemplo:** Un troyano puede instalarse como un servicio de Windows, o modificar un script de inicio para ejecutarse automáticamente.

Stage 3: Command and Control (C2) Setup

- **Objetivo:** Establecer comunicación con el servidor de control.
- **Descripción:** El malware comienza a comunicarse con su servidor C2 para recibir más instrucciones, exfiltrar datos o descargar nuevos módulos de ataque. La comunicación puede estar encriptada para evitar la detección.
- **Ejemplo:** El malware establece una conexión HTTPS con un servidor remoto para recibir comandos adicionales.

Stage 4: Payload Activation

- **Objetivo:** Ejecutar las acciones maliciosas.
- **Descripción:** Aquí es donde el malware realiza su propósito final. Puede ser desde exfiltración de datos, cifrado de archivos (ransomware), control remoto del sistema (RAT), instalación de rootkits, hasta la expansión lateral por la red.
- **Ejemplo:** Un ransomware cifra los archivos y muestra una nota de rescate, o una herramienta de control remoto permite al atacante manipular el sistema a voluntad.

Stage 5: Covering Tracks & Evasion

- **Objetivo:** Ocultar la actividad maliciosa.
- **Descripción:** En esta etapa, el malware intenta cubrir sus huellas y evitar la detección. Puede eliminar logs, deshabilitar antivirus, o realizar técnicas de ofuscación. También es común que el malware desinstale algunos de sus componentes una vez completada la actividad maliciosa.
- **Ejemplo:** Un malware puede borrar archivos temporales, deshabilitar servicios de seguridad o alterar los registros para eliminar evidencias.

Stage 6: Lateral Movement (Opcional)

- **Objetivo:** Expandirse dentro de la red.
- **Descripción:** Si el ataque está diseñado para comprometer más de un sistema, el malware puede propagarse por la red interna de la organización buscando otros sistemas vulnerables o credenciales que le permitan expandirse.
- **Ejemplo:** Un exploit busca vulnerabilidades en otros equipos de la red y lanza nuevos ataques en sistemas comprometidos, robando credenciales o ejecutando exploits como en el caso de EternalBlue.

Stage 7: Post-Exploitation & Data Exfiltration

- **Objetivo:** Extraer información o controlar el sistema.
- **Descripción:** Después de comprometer el sistema, los atacantes pueden realizar acciones post-explotación como la exfiltración de datos sensibles, o el uso del sistema comprometido como base para lanzar más ataques.
- **Ejemplo:** Un troyano captura datos sensibles (como información financiera) y los envía a un servidor remoto controlado por los atacantes.

Sliver: Framework de Comando y Control (C2)

Características de Sliver

Sliver destaca por tener un diseño modular y ser altamente configurable, lo que lo hace una herramienta versátil tanto para profesionales de seguridad como para atacantes malintencionados. Algunas de sus características más relevantes son:

- **Soporte Multiplataforma**: Sliver puede ser usado en Windows, Linux y macOS.
- **Implantes (Beacons) en Go**: El implante de Sliver está escrito en el lenguaje Go, lo que facilita la generación de payloads con múltiples configuraciones, soporta evasión avanzada y reduce las firmas detectables por soluciones de seguridad.
- **Evasión Anti-Evasión**: Incluye técnicas integradas para evadir análisis estático y dinámico, como ofuscación y cifrado de payloads.
- **Comunicación Segura**: Utiliza canales de comunicación cifrados con soporte para HTTP/HTTPS, DNS, y mTLS (mutual TLS).
- **Módulos Extensibles**: Permite agregar módulos personalizados para ejecutar tareas específicas.

- **Integración con BYOB (Build Your Own Botnet)**: Sliver puede integrarse con otros frameworks y herramientas personalizadas.

