Emilio Chica Jiménez 16/12/2014

Diario de un Cracker

Introducción

La práctica que se nos ha propuesto tiene como objetivo aprender a desemsamblar, utilizar la ingeniería inversa en código ajeno y poder entender dicho código y las posibles vulnerabilidades ante un ataque.

Para observar dichas vulnerabilidades hemos usado herramientas tanto de depuración, como de desemsamblado para tener una guía de las instrucciones del programa y así poder ir comparando con los resultados que nos muestra.

Herramientas Utilizadas

- Desemsamblado:
  - **objdump -d** : Con esto obtendremos el código de depuración que incluye el archivo ejecutable que no ha sido strippeado o compilado sin depuración. Podemos usar objdump -d {archivo} > resultado.txt y tener guardado dicho código.
- Edición del ejecutable:
  - **ghex :** Este programa nos permite editar el código que contiene nuestro programa en formato hexadecimal. Para poder entender el formato hexadecimal y que representa cada línea que estamos viendo usamos el resultado de objdump que nos traduce las lineas de ensamblador a hexadecimal y ya sólo tendríamos que buscar en el programa dicha línea concreta y editarla.
- Seguimiento del programa:
  - **ltrace y strace**: Dos herramientas muy útiles a la hora de ver las llamadas que está realizando el programa investigado. **ltrace** nos permite ver las llamadas internas que hace el usuario que ha programado el programa. **strace** nos permite ver las llamadas al sistema que ha realizado el usuario que ha programado el programa.
  - **ddd**: Herramienta de depuración que usamos para seguir la traza del programa que estamos investigando. Para ello este programa se sirve de código ensamblador para mostrarnos la línea por la que empieza a ejecutarse el programa. Este programa hace uso de gdb como depurador y nos presenta una interfaz gráfica donde podemos poner puntos de interrupción al programa y partir de estos ver el contenido de los registros y pila por la línea en la que estamos.

Emilio Chica Jiménez 16/12/2014

Procedimiento de búsqueda de vulnerabilidades y uso de herramientas

Búsqueda de contraseñas:

Los programas que nos proporcionan los compañeros son programas los cuales no se les ha quitado el código de depuración por facilitar la práctica. Las opciones de compilación han sido de 32 bits para que podamos entender el código ensamblador de manera sencilla. Por lo tanto podemos usar las herramientas que muestro a continuación y de la forma que muestro para encontrar las claves del programa sin muchas variaciones de un programa a otro.

Primer paso que doy para situarme y ver de que va el programa es el desemsamblado del mismo:

objdump -d {programa} > resultado.txt

Ahora que tengo el código de depuración en un archivo de texto y en un formato como este:

Desensamblado de la sección .text:

```
080484f0 <main>:
 80484f0:       55                      push   %ebp
 80484f1:       89 e5                   mov    %esp,%ebp
 80484f3:       56                      push   %esi
 80484f4:       53                      push   %ebx
 80484f5:       83 e4 f0                and    $0xfffffff0,%esp
 80484f8:       81 ec 90 00 00 00       sub    $0x90,%esp
 80484fe:       8d 74 24 1c             lea    0x1c(%esp),%esi
 8048502:       c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
```

Ahora puedo proceder a usar otras herramientas, el orden es el que se indica a continuación a menos que hayan cambiado el orden de las claves en cuyo caso habría que testearlo con ltrace.

Por ejemplo para sacar la clave de texto que usan mis compañeros, al usar llamadas a funciones ya definidas como son strlen o strncmp para comparar las cadenas y ver si la cadena introducida por teclado como clave es la correcta, es fácil ver con ltrace dicha clave. Pues ltrace se encargará de mostrarnos los argumentos que se le pasan a dichas funciones usandolo de forma:

ltrace ./bombapaco

```
__libc_start_main(0x80484f0, 1, 0xffffbf344, 0x8048790 <unfinished ...>
gettimeofday(0xffffbf22c, 0)                    = 0
printf("Introduce la contrase\303\261a: ")      = 26
fgets(Introduce la contraseña: prueba
"prueba\n", 100, 0xf76d2c20)                    = 0xffffbf23c
strlen("f0b1n1ccI\n")                           = 10
strncmp("prueba\n", "f0b1n1ccI\n", 10)          = 1
puts("***************"***************
```

