Análisis del Malware Tema 1

Introducción

El malware se define como un software malicioso que realiza acciones mal intencionadas. Generalmente e busca analizar el malware para asesorar daños, identificar vulnerabilidades, capturar a los “chicos Malos” y responder respuestas.

¿Por que se crea malware?

El primer malware fue un gusano que trataba de medir el tamaño de internet en los 80. El gusano se comportaba como una forkbomb y se propagó de forma increíblemente rápida. En los 90 los virus se hicieron para ganar gloria personal, haciendo que el malware mostrara mensajes en pantalla. En la actualidad se crean para ganari dinero, robar contraseñas, información bancaria o secretos industriales. En el futuro se cree que se utilizarán para guerra cibernética con malware que utilizaría vilnerabilidades de tipo Zero Days con el objetivo de causar daño en instalaciones físicas.

Cuestiones prácticas

- ¿Cual es el objetivo de este malware?
- ¿Como y cuando fui infectado?
- ¿Quien me ha establecido como objetivo?
- ¿Como evitarlo?
- ¿Que han obtenido mediante el malware?
- ¿Es capaz de reproducirse?
- ¿Como lo puedo encontrar en otro lugar?
- ¿Como se previene otra futura infección?

Cuestiones técnicas

- ¿Cuales son los indicadores de red?
- ¿Cuales son los indicadores a nivel de host?
- ¿Persistencia?
- ¿Fecha de compilación?
- ¿Fecha de instalación?
- ¿Leguaje de programación?
- ¿Empaquetado?
- ¿Tiene funcionalidades rootkit?

Términos populares de análisis del malware

- Virus: Código que va unido a una aplicación que busca replicarse en aplicaciones similares hasta que pueda ejecutar una payload.
- Gusano: Malware que se propaga muy rápido a través de la red.

- Troyano: Malware que se camufla como otra aplicación para infectar el sistema.
- Spyware/adware: Malware para espiar o meter publicidad
- Backdoor: Vulnerabilidad dejada de forma deliberada por el fabricante
- Rootkut sniffers: Rootkit que escucha todos los paquetes que pasan por la red.
- Exploit: comandos que toman ventajas de vulnerabilidades del sistema
- Disassembler: Programa que recibe un ejecutable y genera un archivo de texto con el código en del programa en ensamblador.
- Decompiler: Toma un archivo binario y trata de producir código de alto nivel usando este como base
- Debugger: Programa que permite observar el código mientras se ejecuta.
- Sinkhole: host de la red interna que recibe tráfico malicioso redireccionado desde un dominio malicioso.
- Intrusion Detection System (IDS): Sistema de Software/Hardware que trata de detectar uso no autorizado de la red.
- Intrusion Prevention System (IPS): Intentan detener un intruso que se haya colado en el sistema.
- Operations Security (OPSEC): Proceso de prevenir que un adversario obtenga información sensible.
- Ingeniería inversa
- Ransomware: malware que pide rescates para recuperar un sistema
- Creeping
- Phising
- Pharming
- Bloatware
- Doxing
- Flaming

Webs interesantes

Ciberthreat live map

Metas y Tipos de Análisis del Malware

Objetivos del análisis del malware

Se busca obtener un entendimiento de como un malware específico funciona para construir defensas para proteger nuestros sistemas en el futuro.

Tipos de análisis del malware

- Análisis estático: Análisis del código para obtener un mejor entendimiento del malware. No se ejecuta.
- Análisis dinámico: Se analiza como se comporta el malware cuando es ejecutado, observando con que se trata de comunicar y como funciona.

Se deben realizar estos dos tipos de análisis para obtener un entendimiento completo de como

funciona un malware. Aunque ambos tipos consiguen lo mismo, se necesitan diferentes habilidades para realizarlos.

Análisis estático de código

El análisis estático es más seguro ya que no se está ejecutando código malicioso, pero es muy lento y difícil ya que se necesitan muchas herramientas tanto gratuitas como de pago para proceder. cuando se hace ingeniería inversa se deben usar desensambladores, debuggers y compiladores (Cuidado con las leyes ya que en algunos países el uso de estas herramientas se puede considerar piratería)

Análisis dinámico del comportamiento

Es una manera rápida de analizar un malware. Es muy importante que el laboratorio de malware no esté conectado a una red externa. Este tipo de análisis observa como se comporta un malware y que cambios trata de realizar en el sistema. Cuando se haga este análisis se debe estar atento de que cambios han surgido en el sistema, así como si hay comportamiento poco usual por parte del equipo. Cambios que pueden ser indicativos de algo malo:

- Archivos añadidos o modificados
- Nuevos servicios de red instalado
- Nuevos procesos arrancando
- Modificaciones de registro
- Modificaciones de ajustes del sistema
- Cambio en configuraciones de red (DNS)

Además del comportamiento del sistema también se debe analizar el tráfico de red.

