
18. Desbordamiento de Búfer
Julio Javier Iglesias Pérez
C|EH Julio Iglesias Pérez

Introducción

Un buffer overflow ocurre cuando un búfer que ha sido alojado en un espacio de almacenamiento tiene más datos copiados de los que puede manejar.

Cuando el programa es compilado y ejecutado, asignará un bloque de memoria de longitud 11 bytes para sostener la cadena atacante.


_SHOpenRegStream2A+0081
PROT32
001B:77F7B0E9 PUSH EBX
001B:77F7B0EA LEA ECX,[EBP+14]
001B:77F7B0ED PUSH ECX
001B:77F7B0EE PUSH EBX
001B:77F7B0EF PUSH DWORD PTR [EBP+10]
001B:77F7B0F2 PUSH EAX
001B:77F7B0F3 CALL [__imp__RegQueryValueExA]
001B:77F7B0F9 TEST EAX,EAX
001B:77F7B0FB JNZ ↓77F7B131
001B:77F7B0FD CMP [EBP+08],EBX
001B:77F7B100 JZ ↓77F7B131
001B:77F7B102 PUSH DWORD PTR [EBP+08]
001B:77F7B105 MOV ECX,EDI
001B:77F7B107 CALL CMemStream::GrowBuffer
001B:77F7B10C TEST EAX,EAX
001B:77F7B10E JZ ↑77F7A373
001B:77F7B114 LEA EAX,[EBP+08]
001B:77F7B117 PUSH EAX
001B:77F7B118 PUSH DWORD PTR [EDI+08]
001B:77F7B11B LEA EAX,[EBP+14]
001B:77F7B11E PUSH EAX
001B:77F7B11F PUSH EBX
001B:77F7B120 PUSH DWORD PTR [EBP+10]
001B:77F7B123 PUSH DWORD PTR [ESI]
001B:77F7B125 CALL [__imp__RegQueryValueExA]
001B:77F7B12B MOV EAX,[EBP+08]
001B:77F7B12E MOV [EDI+10],EAX
001B:77F7B131 PUSH 12
001B:77F7B133 CALL _IsOS
001B:77F7B138 TEST EAX,EAX
First Call
Buffer Allocation
Second Call
CEH Julio Iglesias Pérez

Introducción

La función "strcpy" copiará la cadena "DDDDDDDDDDDDDDD" dentro de la cadena atacante, que excederá el buffer 11 bytes de tamaño, resultando en un buffer overflow.

```
<stdio.h>
int main (int argc, char ***argv)
{
char attacker[11]="AAAAAAAAAAA";
strcpy(attacker,"DDDDDDDDDDDDDDD");
prinft("% \n", attacker);
return 0; }
```

¿Por qué los programas y aplicaciones son vulnerables?

- Los controles de límite no se realizan plenamente, en muchos casos, son ignorados completamente.
- Los lenguajes de programación como el C tienen vulnerabilidades en ellos.
- Los programas y aplicaciones no se adhieren a las mejores prácticas de programación.

¿Por qué los programas y aplicaciones son vulnerables?

- Las siguientes funciones pueden ser explotadas por no revisar el tamaño del buffer:
  - strcat()
  - strcpy()
  - sprintf()
  - vsprintf()
  - bcopy()
  - gets()
  - scanf()

Entendiendo las Pilas

- La pila utiliza en mecanismo Último en Entrar, Primero en Salir (LIFO) para pasar los argumentos a las funciones y referir las variables locales.
- Actúa como un buffer, reteniendo toda la información que la función necesita.
- La pila es creada al principio de una función y liberada al final.

Buffer Overflow basado en pila

- Ocurre cuando un buffer ha desbordado el espacio de la pila.
- El atacante inyecta código malicioso en la pila y la desborda para sobrescribir el retorno del puntero de manera tal que el flujo de control cambia al código malicioso.

Entendiendo el Heap

- Heap es un área de la memoria utilizada por una aplicación y es alojado dinámicamente en un tiempo de ejecución con funciones como "malloc()".
- Las variables estáticas son almacenadas en la pila a lo largo con los datos alojados utilizando la interfaz malloc.
- Heap almacena todas las instancias o atributos, constructores y métodos de una clase u objeto.

