Adrián de la Torre Rodríguez 2ºA

DESACTIVACIÓN DE LA BOMBA

CONTRASEÑA: `explosion`

CÓDIGO: `1234`

Mi bomba cuenta con un cifrado de la contraseña (explosion). Hay un array de caracteres aparentemente sin sentido, pero en los índices primos(2, 3, 5, 7...) se encuentran los caracteres de la contraseña, este es dicho array:

```
char codigo[]="atexupslsowoysghziqolkknqwfrh\n";
```

Para decodificar este array tengo dos funciones, una que recorre este array(decodifica ()) y si llega a un índice primo, almacena ese carácter en otro array que está en memoria llamado password. La otra función comprueba si un número es primo(es_primo()).

Voy a explicar como se quedan estos arrays antes y después de decodificar:

1.- Antes de llamar a la función decodifica() los arrays están así:

```
char codigo[]="atexupslsowoysghziqolkknqwfrh\n";
char password[10]= //Basura//
```

2.- Después de llamar a la función decodifica():

```
char codigo[]="atexupslsowoysghziqolkknqwfrh\n";
char password[10]= "explosion\n"
```

Una vez explicado el funcionamiento de estas funciones voy a explicar el "truco" o atajo para descubrir mi bomba. Si desamblamos el ejecutable, podemos ver que hay dos llamadas a la función decodifica() justo después de escribir cada contraseña. Esto da a pensar que lo que escribimos está siendo decodificado o haciendo una operación desconocida con la contraseña cuando en realidad la llamada a la función codifica es necesaria solo una vez para que almacene la contraseña correcta en el array password.

El alumno que se meta a desamblar la función decodifica() se va a encontrar con un código complejo(ya que trabaja con la función es_primo() , con índices de arrays y saltos condicionales). Este es el código maquina de la función decodifica():

```
0x080486f7 <+0>:     push   %ebp
0x080486f8 <+1>:     mov    %esp,%ebp
0x080486fa <+3>:     sub    $0x14,%esp
0x080486fd <+6>:     movl   $0x0,-0x4(%ebp)
0x08048704 <+13>:    movl   $0x0,-0x8(%ebp)
0x0804870b <+20>:    jmp    0x804873a <decodifica+67>
0x0804870d <+22>:    mov    -0x8(%ebp),%eax
0x08048710 <+25>:    mov    %eax,(%esp)
0x08048713 <+28>:    call   0x8048679 <es_primo>
0x08048718 <+33>:    test   %eax,%eax
0x0804871a <+35>:    je     0x8048736 <decodifica+63>
0x0804871c <+37>:    mov    -0x8(%ebp),%eax
0x0804871f <+40>:    add    $0x804a040,%eax
0x08048724 <+45>:    movzbl (%eax),%eax
0x08048727 <+48>:    mov    -0x4(%ebp),%edx
0x0804872a <+51>:    add    $0x804a069,%edx
0x08048730 <+57>:    mov    %al,(%edx)
0x08048732 <+59>:    addl   $0x1,-0x4(%ebp)
0x08048736 <+63>:    addl   $0x1,-0x8(%ebp)
0x0804873a <+67>:    cmpl   $0x1d,-0x8(%ebp)
0x0804873e <+71>:    jle    0x804870d <decodifica+22>
0x08048740 <+73>:    leave
0x08048741 <+74>:    ret
```

Se puede dar uno cuenta con este código que se accede a dos direcciones de memoria con las funciones add(una es password donde no hay nada y otra es codigo donde están todos los caracteres, pero esto el alumno no lo sabe). Si se examinan las direcciones de memoria con el ddd haciendo Data>Memory:

Examine 30, char, bytes, $0x0804a069

Esta corresponde con el array password que este momento esta vacío y obtenemos basura.

Lo volvemos a hacer con la otra dirección:

Examine 30, char, bytes, $0x0804a040

En este caso obtenemos los 30 caracteres del array codigo pero no tienen sentido y no se sabe para que sirven.

El alumno puede estar mucho rato intentando comprender esta función, pero también se puede dar cuenta de que el main es exactamente igual al ejemplo visto en clase solo que cambia que se llama a la función decodifica( ) . En este caso basta con seguir lo que se vio en clase para averguar las constraseñas con el ddd.

PASOS PARA AVERGUAR LAS CONTRASEÑAS

1.- Ejecutar el ddd.

2.- Escribir info line main para acceder al código máquina del main.

3.- Obviar las llamadas a decodifica( ) y centrarse en la llamada a strncmp.

4.- A la función strncmp se le pasan tres argumentos, la contraseña introducida, el array password y la longitud de el array password. El argumento que nos interesa es password, que corresponde a la instrucción máquina “mov $0x0804a069,0x4(%esp)” (Mover esa dirección de memoria a cuatro por encima de esp, es decir, que corresponde al segundo argumento de la función strncmp).

5.- Acceder a esa dirección de memoria. Para ello ponemos un breakpoint después de la función decodifica(), yo por ejemplo lo he puesto en <main+93>. Arrancamos el programa y cuando estemos parados en el breakpoint hacemos un volcado de memoria de la dirección de password:

Data > Memory

Examine 10, char, bytes, from 0x0804a069

(Pulsamos print)

Por pantalla tenemos lo siguiente:

```
0x804a069 <password>:     101 'e' 120 'x' 112 'p' 108 'l' 111 'o' 105 'i' 111 'o'
0x804a071 <password+8>:   110 'n' 10 '\n'
```

Con esto sabemos que la contraseña con la que estamos comparando es “explosion\n”

6.- Una vez sabemos la contraseña, vamos a averiguar el código. Para ello paramos la ejecución del programa, quitamos el breakpoint que tenemos y lo ponemos en <main+226> en esta línea se compara el código que hemos introducido con otro (cmp %eax,%edx), ambos códigos están en los registros edx y eax.

7.- Arrancamos el programa, introducimos la contraseña que hemos averiguado y metemos un código cualquiera.

8.- Una vez estamos en el breakpoint, inspeccionamos los registros:

Status > Registers

Ahí vemos que eax vale 1234 y edx vale otro valor. El valor correcto del código es 1234.

9.- Ya sabemos la contraseña y el código, ahora solo queda volver a ejecutar el programa, introducirlos y desactivar la bomba.