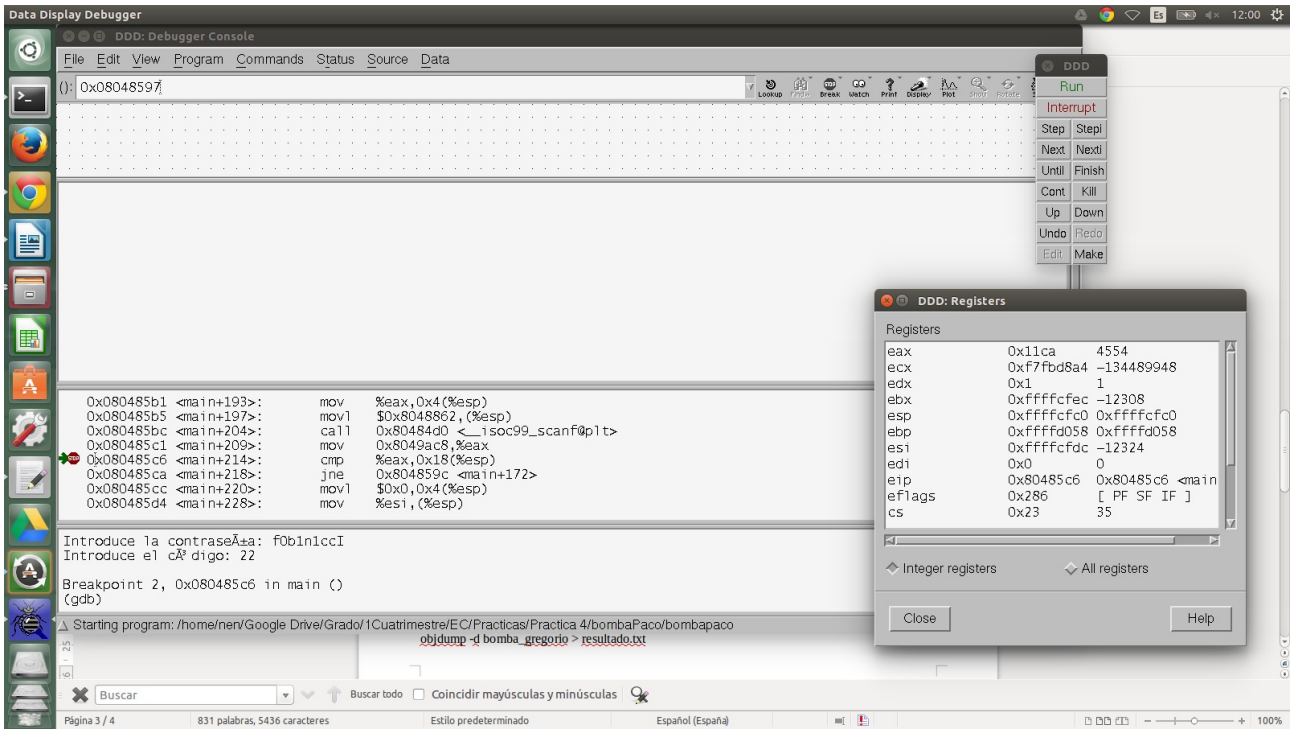
) = 16
puts("* BOOM!!! ***"*** BOOM!!! ***
) = 16**
puts("********"**********
) = 16**
exit(-1 <no return ...>
+++ exited (status 255) +++

Como vemos resaltado en morado las funciones usadas en el programa son las indicadas anteriormente strlen y strncmp y con ltrace podemos ver los argumentos de dichas funciones por lo tanto ya tenemos la primera contraseña.

