Averiguando contraseñas

Julio A. Fresneda García – 49215154F

Bombas usadas: de Adrián Peláez y Jose Antonio Ruíz

Bomba de Adrián:

Desensablamos la bomba con objdump -d:

```
0804860d <encriptar_password>:
 804860d:       55                      push   %ebp
 804860e:       89 e5                   mov    %esp,%ebp
 8048610:       83 ec 28                sub    $0x28,%esp
 8048613:       c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%ebp)
 804861a:       8b 45 08                mov    0x8(%ebp),%eax
 804861d:       89 04 24                mov    %eax,(%esp)
 8048620:       e8 ab fe ff ff          call   80484d0 <strlen@plt>
 8048625:       83 e8 01                sub    $0x1,%eax
 8048628:       89 45 f4                mov    %eax,-0xc(%ebp)
 804862b:       c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%ebp)
 8048632:       eb 40                   jmp    8048674 <encriptar_password+0x67>
 8048634:       8b 45 f0                mov    -0x10(%ebp),%eax
 8048637:       83 e0 01                and    $0x1,%eax
 804863a:       85 c0                   test   %eax,%eax
 804863c:       75 1a                   jne    8048658 <encriptar_password+0x4b>
 804863e:       8b 55 f0                mov    -0x10(%ebp),%edx
 8048641:       8b 45 08                mov    0x8(%ebp),%eax
 8048644:       01 c2                   add    %eax,%edx
 8048646:       8b 4d f0                mov    -0x10(%ebp),%ecx
 8048649:       8b 45 08                mov    0x8(%ebp),%eax
 804864c:       01 c8                   add    %ecx,%eax
 804864e:       0f b6 00                movzbl (%eax),%eax
 8048651:       83 c0 02                add    $0x2,%eax
 8048654:       88 02                   mov    %al,(%edx)
 8048656:       eb 18                   jmp    8048670 <encriptar_password+0x63>
 8048658:       8b 55 f0                mov    -0x10(%ebp),%edx
 804865b:       8b 45 08                mov    0x8(%ebp),%eax
 804865e:       01 c2                   add    %eax,%edx
 8048660:       8b 4d f0                mov    -0x10(%ebp),%ecx
 8048663:       8b 45 08                mov    0x8(%ebp),%eax
 8048666:       01 c8                   add    %ecx,%eax
 8048668:       0f b6 00                movzbl (%eax),%eax
 804866b:       83 c0 01                add    $0x1,%eax
 804866e:       88 02                   mov    %al,(%edx)
 8048670:       83 45 f0 01             addl   $0x1,-0x10(%ebp)
 8048674:       8b 45 f0                mov    -0x10(%ebp),%eax
 8048677:       3b 45 f4                cmp    -0xc(%ebp),%eax
 804867a:       7c b8                   jl     8048634 <encriptar_password+0x27>
 804867c:       c9                      leave
 804867d:       c3                      ret
```

Vemos que hay un for por lo siguiente:

jmp 8048674

cmp -0xc(%ebp),%eax → compara si el bucle for ha acabado.

jl 8048634 → si no ha acabado, va al principio del bucle.

```
mov    -0x10(%ebp),%eax
and    $0x1,%eax
test   %eax,%eax
jne    8048658 <encriptar_password+0x4b>
```

→Parece ser un if pass[i] es par, aunque no lo sé a ciencia cierta. Vamos a tomarlo como si así fuera.

```
mov    -0x10(%ebp),%edx
mov    0x8(%ebp),%eax
add    %eax,%edx
mov    -0x10(%ebp),%ecx
mov    0x8(%ebp),%eax
add    %ecx,%eax
movzbl (%eax),%eax
add    $0x2,%eax
mov    %al,(%edx)
jmp    8048670 <encriptar_password+0x63>
```

→ Este código es al que accede si se cumple el if anterior. La clave está en el add $0x2, %eax.

