

Analista de Seguridad Informática



Organiza



Enseñanzas Propias



Dpto. Lenguajes y Computación

Julio Gómez López

Capítulo 5. Analista Aplicaciones

Contenido

Introducción

Estudiaremos:

- Crack
- Exploit




Introducción

Hoy en día las palabras PIRATERIA, VIRUS, BUG son mencionadas en noticiarios de todo el mundo, creando debates, leyes y controversia en muchos casos y desconocimiento y mitos en muchos otros. Hay defensores del software libre, también hay empresas y asociaciones que defienden derechos de autor, así como también hay auténticas mafias “cibernéticas” muy especializadas en aprovechar todos los recursos de la informática para llevar a cabo sus fines delictivos.

Capítulo 5. Analista Aplicaciones

Contenido
Introducción
Estudiaremos:

- Crack
- Exploit



Introducción


¿En que punto nos encontramos los profesionales de la informática y como debemos afrontar todo esto?.

La piratería se produce como consecuencia de informáticos capaces de romper los caros diseños de protección en el software comercial, así como de los soportes multimedia de que se dispone hoy en día. En este caso vemos como dos personas con las mismas inquietudes y conocimientos tratan desde lados opuestos de buscar la solución a un problema.

Capítulo 5. Analista Aplicaciones

Contenido
Introducción
Estudiaremos:

- Crack
- Exploit
- NetCat



Introducción

veremos en este apartado como trabajar en este ámbito para evitar ser victimas en la medida de lo posible. Aprenderemos como saltar sistemas de protección en el software, también describiremos como fabrdcar herramientas capaces de darnos control total sobre la víctima. Finalmente veremos como detectar un tipo de vulnerabilidad que afecta a miles de aplicaciones, conocida como Buffer overflow.

Analista de Seguridad Informática

Capítulo 6. Analista de aplicaciones (I)

Capítulo 5. Analista Aplicaciones

Contenido

Introducción

Estudiaremos:

- Crack
- Exploit
- NetCat



Cracks

En este apartado veremos como saltar las protecciones del software comercial que requiera un serial para poder utilizarse. Para ello usaremos la herramienta OllyDebugger.

Se recuerda que estos contenidos son para saber como proteger nuestros desarrollos en la medida de lo posible, sin ser víctimas de crack o keygen.

Capítulo 5. Analista Aplicaciones

Contenido

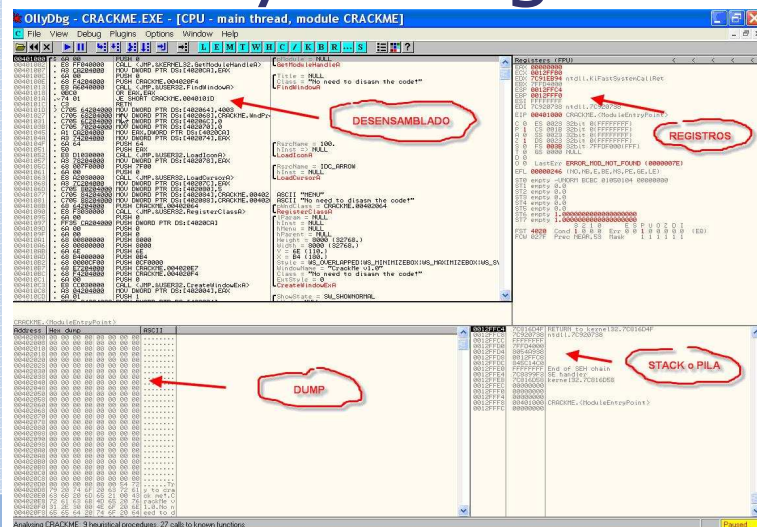
Introducción

Estudiaremos:

- Crack
- Exploit
- NetCat



OllyDebugger



Capítulo 5. Analista Aplicaciones

Contenido
Introduccion
Estudiaremos:

- Crack
- Exploit
- NetCat



OllyDebugger

1) DESENSAMBLADO :
Aquí el OLLY nos muestra el listado desensamblado del programa que vamos a debuggear, por DEFAULT el OLLY viene configurado para analizar el programa que vamos a debuggear al iniciar.

2) REGISTROS
La segunda ventana importante del OLLYDBG es la de los REGISTROS. Recordamos que la ventana de registros se encuentra en la parte superior derecha del OLLYDBG, allí muestra bastante mas información que los registros en si.

Capítulo 5. Analista Aplicaciones

Contenido
Introduccion
Estudiaremos:

- Crack
- Exploit
- NetCat



OllyDebugger

3) STACK O PILA:
Bueno allí vemos el llamado STACK O PILA aquí no hay mucha configuración posible solo la opción de mostrar la información relativa al registro ESP o al registro EBP.

Por default y lo que mas se utiliza es la vista relativa a ESP, pero para cambiar a la vista según EBP, haciendo clic derecho en el stack eligiendo GO TO EBP cambiamos y para volver GO TO ESP volvemos a la opción por default.