Malware Armado: características

- Encriptado: el contenido se oculta encriptándolo.
- Compresión
- Ofuscación: Se trata de dificultar ver que hace el código haciéndolo deliberadamente difícil de entender.
- Anti-parcheo (CRC check): Detecta si se han realizado modificaciones. De forma que el malware es capaz de saber si ha sido manipulado en el proceso de ingeniería inversa.
- Anti-tracing: si detecta que se está tratando de ver función por función que hace el código corta la ejecución.
- Anti-desempaquetado
- Anti-VMware: Detecta si se está ejecutando en una máquina virtual y en ese caso o no se ejecuta o corta la ejecución. Para detectar esto puede mirar la información del sistema o el historial del navegador.
- Self-Mutating (Poli/metamorpic): Puede cambiar de forma o tener una forma diferentes;
- Fechas restrictivas: Si ha pasado cierta fecha el malware no se ejecutará o restringe en que fechas se puede ejecutar.
- Protección por contraseña: Para ejecutarlo se necesita una contraseña.

La arquitectura x86

En un ordenadores existen diferentes capas de abstracción, estando en lo más bajo el hardware, seguido del lenguaje máquina (Binario u octal), el ensamblador (Instrucciones), el kernel del sistema operativo y arriba de todo los programas. Estas capas se podrían separar en 2 tipos: hardware y sofrware.

Ingeniería inversa

En algunos lenguajes existen decompiladores que permiten obtener una descripción de un código a alto nivel. Normalmente el código en ensamblador es la capa más alta de abstracción que puede ser recuperada de forma fiable y consistente. Entender ensamblador es muy importante para el análisis de malware. El lenguaje ensamblador depende de la familia de procesadores (X86, MIPS, DEC Alpha, ARM, PowerPC...)

La arquitectura x86

Es la arquitectura más atacada ya que es la más popular. La mayoría del mercado actual esta tomado por equipos basados en la arquitectura x86. Ejemplo de instrucción en ensamblador y en lenguaje máquina:

```
mov ecx, 0x42 -> B9 42 00 00 00
```

- Intrucciones: cada una corresponde a un código de operación que puede realizar el procesador
- Operadores: Se usan para identificar los datos usados por una insturcción. Hay 3 tipos:
  - Operadores inmediatos con valores fijos
  - Operadores de registro
  - Direcciones de memoria en corchetes.

registros

- Registros generales: usados por el procesador durante la ejecución
- Registros de segmento: Usados para mantener localizadas secciones de memoria
- Flags de estado: se usan para tomar decisiones
- Puntero de instrucción: Se usa para localizar cual es la siguiente instrucción

Flags

El registro de flags tiene un tamaño de 32 bits, siendo cada bit un flag, de forma que si es 1, esta activado y si es 0 esta desactivado. Se usan para controlar operaciones del procesador o indicar resultados de operaciones. Tipos de flags:

- ZF(Zero Flags): Cuando el resultado de una operación es 0.
- CF(Carry Flag): Cuando el resultado de una operación es demasiado grande o demasiado pequeño para su destino

- SF(Sing Flag): Cuando el resultado de una operación tiene signo negativo
- TF(Trap Flag): Usado para debug. Cuando está activada el procesador solo ejecuta una instrucción cada vez.

Instrucciones de datos

- mov, destino, origen: mueve datos de un origen a un destino
- lea destino, origen: carga una dirección efectiva a un destino

Instrucciones aritméticas

- add/sub destination, value
- mult/div value
- operadores lógicos or y xor
    - or/ y /xor destino, valor
    - Se usan como una forma rñapido de establecer eax a 0
- shr/shl destino, contador
- ror/rol: instrucciones de rotación.

Llamadas de función

Ejecutan subrutinas para ejecutar algo

- Function prologue: Unas pocas líneas de código al inicio de la función, prepara los stacks/registros para usarlos en la función
- Function epilogue: Unas pocas líneas de código al final de la función, reestablece los stacks/registros a los valores antes de la llamada a la función.

El Stack

Instrucciones: push, pop, call, ret… Convenciones:

- cdecl: llamar a una función vacía el stack
- stdcal: La función llamada vacía el stack antes de terminar
- fastcall: Los primeros argumentos se pasan en los registros, normalmente edx y ecx.

Teniendo una función dada del frame del stack:

- Las variables locales estarán en un offset negativo de EBP.
- Los argumentos de las funciones tendrán un offset positivo.

Condicionales

- test valie, value: es como un AND, activa la ZF Zero Flag
- cmp destination, source: Equivalente a la instrucción sub pero no afecta a los operadores.

branching

- jmp ubicación: una rama es una secuencia de código que es condicionalmente ejecutada para saltar instrucciones.

Repeating

- rep instructions: set de instrucciones para manipular buffers de datos como arrays de bits usando movsx, cmpsx, stosx, scasx, siendo la x = b(byte, 8 bits), w(word, 16 bits), d(double word, 32 bits)