Si pass[i] es par, pass[i]+=2

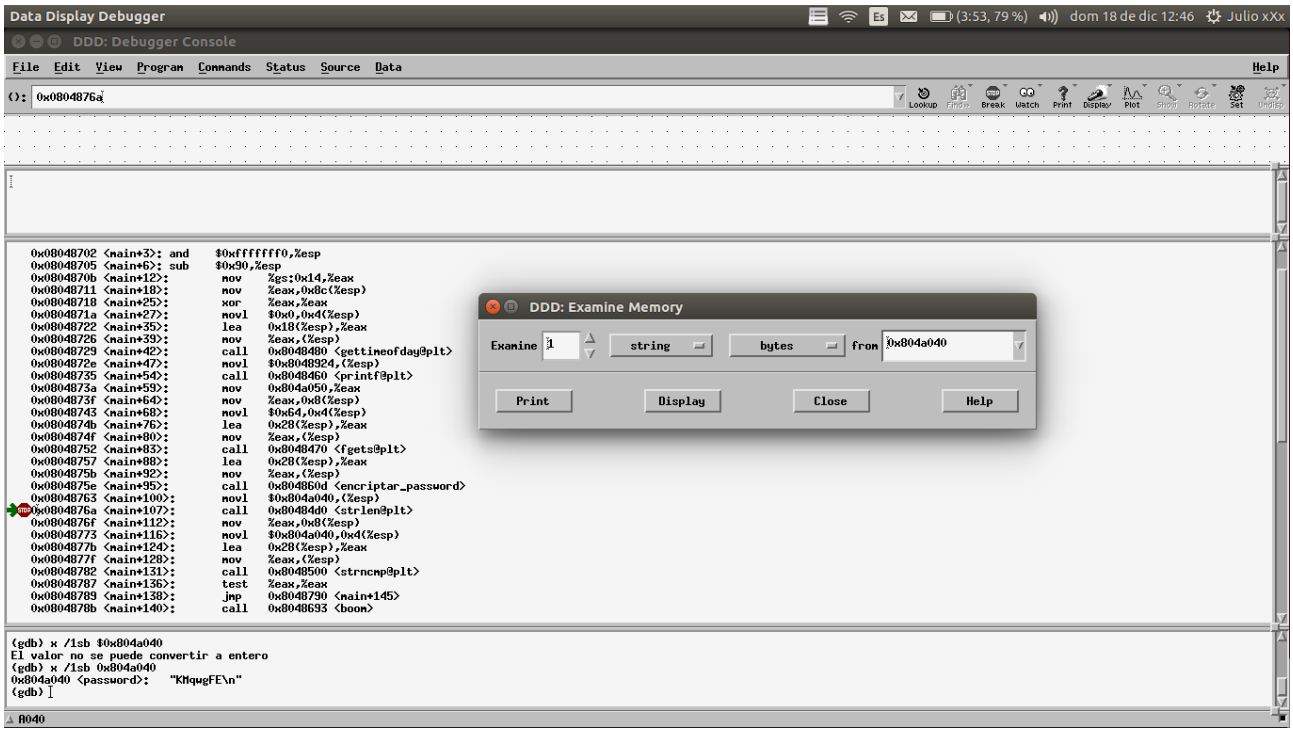
```
mov    -0x10(%ebp),%edx
mov    0x8(%ebp),%eax
add    %eax,%edx
mov    -0x10(%ebp),%ecx
mov    0x8(%ebp),%eax
add    %ecx,%eax
movzbl (%eax),%eax
add    $0x1,%eax
mov    %al,(%edx)
addl   $0x1,-0x10(%ebp)
```

→ Este código es al que accede si NO se cumple el if anterior. No se ve tan claro como ántes, pero parece que hace pass[i]+=1