Capítulo 5. Analista Aplicaciones

Contenido
Introducción
Estudiaremos:

- Crack
- Exploit
- NetCat



OllyDebugger

4) DUMP:

La ventana del DUMP tiene muchas opciones de visualización, por DEFAULT nos muestra la visualización HEXADECIMAL de 8 columnas o bytes, la cual puede ser modificada haciendo CLICK DERECHO en el DUMP y eligiendo la opción deseada.


Se entrega una guía anexa para ver en profundidad la configuración y posibilidades de este programa.

Ahora pasemos a la práctica.....

Capítulo 5. Analista Aplicaciones

Contenido
Introducción
Estudiaremos:

- Crack
- Exploit
- NetCat



La táctica a seguir será la siguiente:
Intentaremos buscar como alterar el salto condicional que nos lleva de la zona “CHICO MALO”=(serial incorrecto) a la de “CHICO BUENO”=(serial correcto). Los pasos que seguiremos serán los siguientes:

Paso 1. Buscar con el ollydb las STRING REFERENCES.

Paso 2. Buscar con GOTO el lugar en el que se encuentra la zona de “CHICO MALO”.


Paso 3. Localizar las comparaciones y los saltos a las zonas “CHICO MALO” y “CHICO BUENO”.

Paso 4. Modificar con el ollydb los números hexadecimales.

Capítulo 5. Analista Aplicaciones

Contenido
Introduccion
Estudiaremos:

- Crack
- Exploit
- NetCat



Exploit

Los bug, son fallos de programación, algunos de ellos afectan a la seguridad de la aplicación, y se convierten así en una vulnerabilidad que podrá ser usada por los atacantes para desarrollar EXPLOIT (programas que aprovechan ese bug) que acaben por proporcionar, shells remotas, DOS y otros muchas ventajas para atacantes.


Estos fallos de programación se producen por:

- Desconocimiento del programador en metodologías seguras de desarrollo.
- Descuidos del programador.

Capítulo 5. Analista Aplicaciones

Contenido
Introduccion
Estudiaremos:

- Crack
- Exploit
- NetCat




Exploit

- Hackers**
 - Finalidades
 - WhiteHats: para demostrar que la vulnerabilidad es real.
 - GreyHats: para comercializarlos y sacar rédito económico.
 - BlackHats: para perjudicar y/o sacar beneficio personal.
- Crackers**
 - Para destruir o utilizar los sistemas con fines delictivos.
 - Espionaje.
 - Adulterar información.
 - Robo.
 - Placer por destruir
- Consultores especializados**
 - Para auditar y demostrar fehacientemente que los sistemas son vulnerables

Capítulo 5. Analista Aplicaciones

Contenido
Introduccion
Estudiaremos:

- Crack
- Exploit
- NetCat




Exploit

- ¿Que es una shellcode?
Una shellcode es un código básico en ASM, muy corto generalmente, que ejecuta los comandos que queremos, como `system("cmd.exe")` (ejecuta una shell msdos en windows); o `execv("/bin/sh")` (ejecuta una shell sh en Linux/Unix), o sirve para añadir un usuario a la cuenta del sistema, para descargar un troyano y ejecutarlo, para dejar abierto un puerto conectado a una shell, etc.... Es el código que ejecutara el programa vulnerable una vez tengamos su control.

Capítulo 5. Analista Aplicaciones

Contenido
Introduccion
Estudiaremos:

- Crack
- Exploit
- NetCat



Exploit

Estudiaremos el Buffer Overflow o también llamado stack overflow como ejemplo de Bug, aunque existen muchos otros tipos como Format String, Heap Overflow, etc. Un overflow es, básicamente, cuando resguardamos espacio de memoria insuficiente para una variable (allocate), y le introducimos más datos a dicha variable de los que puede soportar. La variable "desborda", y los datos que no caben sobrescriben memoria continua a dicha variable. Si declaramos una variable que solo debe soportar 8bytes, si le movemos 10bytes, los 2bytes restantes no se pierden, sino que sobrescriben la memoria contigua a dicha variable.

Capítulo 5. Analista Aplicaciones

Contenido

Introducción

Estudiaremos:

- Crack
- Exploit
- NetCat

Exploit

- ¿Porque se le llama Stack Overflow?

La pila (stack) es una estructura tipo LIFO, Last In, First Out, ultimo en entrar, primero en salir.

Bien, pues el SO (tanto Windows como Linux, como los Unix o los Macs) se basa en una pila para manejar las variables locales de un programa, los retornos (rets) de las llamadas a una función (calls), las estructuras de excepciones (SEH, en Windows), argumentos, variables de entorno, etc...

Por ejemplo, para llamar a una función que necesite dos argumentos, se mete primero el argumento 2 en la pila del sistema, luego el argumento 1, y luego se llama a la función.

