

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?	1
2. Redirección de la ejecución	2
2.1. ¿Cuál es la dirección de las variables <code>func</code> y <code>funcsec</code> ? ¿En qué parte de la memoria se encuentran?	2
2.2. ¿Cuál es la dirección del método <code>showSecret1</code> ?	2
2.3. ¿Qué datos de entrada proporcionas al programa para que <code>func[s]</code> lea el puntero a la función guardado en <code>funcsec</code> , en lugar de un puntero a una función guardado en <code>func</code> ?	2
3. Ejecución del método <code>mostrarSecreto2</code>	3
3.1. ¿Cuál es la dirección del búfer asociado a la variable <code>resp</code> ?	3
3.2. ¿Qué datos de entrada proporcionas al programa para que <code>func[s]</code> lea a partir del 126º byte en <code>resp</code> , es decir, a partir de <code>resp[125]</code> ?	3
3.3. ¿Hay otra forma de conseguir la escritura del segundo mensaje secreto por pantalla?	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 0. 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 `mostrarCalorias`, 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`

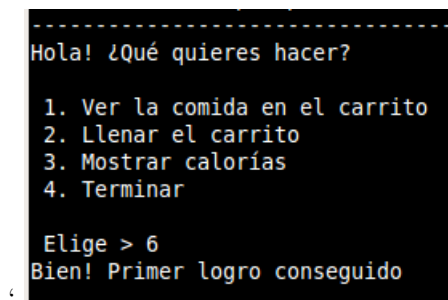
Las variables se encuentran en la zona de datos inicializados (*Initialized Data Segment*) ya que son variables globales cuyo valor ha sido otorgado por el programador.

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` ($0x804b064 + (5 * \text{size_puntero}) = 0x804b078$).



```
-----
Hola! ¿Qué quieres hacer?

1. Ver la comida en el carrito
2. Llenar el carrito
3. Mostrar calorías
4. Terminar

Elige > 6
Bien! Primer logro conseguido
```

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

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

Dirección de `mostrarSecreto2` = `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.