Buffer Overflow basado en los Heap

- Si una aplicación copia datos sin revisar si no se ajusta dentro de un destino, los atacantes pueden abastecer a la aplicación con datos largos, sobrescribiendo la información de administración de los heap.
- El atacante hace un buffer overflow en la parte más pequeña del heap, sobrescribiendo las otras variables dinámicas, que puede llegar a tener efectos no esperados y no deseados.

Operaciones de Pila

- POP: Quitar un ítem desde la parte superior de la pila.
- -Push: Coloca un ítem en la parte superior de la pila.

Estas operaciones retornar contenido señalado por el puntero y cambia el puntero

  - Puntero de Instrucción Extendida: EIP Apunta al código que esta en ejecución. Cuando se llama a una función, esta se guarda en la pila para uso posterior.

Operaciones de Pila

- Puntero de Pila extendida: ESP apunta a la posición actual en la pila y permite que se puedan agregar y quitar cosas de la pila utilizando las operaciones pop y push o dirige los punteros de manipulación de la pila.
- Puntero de Base Extendida: EBP sirve como punto estático para referenciar información como variables y datos en una función utilizando compensaciones. Esto casi siempre apunta a la parte superior de la pila para una función.

Shellcode

- Es un pequeño código utilizado como payload en la explotación o vulnerabilidad de software.
- Los buffers son blancos blandos para los atacantes y ellos hacen overflow de manera fácil si es que las condiciones están propicias.

Shellcode

- Buffer overflow shellcodes, escritos en lenguaje ensamblador, explotan vulnerabilidades en la pila y en la administración de la memoria heap.

Ejemplo:

"\x2d\x0b\xd8\x9a\xac\x15\xa1\x6e\x2f\x0b\xdc\xda\x90\x0b\x80\x0e"

"\x92\x03\xa0\x08\x94\x1a\x80\x0a\x9c\x03\xa0\x10\xec\x3b\xbf\xf0"

No Operaciones (NOPs)

- La mayoría de los CPUs no tienen la instrucción NOP (no hace nada pero avanza el puntero de instrucción).
- Usualmente, se puede colocar algo de esto por adelantado de su programa (en la cadena). Mientras que la nueva dirección de retorno apunte al NOT, está bien.
- La mayoría de los IDS buscan por las firmas de los NOT sleds (trineos).

No Operaciones (NOPs)

- El atacante rellena el comienzo del buffer overflow previsto con una gran cantidad de ejecución de instrucciones NPP, así el CPU no hace nada asta que llega el "evento principal" (que precede del puntero de retorno).
- ADMutate (por K2) acepta un exploit buffer overflow como entrada y aleatoriamente crea una versión equivalente de funcionalidad (polimorfismo).

Metodología Buffer Overflow

- Entender el tipo de pila y heap del proceso de memoria.
- Conocimiento sobre lenguaje máquina y ensamblador.
- Conocimiento de cómo las llamadas al sistema trabajan a nivel de código maquina.
- Conocimiento de programación en C y Perl.
- Familiaridad con herramientas de compilación y debugger como gdb.

Pasos Buffer Overflow

Paso 1. Encontrar la presencia y locación de la vulnerabilidad buffer overflow.

Paso 2. Escribir más datos de lo que el buffer pueda manejar.

Paso 3. Sobrescribir la dirección de retorno de la función.

Paso 4. Cambiar el flujo de ejecución al código hacker.

Atacando un programa real

- Asumiendo que la función de cadena es explotada, el atacante puede enviar cadenas largas como entrada. Esta cadena hace un overflow al buffer y causa un error de segmentación.
- El punto de retorno de la función es sobrescrita, y el atacante tiene éxito en la alteración del flujo de ejecución.
- Si el usuario ha insertado código en la entrada, él o ella tiene que saber la dirección y tamaño exactos de la pila y hacer un puntero de retorno a su código para la ejecución.