Arquitectura de Sliver

La arquitectura de Sliver está basada en un **servidor de C2** que controla los **implantes** (beacons) distribuidos en los sistemas objetivo. Se compone de los siguientes elementos:

- **Servidor (C2 Server)**: Punto central de comando y control que recibe conexiones de los implantes y les envía comandos.
- **Implante (Beacon/Agent)**: Payload que se ejecuta en la máquina víctima y se comunica con el servidor para recibir comandos y exfiltrar información.
- **Cliente CLI o GUI**: Consola de administración para interactuar con los implantes y gestionar la infraestructura C2.

Componentes de Sliver

- **Payload Generators**: Genera implantes para distintas plataformas, soportando técnicas de evasión como **sleeping**, **obfuscation**, y **cryptography**.
- **Listeners**: Canales de comunicación que el servidor utiliza para recibir información de los implantes. Los listeners pueden usar HTTP, DNS, o mTLS.
- **Modules**: Plugins y funcionalidades que extienden el C2, incluyendo post-explotación, escalación de privilegios, y más.

Técnicas de Evasión

Sliver implementa varias técnicas de evasión para mantenerse indetectable:

- **Ofuscación de Payloads**: Los payloads generados por Sliver están ofuscados para evitar su detección por herramientas de análisis estático.
- **Comportamiento Dormido (Sleeping)**: El implante puede "dormir" durante un tiempo configurado, simulando un comportamiento legítimo y evitando su identificación en entornos de sandbox.
- **Evasión de Debuggers y Sandboxes**: Utiliza técnicas para detectar y evadir debuggers, sandboxes y entornos de análisis como detección de procesos de análisis comunes y chequear artefactos de máquinas virtuales.
- **Manejo y Uso Básico:** Para usar Sliver, primero debes instalar el servidor y luego interactuar con él a través de la CLI o GUI.

Ejecución y Post-Explotación

Después de que el implante esté en ejecución, se puede interactuar con él para ejecutar comandos en el sistema víctima, realizar movimientos laterales, e incluso desplegar otros payloads o ejecutar scripts en PowerShell o Bash.

Integración con Tácticas, Técnicas y Procedimientos (TTPs)

Sliver permite a los atacantes y equipos de red teaming simular TTPs de actores de amenaza reales. Por ejemplo, se puede configurar un beacon para enviar tráfico malicioso a través de DNS para evadir filtros de firewall o utilizar mTLS para establecer comunicaciones seguras con el servidor de C2.

Ciclo de Vida de Ejecución de un Malware

Un malware sigue generalmente un ciclo de ejecución que se puede dividir en las siguientes etapas:

a) **Infección inicial**: El malware llega al sistema a través de vectores como correos electrónicos, sitios web comprometidos, exploits, entre otros.

b) **Ejecución**: Una vez que el malware se ejecuta, puede usar diversas técnicas para cargarse en la memoria y comenzar su actividad.

c) **Persistencia**: Si el malware necesita mantenerse en el sistema, creará mecanismos de persistencia como claves de registro, tareas programadas, o loaders adicionales.

d) **Comunicación**: Algunos malware se conectan a servidores de comando y control (C2) para recibir instrucciones o exfiltrar información.

e) **Evasión y anti-análisis**: Utilizan técnicas para evitar la detección y dificultar el análisis.

f) **Payload**: El payload realiza la función principal del malware, como robar información, cifrar archivos, o instalar componentes adicionales.

Técnicas Comunes en Malware

Empaquetadores y Obfuscadores

Los empaquetadores son programas que comprimen y cifran el malware con el objetivo de reducir su tamaño y dificultar el análisis. Los obfuscadores, por su parte, alteran la estructura y el flujo del programa para esconder la verdadera intención del malware.

- **Ejemplos de empaquetadores**: UPX, ASPack, Themida, VMProtect.
- **Ejemplos de obfuscadores**: ConfuserEx, Obfuscator-LLVM (para binarios), Babel (para JavaScript).

Funcionamiento: El empaquetador genera un archivo ejecutable que contiene el código comprimido o cifrado del malware, además de un pequeño stub (fragmento de código) que, al ejecutarse, desempaqueta o descifra el malware en la memoria y lo ejecuta.

