Sistemas operativos

Profesor: Gunnar Wolf

Alumno: Diego Armenta

Buffer Overflows

Introducción:

Los Buffer Overflows han sido a lo largo de la historia de la seguridad informática uno de los tipos de vulnerabilidades que han dominado el área.
El principal uso que se le da a un buffer overflow por parte de un atacante es el de ganar control parcial o total de un sistema mediante la inyeccion y ejecución de código por parte del atacante. Esto es posible ya que el ataque se da a partir de las vulnerabilidades que puedan existir en un programa ya existente en el sistema al que se va a lanzar el ataque, debido a que este programa consta de ciertos permisos dentro del sistema, el código inyectado contara con los mismos permisos que el programa desde donde se inyecto el código malicioso.

Detalles de Buffer Overflow:

Usualmente el buffer overflow se puede dividir en dos objetivos que necesitan ser cumplidos para poder realizar un ataque con éxito. Como su nombre lo indica una parte se encarda del buffer, es decir de cargarlo con el código que se necesita ejecutar. Por otra parte esta el

overflow o desborde que es la parte donde el atacante busca hacer que el programa salte a donde está el código que se quiere ejecutar.

Para poder utilizar el código que se quiere ejecutar existen dos posibilidades:

- Inyectarlo:
  - Mediante la utilización de un parámetro dentro del programa vulnerable el atacante puede inyectar una cadena que contenga instrucciones nativas al CPU (ej. shellcode)
- El código ya se encuentra ahí:
  - Una línea de código que ya exista dentro del programa como “exec(arg)” puede ser utilizada para mandar como parametro un parámetro que nos de accceso a una terminal, si en el argumento del ejemplo se introduce la linea “bin/sh”.

Para cumplir el segundo objetivo (es decir, hacer que el programa redireccione hacia la ejecución del código malicioso) existen dos fromas más utlizadas:

- Record de activación:
  - Cuando se llama una función se genera en la pila una dirección de retorno a donde se regresa una vez que la función termina su ejecución. La forma más común de buffer overflow es corromper la dirección de retorno para hacer que esta apunte a el código malicioso. A esta forma específica del buffer overflow se le llama “stack smashing attack” o traducido ataque de golpe a la pila.
- Apuntadores de función:
  - De manera similar al ejemplo anterior se puede corromper el apuntador a una función. Ya que estos se encuentran en cualquier parte de la pila el atacante al encontrar alguno que se pueda desbordar para que cada vez que se llame a la función el puntero vaya a el código malicioso.

Buffer Overflow en C:

El lenguaje C, a pesar de ser considerado de alto nivel, tiene la particularidad de dejar la responsabilidad de la integridad de los datos al programador, en parte a esto se debe su velocidad de ejecución, ya que si esta responsabilidad se dejara al lenguaje tomaría mucho más tiempo procesar los ejecutables ya que se tendría que checar la integridad en cada una de las variables. Esto es una gran ventaja pero también una desventaja, ya que si el programador no es lo suficientemente cuidadoso se pueden dar desbordamientos de memoria, esto significa que si se determina un espacio de memoria de 8 bytes y el programador decide alocar en ese espacio 10 bytes de información el lenguaje lo permitirá y al compilar el programa no se mandará ningun error. Ya que los dos bytes extra de información no caben en el buffer, estos sobreescribiran lo que sea que esté despues del buffer, si se sobreescribe una parte crítica del programa éste dejara de funcionar.

Ejemplo de overflow en C:

overflow_example.c

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

int main(int argc, char *argv[]) {
   int value = 5;
   char buffer_one[8], buffer_two[8];

   strcpy(buffer_one, "one"); /* Put "one" into buffer_one. */
   strcpy(buffer_two, "two"); /* Put "two" into buffer_two. */

   printf("[BEFORE] buffer_two is at %p and contains \'%s\'\n", buffer_two, buffer_two);
   printf("[BEFORE] buffer_one is at %p and contains \'%s\'\n", buffer_one, buffer_one);
   printf("[BEFORE] value is at %p and is %d (0x%08x)\n", &value, value, value);

   printf("\n[STRCPY] copying %d bytes into buffer_two\n\n",  strlen(argv[1]));
   strcpy(buffer_two, argv[1]); /* Copy first argument into buffer_two. */

   printf("[AFTER] buffer_two is at %p and contains \'%s\'\n", buffer_two, buffer_two);
   printf("[AFTER] buffer_one is at %p and contains \'%s\'\n", buffer_one, buffer_one);
   printf("[AFTER] value is at %p and is %d (0x%08x)\n", &value, value, value);
}
```

Erickson, J. (2008). overflow_example.c [Pdf]. En *Hacking the art of exploitation* (2.ª ed., p. 120).

Ejecución:

```
reader@hacking:~/booksrc $ gcc -o overflow_example overflow_example.c
reader@hacking:~/booksrc $ ./overflow_example 1234567890
[BEFORE] buffer_two is at 0xbffff7f0 and contains 'two'
[BEFORE] buffer_one is at 0xbffff7f8 and contains 'one'
[BEFORE] value is at 0xbffff804 and is 5 (0x00000005)

[STRCPY] copying 10 bytes into buffer_two

[AFTER] buffer_two is at 0xbffff7f0 and contains '1234567890'
[AFTER] buffer_one is at 0xbffff7f8 and contains '90'
[AFTER] value is at 0xbffff804 and is 5 (0x00000005)
reader@hacking:~/booksrc $
```

En el ejemplo anterior se demuestra el funcionamiento basico de un desbordamiento de memoria en C. Al mirar la ejecución del código se puede notar como los buffers uno y dos estan juntos y al sobrecargar el buffer dos la información que se desborda se sobreescribe en el buffer uno.

Conclusiones:

El buffer overflow es una vulnerabilidad con muchas variantes que se puede explotar en diversos sistemas y lenguajes de programación. Conocer su funcionamiento permite crear contramedidas y evitar malas practicas en el diseño de programas para evitar vulnerabilidades de este tipo.

Dentro de las concideraciones que se pueden tener para evitar que los sistemas sean vulnerables al desbordamiento se puede poner como primer lugar el escribir codigo que no incurra en malas practicas y verificar que al utilizar funciones como “strcpy”, “sprintf” se revise antes la longitud de los argumentos.

En algunos sistemas se comenzo a eliminar la posibilidad de crear buffers que sean ejecutables, sin embargo para diversas optimizaciones algunos sistemas UNIX y Windows

utlizan segmentos dinamicos de codigo dentro de los datos de algunos programas. La forma más segura es evitar el desbordamiento sin preocuparnos tanto por la inyeccion, verificando los arreglos de un programa para que a estos no se puedan ingresar datos que sobrepasen el tamaño del arreglo es de las mejores contramedidas para evitar que se sobreescriban pedazos de memoria de manera ilegal.

Referencias:

- C. Cowan, F. Wagle, Calton Pu, S. Beattie and J. Walpole, "Buffer overflows: attacks and defenses for the vulnerability of the decade," *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, Hilton Head, SC, USA, 2000, pp. 119-129 vol.2, doi: 10.1109/DISCEX.2000.821514.
- Erickson, J. (2019). *Hacking: The art of exploitation*. Brantford, Ont.: W. Ross MacDonald School Resource Services Library.
- Fernandez, D., 2002. *Creacion De Shellcodes Para Linux*. [ebook] Madrid. Available at: <https://www.isecauditors.com/sites/default/files/files/creacion-shellcodes-para-exploits-madhack.pdf> [Accessed 18 January 2021].