Overflow utilizando cadena de formato

En C, consideramos este ejemplo de BoF utilizando un problema de cadena de formato:

```
char errmsg[512],
outbuf[512];
sprintf (errmsg, "Illegal
command: %400s", user);
sprintf( outbuf, errmsg);
```

Que pasa si el usuario = "%500d <nops> <shellcode>

- Salta la limitación de "%400s"
- Hará un overflow outbuf

Rompiendo la pila

- La idea general es desbordar al buffer así escribe una dirección de retorno.
- Cuando la función está hecha saltará a la dirección cuál sea la dirección en la pila.
- BoF nos permite cambiar la dirección de retorno de una función.
- Colocar algún código en el buffer y colocar la dirección de retorno para que lo apunte.

Una vez que la pila está rota...

Obtener acceso

- Una vez el proceso vulnerable es comandado, el atacante tiene los mismos privilegios que el proceso y puede obtener acceso normal.
- El o ella puede entonces explotar una vulnerabilidad BoF local para obtener acceso de súper usuario.

Una vez que la pila está rota...

Crear un backdoor:

- Utilizando inetd (unix).
- Utilizando Triviál FTP (TFTP) incluido en Windows 2000 y algunos Unix.
- Utilizar Netcat: Para realizar raw y conexiones interactivas
- GUI específica de Unix.
- Lanzar una nueva conexión Xterminal.

Ejemplos BoF

Simple Overflow no controlado

```
#include <sdtlib.h>
#include <sdtio.h>
#include <string.h>
int bof() {
char buffer[8];
strcpy(buffer,"AAAAAAAAAAAAAAAAAAAAAA");
/*copia 20 bytes de A dentro del buffer*/
return 1; /*retorna, esto causar[a una violacion de acceso debido a la corrupcion de la pila*/
int main(int argc, char **argc) {
bof(); /*llama a nuestra funcion*/
/*imprime un mensaje cordo, la ejecucion nunca llegara a este punto debido al overflow*/
printf("Lets go\n");
return 1; /*deja la funcion principal*/ }
```

Ejemplos BoF

Ejemplo de Overflow Heap no controlado

```
/*heap1.c / the manera mas simple de heap overflows*/
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
char *input = malloc (20);
char *output = malloc (20);
strcpy (output, "normal output");
strcpy (input, argv[1]);
printf ("input at %p: %s\n", input, input(;
printf ("output at %p: %s\n", output, output);
printf("\n\n%s\n", output);
}
```

Ejemplos BoF

Buffer Overflow simple en C

```
#include <stdio.h>
main() {
char *name;
char *dangerous_system_command;
name = (char *) malloc(10);
dangerous_system_command = (char *) mallloc(128);
printf("Address of name is %d\n", name);
printf("Address of command is %d\n", dangerous_system_command);
sprintf(dangerous_system_command, "echo %s", "Hello word!");
printf("What is your name?");
gets(name);
system(dangerous_system_command);
}
```

Ejemplos BoF

Explicación

- Lo primero que hace el programa es declarar dos variables string y asignarlas a la memora.
- Se da 10 bytes de memoria a la variable "name" (lo que permite retener una cadena de 10 caracteres)
- A la variable "dangerous_system_command" se le da 128 bytes.

Ejemplos BoF

- En el lenguaje C, los resquicios de la memoria dados a estas variables se encontrarán directamente junto a la otra en el espacio de memoria virtual dado al programa.

Ejemplos BoF

Análisis del código

- El código "gets", que lee una cadena desde la entrada estándar a la locación de la memoria especificada, no tiene una especificación de "longitud".
- Esto significa que leerá tantos caracteres como se tarde en llegar al final de la línea, incluso si sobrescribe el final de la memoria alojada.

Ejemplos BoF

- Conociendo esto, un atacante puede rebasar la memoria "name" dentro de la memoria "dangerous_system_command", y ejecutar el comando que le plazca:

```
gcc overrun.c -o overrun
./overrun
What is your name? 01234567891234567cat /etc/passwd
```

Y muestra el archivo :)

Explotando los comentarios de semántica en C (anotaciones)

Agregando @ luego de la "/*"

- Agregar una @ luego de la /* es reconocida como una entidad sintáctica por la herramienta LCLint.
- En una declaración de parámetro, indica que el valor pasado por este parámetro no será NULL
- Ejemplo /*@ este valor no necesita estar en blanco (null)*/

comentarios de semántica en C

Las anotaciones pueden ser definidas por LCLint utilizando cláusulas

- Describir los supuestos acerca de los buffers que se pasan a las funciones.
- Restringir el estado de buffers cuando se devuelven los supuestos y las restricciones utilizadas en el ejemplo siguiente: minSet, maxSet, minRead, maxRead.