Anti-análisis: Estos empaquetadores también pueden tener funciones anti-debugging o anti-VM, que pueden detectar si el malware se está ejecutando en un entorno de análisis y alterar su comportamiento.

Desempaquetado y Descompresión

El desempaquetado es el proceso de quitar la capa de protección del malware para obtener el binario original. Existen herramientas como **UPX** (para desempaquetar archivos empaquetados con UPX) o **OllyDbg** y **x64dbg** para realizar desempaquetado manual:

- **Identificación del empaquetador**: Usa herramientas como *PEiD*, *Exeinfo PE*, o *Detect It Easy (DIE)* para identificar el empaquetador.
- **Uso de herramientas específicas**: Si el empaquetador es reconocido y hay herramientas disponibles, usa programas como upx -d <file.exe> para desempaquetarlo.
- **Desempaquetado manual**:
    - Carga el binario en un debugger como OllyDbg.
    - Establece un punto de interrupción en la primera instrucción de la sección .text.
    - Desensamblar las instrucciones hasta encontrar el jmp o call que transfiere el control al código desempacado.
    - Dump del binario desempaquetado desde la memoria.

Descifrado y Recuperación de Strings

Muchos malware cifran sus cadenas de texto (strings) para esconder direcciones URL, comandos, nombres de procesos, etc. Existen diversas técnicas para descifrar estas cadenas:

- **Extracción de Strings**: Usa strings en Linux o herramientas como BinText o x64dbg para buscar cadenas en binarios.

- **Desensamblado**: Observa el código en el desensamblador (IDA Pro o Ghidra) para identificar rutinas de cifrado.
- **Técnicas de descifrado**: Identifica funciones de cifrado como XOR, Base64, o AES y usa el código para descifrar los strings durante el análisis.

Evasión Anti-Debugging y Anti-VM

El malware moderno emplea técnicas para detectar entornos de análisis y comportarse de manera diferente:

- **Anti-debugging**: Detecta la presencia de debuggers usando funciones como IsDebuggerPresent en Windows.
- **Anti-VM**: Busca artefactos de máquinas virtuales (archivos de VMware, procesos de VirtualBox) y se niega a ejecutar o altera su comportamiento.

Análisis Estático vs. Análisis Dinámico

Existen dos métodos principales de análisis:

1. **Análisis Estático**: Examinar el binario sin ejecutarlo, utilizando herramientas como:
    - IDA Pro: Para desensamblado.
    - Ghidra: Para ingeniería inversa.
    - Cutter y radare2: Alternativas gratuitas.
    - CFF Explorer y PE Studio: Para explorar los headers de PE y entender la estructura del archivo.

Análisis Dinámico: Ejecutar el malware en un entorno controlado para observar su comportamiento, utilizando herramientas como:

- **Cuckoo Sandbox**: Entorno automatizado para análisis de malware.
- **Process Monitor (Procmon)**: Para observar cambios en archivos, registro y procesos.
- **Wireshark**: Para capturar tráfico de red.
- **Regshot**: Para comparar cambios en el registro.

Ejemplo Práctico

Un malware empaquetado con UPX podría seguir estos pasos para analizarse:

1. **Identificar el empaquetador** usando Detect It Easy.
2. **Desempaquetar** el binario con upx -d.
3. **Desensamblar** el binario desempacado en IDA Pro y buscar llamadas API sospechosas como CreateProcess o VirtualAlloc.
4. **Descifrar strings** extrayendo los mismos y aplicando algoritmos de descifrado si están cifrados.

Herramientas Útiles

- **Yara**: Para crear reglas de detección basadas en patrones.
- **Process Hacker**: Monitor de procesos.
- **Sysinternals Suite**: Para monitoreo y manipulación de procesos, registro y red.
- **dnSpy**: Descompilador para .NET.