Seguridad Ofensiva 2020: Trabajo Práctico 5

Federico Juan Badaloni y Damián Ariel Marotte

29 de noviembre de 2020

Ejercicio 1

Encontramos un crash corriendo «`zzuf -c -s0:10000 -r 0.0001:0.001 ./parse mono.bmp`». El resultado puede reproducirse con la seed `s=36`.

Posteriormente se generamos el archivo malicioso «`cat mono.bmp | zzuf -cvq -s36 -r 0.0001:0.001 > error.bmp`».

Si analizamos la ejecución con gdb (estableciendo un breakpoint en la linea 74) puede observarse que la variable «`infoheader.ncolours`» vale «524288», por lo que «`i`» tarde o temprano asumirá valores mayores a 255 y la posterior llamada a la función «`read`» escribirá en «`colourindex[i]`». Sin embargo puede observarse en la linea 40 que dicho arreglo solo tiene capacidad para 255 valores, por lo que puede desbordarse y modificar la *return adress* de `parse`, lo cual modifica el EIP.

Ejercicio 2

El siguiente script Python utiliza angr para buscar un camino de ejecución que llegué a ejecutar la dirección de memoria donde se imprime la flag.

```
import angr, claripy

base_address    = 0x08048000
success_address = 0x08048570
failure_address = 0x0804852d
pass_length     = 20
binary_name     = "r1"

proj = angr.Project(
        binary_name
        # main_opts    = {'base_addr': base_address},
        # load_options = {'auto_load_libs': False}
)

pass_chars = [
        claripy.BVS(f"pass_char{i}", 8)
        for i in range(pass_length)
]
password   = claripy.Concat(*pass_chars)
```

```
state = proj.factory.full_init_state(
        # args = ["./" + binary_name],
        add_options = angr.options.unicorn,
        stdin = angr.SimFileStream(
                        name='stdin',
                        content=password, has_end=False)
)

sim_mgr = proj.factory.simulation_manager(state)
sim_mgr.explore(find = success_address, avoid = failure_address)

if len(sim_mgr.found):
    for found in sim_mgr.found:
        print(found.posix.dumps(0))
```

Ejercicio 3

1. Se puede instalar `fmem` que crea el dispositivo `/dev/fmem` con el cual podemos dumpear la memoria usando `sudo dd if=/dev/fmem of=/tmp/memory.raw bs=1MB` (notar que se necesitan los *headers* del kernel de Linux).

2. [TODO: LiME]

Ejercicio 4

[TODO: nos falta crackear esto]

Ejercicio 5

Luego de encontrar un perfil de memoria adecuado para el dump en cuestión, pudimos encontrar en rootkit usando el comando `linux_check_modules` de volatility.

strings ubuntu-10.04.3-i386-LiveCD-kbeast.mem | grep "kbeast" -i