Capítulo 5. Analista Aplicaciones

Contenido

Introducción

Estudiaremos:

- Crack
- Exploit
- NetCat

Si el sistema quiere hacer una suma (5+2), primero introduce el 2º argumento en la pila (el 2), luego el 1º argumento (el 5) y luego llama a la función suma. Bien, una "llamada" a una función o dirección de memoria, se hace con la instrucción ASM Call. **Call dirección** (llamar a la dirección) ó **call registro** (llama a lo que contenga ese registro). El registro EIP recoge dicha dirección, y la siguiente instrucción a ejecutar esta en dicha dirección, hemos "saltado" a esa dirección.

Pero antes, el sistema debe saber que hacer cuando termine la función, por donde debe seguir ejecutando código.

El programa puede llamara la función suma, pero con el resultado, puede hacer una multiplicación, o simplemente mostrarlo por pantalla. Es decir, la CPU debe saber por donde seguir la ejecución una vez terminada la función suma.

Memoria Alta

parametros

ret

ebp

Buffer

Memoria Baja

Analista de Seguridad Informática

Capítulo 6. Analista de aplicaciones (I)

Capítulo 5. Analista Aplicaciones

Contenido

Introducción

Estudiaremos:

- Crack
- Exploit
- NetCat

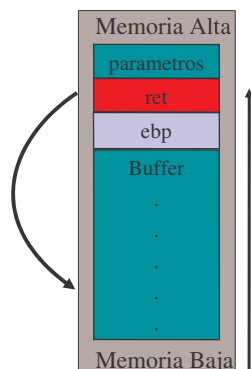


Para eso sirve la pila. Justo al ejecutar el call, se **GUARDA** la dirección de la siguiente instrucción en la pila.

Esa instrucción se denomina normalmente **RET** o **RET ADDRESS**, dirección de "retorno" al programa principal (o a lo que sea).

Entonces, el call se ejecuta, se guarda la dirección, coge los argumentos de la suma, se produce la suma y, como esta guardada la dirección por donde iba el programa, **VUELVE** (RETORNA) a la dirección de memoria que había guardada en la pila (el ret), es decir, a la dirección siguiente del call.

Vamos a verlo por pasos :



Capítulo 5. Analista Aplicaciones

Contenido

Introducción

Estudiaremos:

- Crack
- Exploit
- NetCat



1º Llegamos al call (EIP apunta a la instrucción call).

2º Se ejecuta el call. EIP apunta a la instrucción del call, es decir, donde debemos ir).

3º Se guarda la siguiente instrucción después del call en la pila (el ret).

En ese momento, la pila esta así:

ESP | RET ADDRESS | EBP -8bytes

ESP +4bytes | argumento 1 | EBP -4bytes

ESP +8bytes | argumento 2 | <--- EBP apunta aquí (la base de la pila).

4º La cpu ejecuta la/las instrucciones dentro de la función suma (obviamente, dentro de la función suma se usara la pila para almacenar datos y demás...).

5º La función suma alcanza la instrucción RETN (retorno), y EIP recoge la dirección RET ADDRESS, y vuelve al programa principal, justo después del call suma.

Analista de Seguridad Informática

Capítulo 6. Analista de aplicaciones (I)

Capítulo 5. Analista Aplicaciones

Contenido

Introducción

Estudiaremos:

- Crack
- Exploit
- NetCat



Un stack overflow significa introducir suficientes datos en la pila, hasta poder sobrescribir dicho ret address. Veamos un ejemplo de código:

```
#include <stdio.h>
int main (int argc, char **argv){ // La función "principal" del programa
función
    char buffer[64]; //Declaramos un array con 64 bytes de espacio
    if (argc < 2){ // Si los argumentos son menores que 2...
        printf ("Introduzca un argumento al programa\n");

        return 0; // y retornamos 0 a la función main, y el
        programa acaba
    }

    strcpy (buffer, argv[1]); // Aqui es donde esta el fallo.
    return 0; // Devolvemos 0 a main, y el programa acaba.
}
```

El fallo está en la función strcpy. Esa función copia lo que hayamos metido por argumentos al programa (argv[1]) dentro de la variable buffer. Pero buffer solo tiene espacio para 64 caracteres, no hay ningún chequeo de tamaño de la fuente, y por argumentos al programa le podemos meter lo que queramos.

Capítulo 5. Analista Aplicaciones

Contenido

Introducción

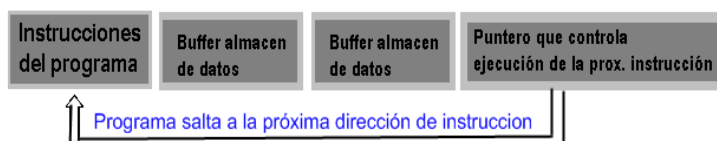
Estudiaremos:

- Crack
- Exploit
- NetCat



ESQUEMA MEMORIA (RAM)

Ejecución normal



Ejecución Hackeada



El Hacker introduce el código que sobre escribe el buffer y lo desborda para corromper el puntero del programa, causando la ejecución del código malicioso.