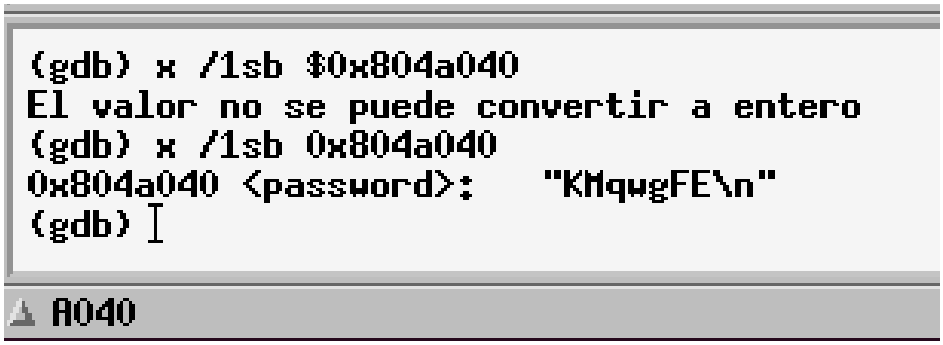
En conclusión, el algoritmo parece el siguiente:

```
for( int i=0; i<strlen(pass); i++ ){
        if( pass[i] %2 == 0 )
                pass[i]+=2;
        else pass[i]+=1;
}
```

Vamos a buscar la contraseña para aplicarle inversamente el algoritmo. Para eso, usaremos ddd.



Mostramos el string justo antes de la llamada a strlen (función que llama a password).



La contraseña encriptada es KmqwgFE.

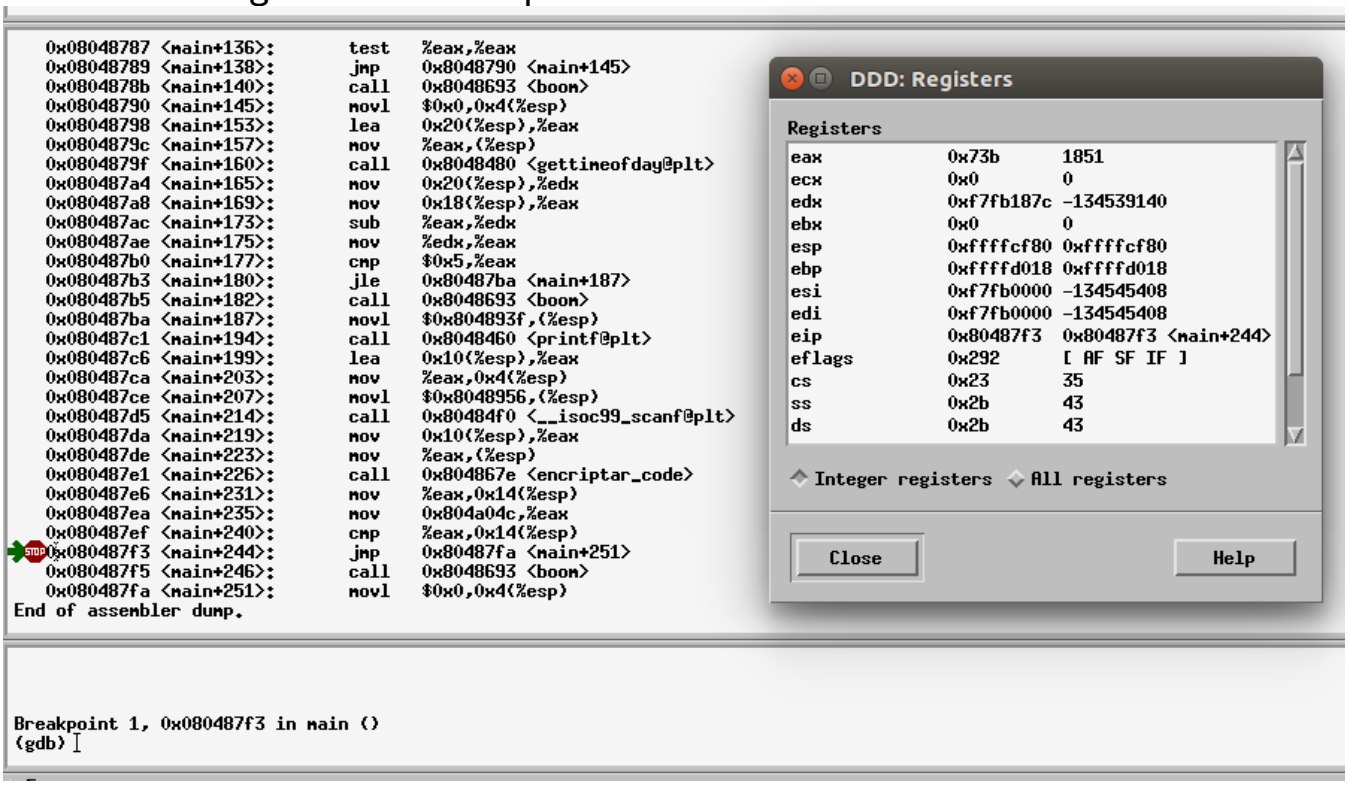
Aplicándole el algoritmo a la inversa, KmqwgFE == IloveEC
La contraseña tiene sentido, podemos llegar a la conclusión que el algoritmo usado es el correcto.