Ahora para encontrar el segundo código numérico lo que me he encontrado en práticamente todas las bombas desactivadas es una comparación numérica sin llamar a ninguna función por lo que ltrace no nos puede ayudar para encontrar esta contraseña.

Sin embargo ddd nos va a servir para esta tarea, ya que veremos el valor de los registros en el momento que queramos que normalmente será en un codigo condición del tipo text %eax, %eax o cmp %eax, 12(%ebp), donde comparamos el valor que ha introducido el usuario con el numero que tenemos guardado como contraseña. Para usar el ddd hacemos algo así:

ddd bombapaco



En este caso el código es: **4554**

Y ya tenemos los códigos del programa.

Emilio Chica Jiménez 16/12/2014

Modificación del programa y contraseñas

Para poder modificar el programa ddd no nos sirve y ltrace tampoco para ello debemos usar la herrramienta que nos queda por usar **ghex,** que combinada con el código proporcionado por objdump nos dará como resultado la modificación de cada línea de código que queramos que o deje de funcionar o funcione como nosotros queramos.

Bien para usarla es muy sencillo escribimos lo siguiente en la linea de comandos:

ghex bombapaco

Ahora veremos una serie de código escrito en hexadecimal la izquierda de la pantalla y a la derecha los caracteres imprimibles de dicho código. Pues bien ahora para proceder a eliminar o modificar lineas de código tenemos que tener en cuenta dos cosas:

Cuáles son los códigos de operación en hexadecimal de líneas que nos ayudarán a dejar el programa como si no tuviese contraseñas. Para ello nos servimos de un pequeño programa escrito en ensamblador en el que tenemos las líneas justas para saber su código de operación correspondiente. Primero lo compilamos y luego lo desemsamblamos con objdump -d. Un ejemplo sería el siguiente:

Desensamblado de la sección .text:

```
0804816f <_start>:
 804816f:       31 c0                   xor    %eax,%eax
 8048171:       90              nop
 8048172:       83 f0 01                xor    $0x1,%eax
 8048175:       83 f3 00                xor    $0x0,%ebx
 8048178:       cd 80                   int    $0x80
 804817a:       90              nop
 804817b:       eb fe                   jmp    804817b <_start+0xc>
 804817d:       90              nop
 804817e:       31 c0                   xor    %eax,%eax
 8048180:       90              nop
 8048181:       e8 00 00 00 00          call   8048186 <empezar>

08048186 <empezar>:
 8048186:       90              nop
```

Hay tener en cuenta a la hora de compilar que es un programa escrito en ensamblador por lo tanto tendremos que usar algunos flags a la hora de compilar con gcc:

gcc -fno-omit-frame-pointer -g0 -m32 -nostartfiles {archivocodop.s} -o resultado

Y luego usar objdump sobre resultado.

Bien ahora ya sabemos las operaciones que podemos usar y cambiar en nuestro código. Pero cómo sabemos donde estan las operaciones que queremos cambiar por ejemplo una del tipo:

