Práctica 4: Vulnerabilidades de desbordamiento

Seguridad Informática

Pedro Allué Tamargo (758267) Juan José Tambo Tambo (755742)

26 de noviembre de 2020

Índice

1.	Identificación de vulnerabilidades	1
	1.1. Hay una vulnerabilidad asociada a una variable que puede ser indexada fuera de su límite	1
	1.2. Hay vulnerabilidades de desbordamiento de búfer en el programa	1
	1.3. ¿Hay otros tipos de vulnerabilidades en el código? ¿Cuáles?	
2.	Redirección de la ejecución	2
	2.1. ¿Cuál es la dirección de las variables func y funcsec? ¿En qué parte de la memoria se encuentran?	2
	2.2. ¿Cuál es la dirección del método showSecret1?	2
	2.3. ¿Qué datos de entrada proporcionas al programa para que func[s] lea el puntero a la función	
	guardado en funcsec, en lugar de un puntero a una función guardado en func?	2
3.	Ejecución del método mostrarSecreto2	3
	3.1. ¿Cuál es la dirección del búfer asociado a la variable resp?	3
	3.2. ¿Qué datos de entrada proporcionas al programa para que func[s] lea a partir del 126º byte en	
	resp, es decir, a partir de resp[125]?	3
		4
4.	Bola extra ????	4

1. Identificación de vulnerabilidades

1.1. Hay una vulnerabilidad asociada a una variable que puede ser indexada fuera de su límite

- ¿Cuál es la variable?
 - La variable es func. Esta variable almacena un *array* de punteros a funciones que devuelven void y no aceptan parámetros.
 - Ejecutando el programa sin las contramedidas, cuando pide la introducción de una opción del menú, se introduce la opción 6 y se indexa el array funcsec, declarado en direcciones contiguas.
- Indicar la línea de código que puede indexar la variable fuera de su límite.
 - La variable se puede indexar fuera de su límite en la línea 131.

1.2. Hay vulnerabilidades de desbordamiento de búfer en el programa

- ¿Cuáles son las variables?
 - La variable comida en la función llenarCarrito. Acepta 512 bytes de longitud pero la función scanf no establece un límite para controlar la longitud de la cadena a copiar.
 - La variable malo en la función mostrarCalorias utiliza una versión no segura de la función strlen que devuelve el número de bytes (caracteres) entre una dirección de inicio y el carácter terminador
 O. Si esta longitud es mayor que 512 (MAX_SIZE) se copiarán tantos caracteres como diga len o hasta llegar al carácter terminador.
- ¿Qué parte de la memoria asociada al proceso se puede desbordar?
 - Se podría desbordar la pila. Al ser variables que se declaran en funciones y no son globales se almacenan en la pila.
- Indicar las líneas de código que pueden desbordar los búferes.
 - comida: la función scanf (línea 58)
 - malo: la función strlen (línea 86) junto con la función strncat (línea 87).

1.3. ¿Hay otros tipos de vulnerabilidades en el código? ¿Cuáles?

- Hay una vulnerabilidad de desbordamiento de enteros en la función mostrarCalorías, en la línea 90 (variable total). Dada una lista de comidas lo suficientemente grande, y debido al bucle for de la línea 81 se podría desbordar el valor de esta variable.
- Existe otra vulnerabilidad de desbordamiento de enteros relacionada con la elección de la opción del menú del usuario (línea 127) puede desbordar el valor del entero s (función atoi, línea 130) si resp no se puede representar en el rango de los enteros (int). Una solución sería utilizar la función strtol que convierte una cadena de texto a un entero tipo long y ante desbordamientos, devuelve los límites máximos o mínimos del tipo long, dependiendo de por donde ha desbordado.

2. Redirección de la ejecución

2.1. ¿Cuál es la dirección de las variables func y funcsec? ¿En qué parte de la memoria se encuentran?

Para obtener la localización de las variables en memoria mediante gdb se utilizará la orden: print &variable. Por lo tanto, las direcciones de las variables serán las siguientes:

- La variable func se encuentra en la dirección 0x804b064
- La variable funcsec se encuentra en la dirección 0x804b078

Las variables se encuentran en la zona de datos inicializados (*Initialized Data Segment*) ya que son variables globales cuyo valor ha sido otorgado por el programador.

2.2. ¿Cuál es la dirección del método showSecret1?