Ahora vamos a desencriptar el código

```
0804867e <encriptar_code>:
 804867e:       55                      push   %ebp
 804867f:       89 e5                   mov    %esp,%ebp
 8048681:       83 ec 10                sub    $0x10,%esp
 8048684:       8b 45 08                mov    0x8(%ebp),%eax
 8048687:       89 45 fc                mov    %eax,-0x4(%ebp)
 804868a:       83 6d fc 7b             subl   $0x7b,-0x4(%ebp)
 804868e:       8b 45 fc                mov    -0x4(%ebp),%eax
 8048691:       c9                      leave
 8048692:       c3                      ret
```

Es más fácil que el pass, la clave está en subl $0x7b, -0x4(%ebp)
Ésa orden le resta 0x7b al code original. 0x7b está en hexadecimal, equivale a 123 en decimal.

Vamos a averiguar el code encriptado.



Miramos el valor del registro justo al comparar el code introducido y el válido. Vemos que %eax tiene 1851. Lo desencriptamos (sumándole 123) y resulta 1974.

Ya tenemos la contraseña (ILoveEC) y el código (1974).

Bomba de Jose:

Desensamblamos la bomba con objdump -d:

```
0804860b <encripta>:
 804860b:       55                      push   %ebp
 804860c:       89 e5                   mov    %esp,%ebp
 804860e:       83 ec 18                sub    $0x18,%esp
 8048611:       c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%ebp)
 8048618:       eb 4b                   jmp    8048665 <encripta+0x5a>
 804861a:       c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)
 8048621:       eb 38                   jmp    804865b <encripta+0x50>
 8048623:       8b 55 f0                mov    -0x10(%ebp),%edx
 8048626:       8b 45 08                mov    0x8(%ebp),%eax
 8048629:       01 d0                   add    %edx,%eax
 804862b:       0f b6 00                movzbl (%eax),%eax
 804862e:       3c 7a                   cmp    $0x7a,%al
 8048630:       75 0d                   jne    804863f <encripta+0x34>
 8048632:       8b 55 f0                mov    -0x10(%ebp),%edx
 8048635:       8b 45 08                mov    0x8(%ebp),%eax
 8048638:       01 d0                   add    %edx,%eax
 804863a:       c6 00 61                movb   $0x61,(%eax)
 804863d:       eb 18                   jmp    8048657 <encripta+0x4c>
 804863f:       8b 55 f0                mov    -0x10(%ebp),%edx
 8048642:       8b 45 08                mov    0x8(%ebp),%eax
 8048645:       01 d0                   add    %edx,%eax
 8048647:       8b 4d f0                mov    -0x10(%ebp),%ecx
 804864a:       8b 55 08                mov    0x8(%ebp),%edx
 804864d:       01 ca                   add    %ecx,%edx
 804864f:       0f b6 12                movzbl (%edx),%edx
 8048652:       83 c2 01                add    $0x1,%edx
 8048655:       88 10                   mov    %dl,(%eax)
 8048657:       83 45 f4 01             addl   $0x1,-0xc(%ebp)
 804865b:       83 7d f4 02             cmpl   $0x2,-0xc(%ebp)
 804865f:       7e c2                   jle    8048623 <encripta+0x18>
 8048661:       83 45 f0 01             addl   $0x1,-0x10(%ebp)
 8048665:       83 ec 0c                sub    $0xc,%esp
 8048668:       ff 75 08                pushl  0x8(%ebp)
 804866b:       e8 50 fe ff ff          call   80484c0 <strlen@plt>
 8048670:       83 c4 10                add    $0x10,%esp
 8048673:       89 c2                   mov    %eax,%edx
 8048675:       8b 45 f0                mov    -0x10(%ebp),%eax
 8048678:       39 c2                   cmp    %eax,%edx
 804867a:       77 9e                   ja     804861a <encripta+0xf>
 804867c:       90                      nop
 804867d:       c9                      leave
 804867e:       c3                      ret
```