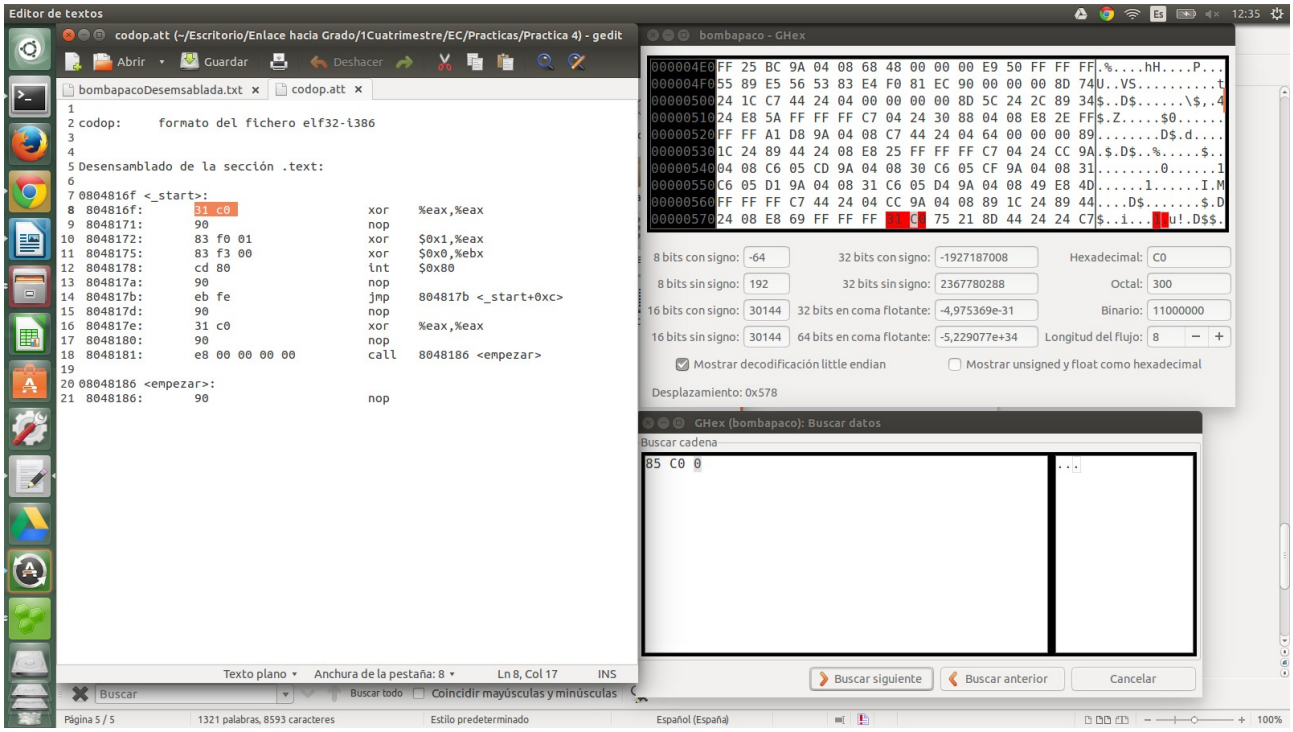
8048577: 85 c0 test %eax,%eax

Pues si nos fijamos en el código resaltado en rosa, el cual encontramos en resultado.txt sabremos que tenemos que buscar en ghex como código de operación. Pero puede que haya muchos iguales, asi que para averiguar exactamente cual es tendremos que mirar al menos de que líneas está seguido:

```
8048577:    85 c0          test   %eax,%eax
8048579:    75 21          jne    804859c <main+0xac>
804857b:    8d 44 24 24    lea    0x24(%esp),%eax
```

Bien ahora sólo queda modificar el código de operación 85 c0 por ejemplo en y este caso en concreto que tenemos un jne es decir salta si no es igual tendremos que asegurarnos que el flag ZF que es el que comprueba esta instrucción esté a cero. Y que mejor que usar xor %eax, %eax que nos modifica dicho flag y como resultado en este caso nos meterá un 0.

Por lo tanto tendríamos que poner el código de operación de xor %eax, %eax en esa linea del siguiente modo:



Vale ya tenemos cambiada la linea de salto que nos hacia saltar hacia el boom de la bomba y este

proceso será el que debemos seguir con cada una de las condiciones. En el caso de que quisieramos cambiar la clave al programa del mismo modo buscaríamos la línea en resultado.txt que contiene la variable y le cambiaríamos el valor con ghex como hemos echo anteriormente.

Ya solo queda guardar el archivo de ghex como resultado de nuestra operación de cambio del programa y vualá ya tenemos el código sin contraseñas molestas.

Bombas desactivadas

Gregorio Carvajal Expósito:

bomba_gregorio
Pass: yosoytupadre
codigo: 405

Julián Torices Hernández

Pass: gusanico
Problema en ddd que no mostraba todo el código al completo del desemsamblado para ello tuve que poner el breakpoint a mano con:
break 0x8048b82
codigo: 15

Francisco Carrillo Pérez

bombapaco
Pass: f0b1n1ccI
código: 4554