¿Cómo mutar un exploit BoF?

Por la Porción NOT

- Aleatoriamente remplazar los NOPs con segmentos de equivalente funcionalidad (ej: x++, x-; ? NOP NOP)
- Por el evento principal
- Aplicar XOR para combinar código con claves aleatorias ininteligibles al IDS. El código CPU debe decodificar las galimatías con el tiempo con el fin de ordenar la ejecución del decoder. Por si mismo, el decodificador es polimórfico y por ende difícil de detectar.
- - Aleatoriamente ajustar el LSB del puntero para el terreno de la zona NOP.

Detección de BoF

Identificando los BoF

Paso 1. Ejecutar el servidor web en el equipo local.

Paso 2. Emitir solicitudes con etiquetas largas, las tags largas terminan con "$$$$$".

Paso 3. Si el servidor web se bloquea, buscar el volcado de memoria "$$$$$" para encontrar la locación del overflow.

Detección de BoF

Paso 4. Utilizar herramientas automatizadas como codeBloquer, eEye Retina, etc.

Paso 5. Utilizar desensambladores y debuggers.

Paso 6. Utilizar IDS-Pro para construir un exploit.

¿Cómo detectar BoF en un programa?

Variables Locales

- En este caso, el atacante puede buscar por cadenas declaradas como variables locales en funciones o métodos, y verificar la presencia de controles de entrono.

Funciones estándar

- También es necesario revisar el uso inapropiado de funciones estándar, especialmente aquellas relacionadas con cadenas y entrada o salida.

¿Cómo detectar BoF en un programa?

BOU (Utilidad BoF)

- Puede ser utilizada por un atacante para testear aplicaciones Web para condiciones BoF.

Esta herramienta necesita dos entradas:

- El archivo de solicitud, que será probado.
- Cuenta cantidad de código debe ser atacada.

Se necesita un archivo de solicitud que se va a probar y todas las salidas de la actividad en STDOUT basado en el nivel de verbosidad especificado.

Testeando Condificiones Heap Overflow

- Variantes en la vulnerabilidad Hean Overflow (Corrupción Heap) incluyendo:
  1. Permite sobre escribir punteros de funciones.
  2. Explota la estructura de administración de la memoria para la ejecución de código arbitrario.

Testeando Condificiones Heap Overflow

- Probando Heap Overflows abasteciendo cadenas de entradas más largas que las esperadas.
    1. Un puntero de intercambio tomando lugar luego de que la rutina de administración de heap entra en acción.

Testeando Condificiones Heap Overflow

- Dos registros pueden ser llenados con direcciones de usuario suministradas.
  1. Una de las direcciones puede apuntar a un puntero de función que necesita ser sobrescrito, por ejemplo UEF (Unhabdled Exception Filter).
  2. Las otras direcciones pueden ser la dirección del código del usuario suministrador que necesita ser ejecutado.

Testeando Condificiones Heap Overflow

Pasos para testear Stack Overflow en OllyDbg Debugger

- Adjuntar un debugger a la aplicación o proceso objetivo.
- Generar entradas mal formadas para la aplicación.
- Someter a la aplicación a entrada malformada.
- Inspeccionar las respuestas en el debugger.

Testeando Condificiones Heap Overflow

- Testeando las condificiones de la cadena de formato utilizando IDA Pro
  - Las vulnerabilidades de cadenas de formato se manifiestan principalmente en: Servidores Web, Servidores de aplicaciones, aplicaciones Web utilizando código basado en C/C++, scripts CGI escritos en C.

Testeando Condificiones Heap Overflow

- Manipulando parámetros de entrada: El atacante manipula los parámetros de entrada excluyendo los tipos específicos %x o %n.

http://hostname/cgi-bin/query.cgi?name=john&code=45765

a

http://hostname/cgi-bin/query.cgi?name=john%x.%x&code=45765%x.%x

Contramedidas

Defensa contra BoF

- Auditoría manual del código.
- Técnicas del compilador.
- Soporte a la librería del C más segura.