Para obtener la dirección de la función showSecret1 mediante gdb se ha utilizado la siguiente orden: print &Carrito::mostrarSecreto1. Ya que mostrarSecreto1 es un método estático de la clase Carrito. Su dirección de memoria es: 0x8048bce.

2.3. ¿Qué datos de entrada proporcionas al programa para que func[s] lea el puntero a la función guardado en funcsec, en lugar de un puntero a una función guardado en func?

La entrada proporcionada al programa para leer un puntero guardado en funcsec sería de al menos 5. Esto es así ya que la dirección inicial de func es 0x804b064 y almacena punteros, cuyo tamaño son 4 bytes. Para leer un puntero de funcsec habría que indexar la quinta posición (empezando por 0) de func $(0x804b064 + (5*size_puntero)) = 0x804b078$).

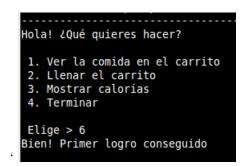


Figura 1: Captura de pantalla del resultado de la ejecución de mostraSecreto1

3. Ejecución del método mostrarSecreto2

3.1. ¿Cuál es la dirección del búfer asociado a la variable resp?

Para conseguir la dirección de memoria de la variable resp mediante gdb se utilizarán los siguientes comandos. El comando backtrace (bt) servirá para obtener la pila de ejecución (backtrace) del programa (Figura numFigura). Tras esto se ejecutará el comando frame 9 (f 9). Este comando establecerá el contexto al de la función main. Una vez situados en este contexto se ejecutará print &main::resp. La dirección de memoria asociada a la variable resp será: 0xbffff34f

```
(gdb) bt
    0xb7fe1430 in
                      _kernel_vsyscall ()
    0xb7e1cf23 in read () from /lib/tls/i686/cmov/libc.so.6
0xb7db404e in _IO_file_underflow () from /lib/tls/i686/cmov/libc.so.6
#2 0xb7db404e in
#3 0xb7db5dab in IO default uflow () from /lib/tls/i686/cmov/libc.so.6
                      uflow () from /lib/tls/i686/cmov/libc.so.6
#4 0xb7db7362 in
   0xb7db08c7 in getc () from /lib/tls/i686/cmov/libc.so.6
    0xb7f6a3c5 in
                      _gnu_cxx::stdio_sync_filebuf<char, std::char_traits<char> >::
#6 0xD/T6a3c5 in __gnu_cxx::std10_sync_t
underflow () from /usr/lib/libstdc++.so.6
    0xb7f4ccc4 in std::istream::sentry::sentry () from /usr/lib/libstdc++.so.6
    0xb7f522a5 in std::operator>><char, std::char_traits<char> > ()
   from /usr/lib/libstdc++.so.6
    0x08048b43 in main () at carrito.cpp:127
```

Figura 2: Captura de pantalla de la salida de la orden backtrace (bt)

3.2. ¿Qué datos de entrada proporcionas al programa para que func[s] lea a partir del 126º byte en resp, es decir, a partir de resp[125]?

Teniendo en cuenta los siguientes datos:

- Dirección de mostrarSecreto2 = 0x08048a54
- Dirección de resp = 0xbffff34f
- Dirección de func = 0x0804b064

Por lo tanto se busca que:

- resp + 125 * 4 = @mostrarSecreto2

Resolviendo con los datos anteriores se obtiene:

- $Y = \frac{@resp + 125*4 @func}{4} = 771674330$
- $\bullet \ xEE... = @mostrarSecreto2 = 0x08048a54$
- El número de A debe por lo tanto abarcar 125 len(771674330) 1 = 115

Por lo tanto para explotar la vulnerabilidad se utilizará la siguiente entrada:

Figura 3: Captura de pantalla de la entrada necesaria para obtener el segundo logro

3.3. ¿Hay otra forma de conseguir la escritura del segundo mensaje secreto por pantalla?

Dirección de mostrar Secreto
2 = 0x08048a54 Dirección saved eip = 0xb7d5f775

Dirección de eip = 0xbffff558

Entre resp y saved eip hay 525 bytes

Hay que hacer que resp desborde la pila para que el saved eip (dirección de retorno) apunte a la dirección de la función mostrarSecreto2.

https://stackoverflow.com/questions/5144727/how-to-interpret-gdb-info-frame-output

4. Bola extra ????

Esto no lo hago ni aunque me paguen.