Igual que antes, vemos que hay un for desde 0 hasta strlen(pass):

```
jmp 8048665
cmp %eax, %edx
ja 804861a
```

Dentro de ese for, podemos ver un for más:

```
jmp 804865b
cmpl $0x2, -0xc(%ebp)
jle 8048623
```

Éste for parece que va desde 0 hasta 2. Vamos a analizar qué hay dentro de éste segundo for.

```
jmp    804865b <encripta+0x50>
mov    -0x10(%ebp),%edx
mov    0x8(%ebp),%eax
add    %edx,%eax
movzbl (%eax),%eax
cmp    $0x7a,%al
jne    804863f <encripta+0x34>
mov    -0x10(%ebp),%edx
mov    0x8(%ebp),%eax
add    %edx,%eax
movb   $0x61,(%eax)
jmp    8048657 <encripta+0x4c>
mov    -0x10(%ebp),%edx
mov    0x8(%ebp),%eax
add    %edx,%eax
mov    -0x10(%ebp),%ecx
mov    0x8(%ebp),%edx
add    %ecx,%edx
movzbl (%edx),%edx
```

→ cmp $0x7a, %al me costó bastante saber que hacía, pero llegué a la conclusión de que compara pass[i] con el carácter equivalente al número, el cual es 'z'.

→ si lo es, hace movb $0x61 a pass[i], es decir, pass[i] = 'a'.

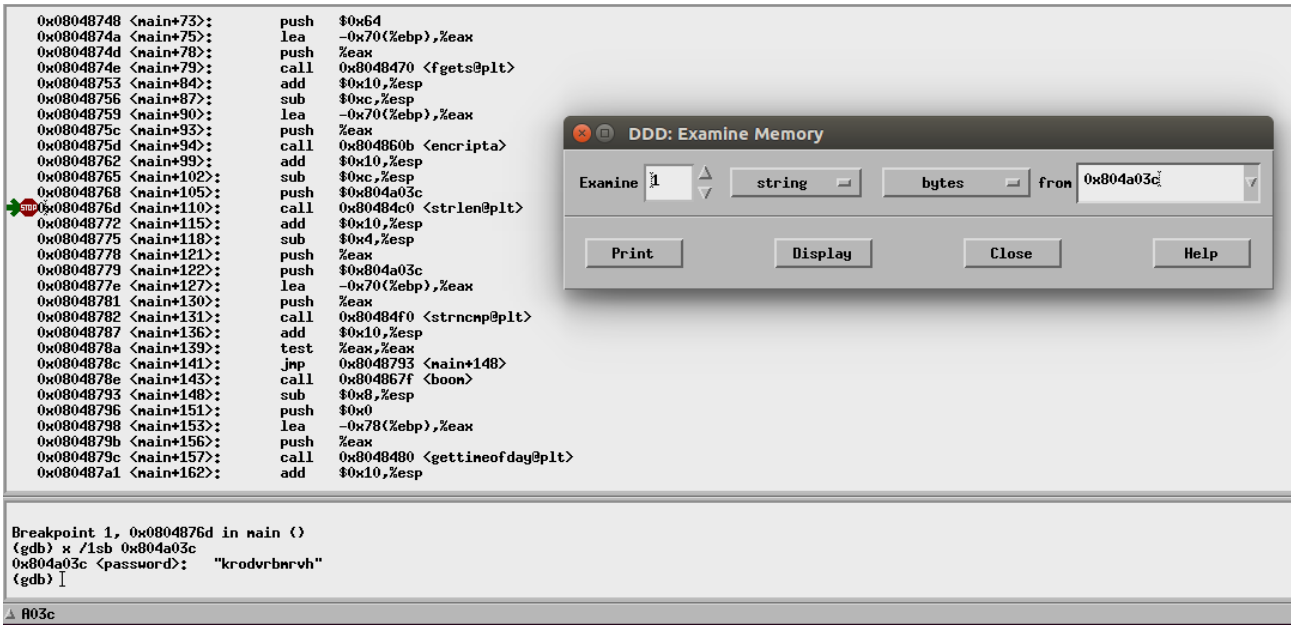
```
jmp    8048657 <encripta+0x4c>
mov    -0x10(%ebp),%edx
mov    0x8(%ebp),%eax
add    %edx,%eax
mov    -0x10(%ebp),%ecx
mov    0x8(%ebp),%edx
add    %ecx,%edx
movzbl (%edx),%edx
add    $0x1,%edx
mov    %dl,(%eax)
addl   $0x1,-0xc(%ebp)
cmpl   $0x2,-0xc(%ebp)
jle    8048623 <encripta+0x18>
addl   $0x1,-0x10(%ebp)
sub    $0xc,%esp
pushl  0x8(%ebp)
```

