Seguridad Ofensiva 2020: Trabajo Práctico 5

Federico Juan Badaloni y Damián Ariel Marotte

30 de noviembre de 2020

Ejercicio 1

La utilidad «`zzuf`» permite realizar pequeñas modificaciones a una entrada válida de un programa, con el fin de encontrar otra entrada que logre un funcionamiento indebido.

La opción de linea de comandos «-s» permite indicar las diferentes semillas con las que se alterará la entada y la opción «-c» establece el radio de perturbación generado.

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 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. La llamada a la función «`read`» escribirá en «`colourindex[i]`» pero puede observarse en la linea 40 que dicho arreglo solo tiene capacidad para 255 valores, por lo que puede desbordarse y modificar la *return address* de la función `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}
)
```

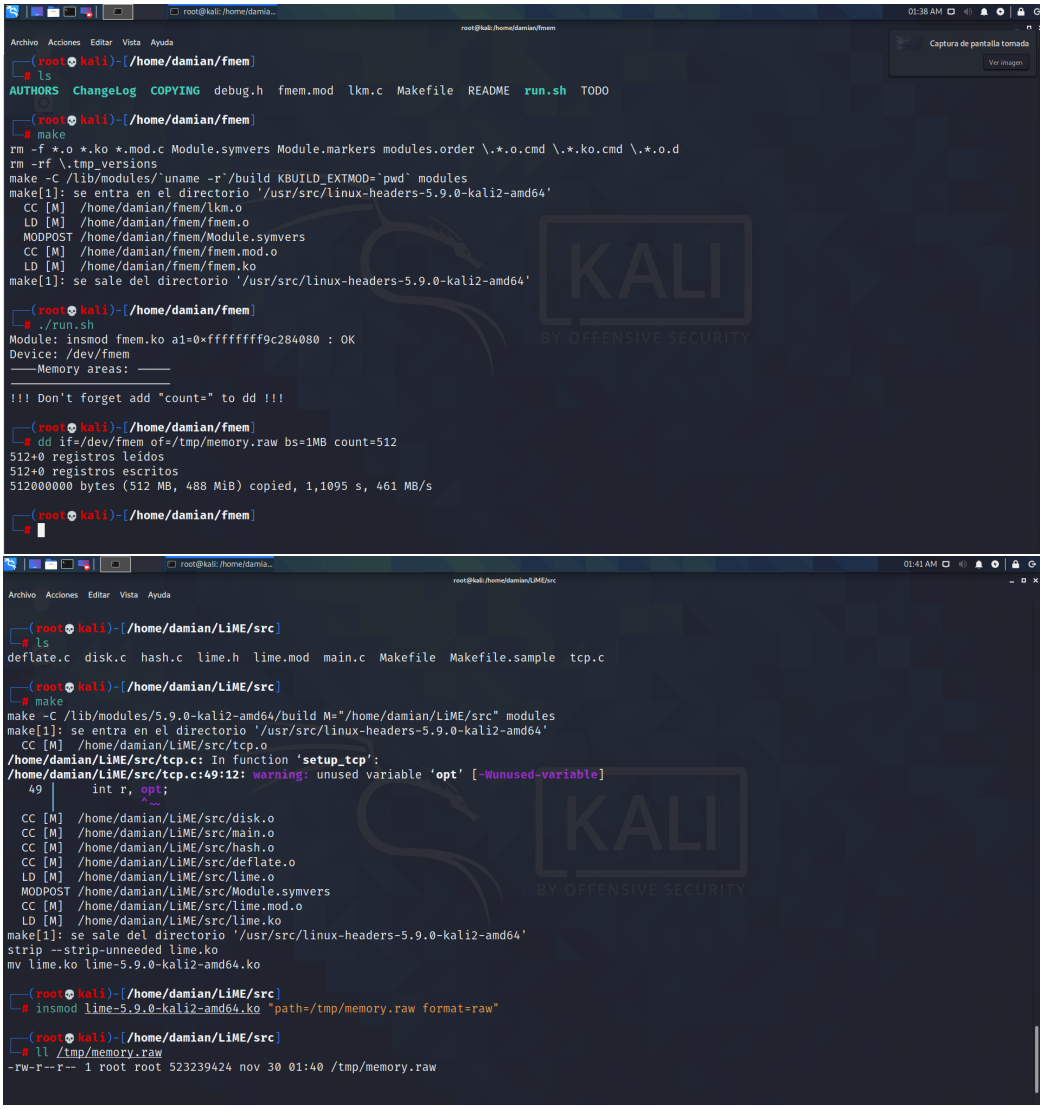
```
pass_chars = [
        claripy.BVS(f"pass_char{i}", 8)
        for i in range(pass_length)
]
password   = claripy.Concat(*pass_chars)

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. El modulo del kernel de Linux «`fmem`», crea el dispositivo `/dev/fmem` con el cual podemos realizar un voldado de la memoria usando el comando `sudo dd if=/dev/fmem of=/tmp/memory.raw bs=1MB count=512` (notar que se necesitan los *headers* del kernel de Linux para su correcta compilación).

2. «LiME» es otro modulo que provee la misma funcionalidad. Luego de compilarlo, el volcado de la memoria se puede realizar con el comando `sudo insmod lime-5.9.0-kali2-amd64.ko "path=/tmp/memory.raw format=raw"`.

A continuación se adjuntan las imágenes que documentan dicho proceso con ambas herramientas.



Ejercicio 4

Para poder recuperar la contraseña del archivo, primero debe extraerse el hash del archivo. Para ello utilizamos la herramienta Online Hash Crack, obteniendo:

«$office$*2007*20*128*16*ba1ae53b4d016fc3a15124b2a3034779*49a69de2853eac6c62cceeeb549aac18*57e5fd8bfd182b4c70071a3052b91194e048055c».

Finalmente basándonos en nuestra experiencia previa, utilizamos «`hashcat`» para recuperar la constraseña. Para ello nos bastó con un ataque de producto cartesiano.

La contraseña recuperada es: «`immyisno.1saop91`».

Ejercicio 5

Para poder analizar la memoria, es necesario contar con un perfil del sistema operativo apropiado. En la siguiente URL pueden encontrarse varios perfiles, entre ellos el que utilizamos con volatility:

https://github.com/volatilityfoundation/profiles/tree/master/Linux/Ubuntu/x86

Luego de configurar volatility, pudimos encontrar en rootkit usando el comando: «`./volatility_2.6_lin64_standalone -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem --profile=LinuxUbuntu10043x86 linux_check_modules`».

En forma alternativa, el comando «`strings ubuntu-10.04.3-i386-LiveCD-kbeast.mem | grep kbeast`» provee suficiente evidencia para confirmar lo que sospechabamos.