Prácticas de Seguridad Informática

—

Práctica 4

—

Grado Ingeniería Informática

Curso 2014-2015

Universidad de Zaragoza
Escuela de Ingeniería y Arquitectura
Departamento de Informática e
Ingeniería de Sistemas
Area de Lenguajes y Sistemas Informáticos

12 de noviembre de 2014

Introducción

En esta práctica vamos a trabajar con pequeños programas que tienen vulnerabilidades de desbordamiento de memoria (*buffer overlow*). El objetivo es ver los problemas en un entorno controlado y manejable que nos permita conocer bien los conceptos y experimentar lo que sucede en estos casos. También veremos cómo el problema puede afectarnos de diferente manera según el computador en el que estemos trabajando y los compiladores que utilicemos.

La práctica se hará en 'hendrix'[1] que tiene CPU UltraSPARC-T2 1165 MHz (interesante porque no es Intel, que seguramente es lo que tenemos más accesible en casa) con sistema operativo Solaris y su propio compilador de c (accesible mediante `cc`) además del de GNU (accesible mendiante `gcc`).

Descripción

Se os han facilitado a través de Moodle (y eventualmente aparecerán en la página web del profesor) las transparencias que se utilizarán para introducir los conceptos generales y el código fuente de los programas que se muestran (puede haber pequeñas variaciones).

El primer objetivo es seguir la presentación haciendo las pruebas con los programas que se os entregan, experimentando las diversas situaciones de desbordamiento de memoria que se producen con el compilador `gcc` (por ejemplo: `gcc -o overflow overflow.c`).

Algunas sugerencias sobre posibles pruebas serían:

- Probar con entradas sucesivamente más largas.[2]
- Cambiar el orden de las variables.
- Añadir y eliminar variables.
- Añadir instrucciones para imprimir el valor de las variables.

El segundo objetivo es reproducir lo que se ha hecho siguiendo los ejemplos, pero con un compilador diferente: el que proporciona el propio sistema de hendrix, que se invoca mediante `cc` (por ejemplo, `cc -o overflow overflow.c`) y responder a las siguientes preguntas:

1. ¿Se comportan todos los ejemplos de la misma manera? Si hay diferencias [3] indicarlas.
2. ¿Cuáles pueden ser los motivos?

[1] `https://diis.unizar.es/WebEstudiantes/hendrix/`

[2] Puede ser de ayuda recordar que `perl -le 'print .a"x10'` genera una tira de diez aes `aaaaaaaaaa`. Sustituyendo diez por otro número y la a por otro carácter podemos generar tiras del carácter que indiquemos y de la longitud deseada. Por ejemplo, `./overflow 'perl -le 'print "b"x100' '` haría una invocación al programa overflow seguido de una tira de cien bes.

[3] Las diferencias, si las hay, pueden ser: falla/no falla; el tamaño de la entrada necesario para fallar es diferente, hay que ordenar las variables de otra forma, otras...

¿Qué hay que hacer?

Además de las **pruebas** que se han señalado anteriormente y la **respuesta razonada (pero breve)** a las preguntas señaladas anteriormente, se propone añadir una variación a las pruebas: podemos probar en casa con nuestro computador (con GNU Linux o Mac OS, basados en x86) y con el compilador gcc para ver qué sucede.

Haremos las mismas pruebas y responderemos a las mismas preguntas.

Y, si queremos aprender más cosas y experimentar más sobre este tema, podemos pasar a las siguientes secciones, cuyo **entrega no es obligatoria**.

Bola extra

Añadir un nuevo compilador (o más de uno) al experimento. Sugerencias: LLVM [4] o el compilador de Intel [5].

Otra bola extra

En lugar de probar nuevos compiladores (aunque eso siempre es una buena idea), podemos trabajar ahora con el programa `overflowSecret.c`.

El trabajo consiste en encontrar una entrada de datos para que el programa modificado muestre el carácter secreto.

Notas:

1. El objetivo en este caso es sobre-escribir la dirección de retorno de la función `claveCorrecta` para conseguir que se ejecute `secreto` con una entrada adecuada.
2. Está permitido que el programa ejecute una llamada a la función y se interrumpa con algún error.
3. Puede ser de utilidad utilizar el depurador `gdb`

   `gcc -ggdb overflowSecret overflowSecret.c`

   para conocer la dirección de las funciones (en realidad nos interesa la diferencia entre ambas).
4. Recordar que no todos los procesadores almacenan los números (y por tanto las direcciones de memoria) de la misma manera (x86 lo hace en ‘little-endian‘).
5. En este caso, puede ser de utilidad la redirección de datos de entrada:

   `perl -le 'print .a"x10' | ./overflowSecret`

[4] The LLVM Compiler Infrastructure, `http://llvm.org/`

[5] Intel® C++ Studio XE 2013 for Linux, descarga gratuita para uso no comercial. `http://software.intel.com/en-us/non-commercial-software-development`