Deshabilitando la ejecución de pilas.

Contramedidas

Prevención de ataques BoF

- Utilizar lenguajes seguros (Java, ML).
- Implementar revisión de tiempo de ejecución.
- Ofuscación de dirección.
- Aleatorizar el lugar de las funciones en libc.
- Análisis de código fuente estático.
- Marcar la pila como no ejecutar, ubicación de pila aleatoria.

Contramedidas

Contramedidas de programación

- Diseñar programas tomando en cuenta la Seguridad.
- Deshabilitar la ejecución de pilas (posible en Solaris).
- Revisar y depurar el código para encontrar errores.
- Prevenir el uso de funciones peligrosas: gets, strcpy, etc.
- Considerar el uso de compiladores "seguros" como StackGuard.
- Prevenir que el retorno de direcciones sea sobrescrito. (cont.)

Contramedidas

- Validar argumentos y reducir la cantidad de código que se ejecuta con privilegio administrativo.
- Prevenir que toda la información sensible sea sobrescrita.
- Realizar cambios en el mismo lenguaje C a nivel de lenguaje para reducir el riesgo de BoF.
- Utilizar analizadores de código fuente estáticos o dinámicos a nivel de código fuente para revisar si hay problemas en el código.
- Cambiar el compilador a nivel de compilador que no limite la revisión o proteja que las direcciones no sean sobrescritas. (cont.)

Contramedidas

- Cambiar las reglas a nivel de S.O. para saber qué páginas de memoria están permitidas de retener datos ejecutables.
- Hacer el uso de librerías seguras.
- Utilizar herramientas de detección de vulnerabilidades BoF.

Contramedidas

Prevención de Ejecución de datos

- DEP (Data execution Prevention) es un conjunto de tecnologías de hardware o software que monitorea programas para verificar si están utilizando la memoria de manera fiable y segura.
- Previene las aplicaciones que puedan acceder a la memoria que no fue asignada por el proceso y se encuentra en otra dirección.
- Cuando la ejecución ocurre el DEP de hardware detecta el código que está ejecutándose desde estas ubicaciones y plantea una excepción.

Contramedidas

- Para prevenir que código malicioso tome una ventaja de los mecanismos de excepción de manejo en Windows es ayudado por un software DEP.
- DEP ayuda a prevenir la ejecución de código desde páginas de datos, como páginas heap por defecto, páginas de memoria, y varias páginas de pilas, donde el código no es ejecutado desde la pila o heap por defecto.

Contramedidas

Enhanced Mitigation Experience Toolkit (EMET)

- Es diseñado para hacer más difícil que un atacante explote vulnerabilidades de un software y obtenga acceso al sistema.
- Soporta técnicas de mitigación que previene técnicas de ataque comunes, principalmente relacionadas con Stack Overflows y técnicas utilizadas por malware para interactuar con el S.O. cuando intenta comprometerlo.
- Mejora la resistencia de Windows a la explotación de BoF.

Desde la ventana de configuración de la aplicación, se pueden agregar aplicaciones para que sean configuradas por EMET.

Herramientas de Seguridad BoF

/GS

- El switch /gs puede ser activado desde la opción de generación de código en el tab C/C++.
- Provee "banda de frenado", o "cookie", entre el buffer y la dirección de retorno que ayuda a prevenir la invasión del buffer.

Herramientas de Seguridad BoF

Herramienta de seguridad BoF: BufferShield

- Permite detectar y prevenir la explotación de BoFs.