→ si no se cumple el if anterior, en add $0x1, %edx se ve que hace pass[i] += 1.

En conclusión, el algoritmo es el siguiente:

```
for( int i=0; i<strlen(pass); i++ ){
        for( int j=0; j<2; j++ ){
                if( pass[i] == 'z' )
                        pass[i] = 'a'
                else pass[i] += 1
        }
}
```

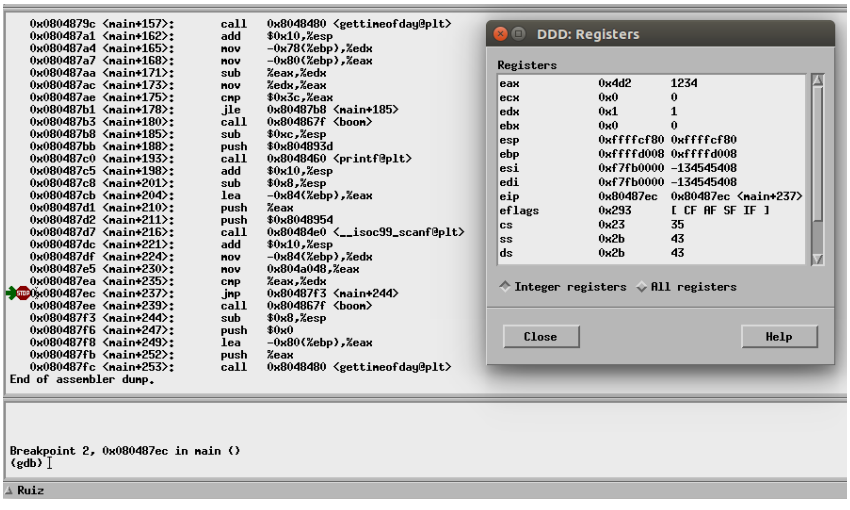
Vamos a buscar la contraseña encriptada.



La contraseña encriptada es “krodovrbmrvh”.

Aplicándole inversamente el algoritmo, el resultado es “holasoyjose”.

El code no tiene encriptación, vamos a mirar cual es.



→ Miramos el code en el mismo lugar donde miramos el de Adrián, y vemos que es 1234.

En resumen:

Bomba de Adrián:
Password: ILoveEC
Passcode: 1974

Bomba de Jose:
Password: holasoyjose
Passcode: 1234

Fin de Práctica 4