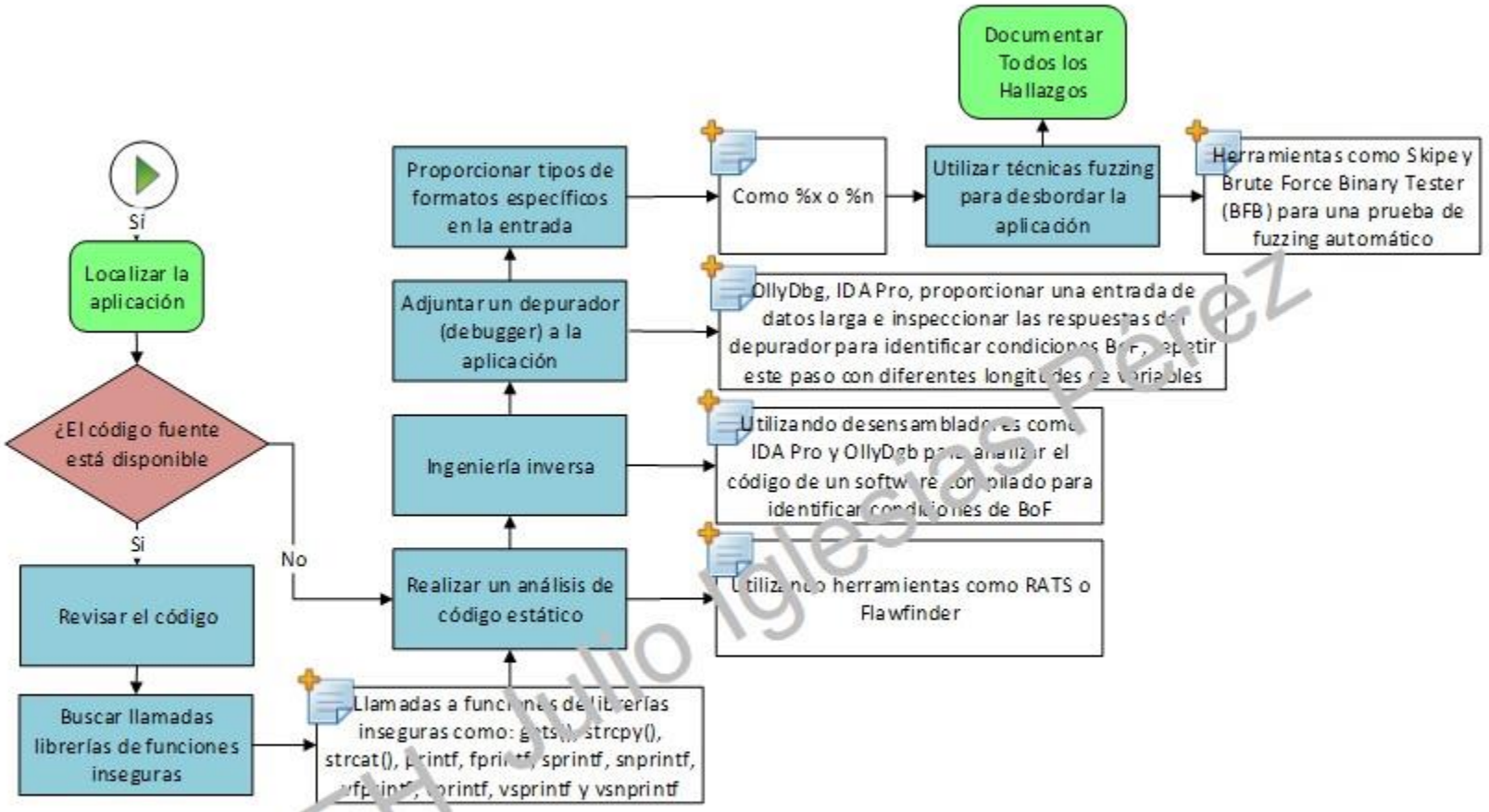
Características:

- Detecta ejecución de código en la pila, heap por defecto, heap dinámico, memoria virtual y segmentos de datos.
- Termina las aplicación si un BoF fue detectado.

Test de Intrusión para BoF

Está diseñado en la presunción de que la aplicación resultara en bloqueo del sistema o comportamiento extraordinario cuando es suministrado con especificadores de tipo de formato y cadenas de entrada que son más largas que las esperadas.

Para esto, el Penetration Tester debe entender como funciona un ataque BoF. Entender lenguajes de programación como C/C++, ensamblador y lenguaje máquina. Competencia en el uso de debuggers, desensambladores y fuzzers.




¡Muchas Gracias!
C|EH Julio Iglesias Perez