

TAREA 3 - ROOTKIT



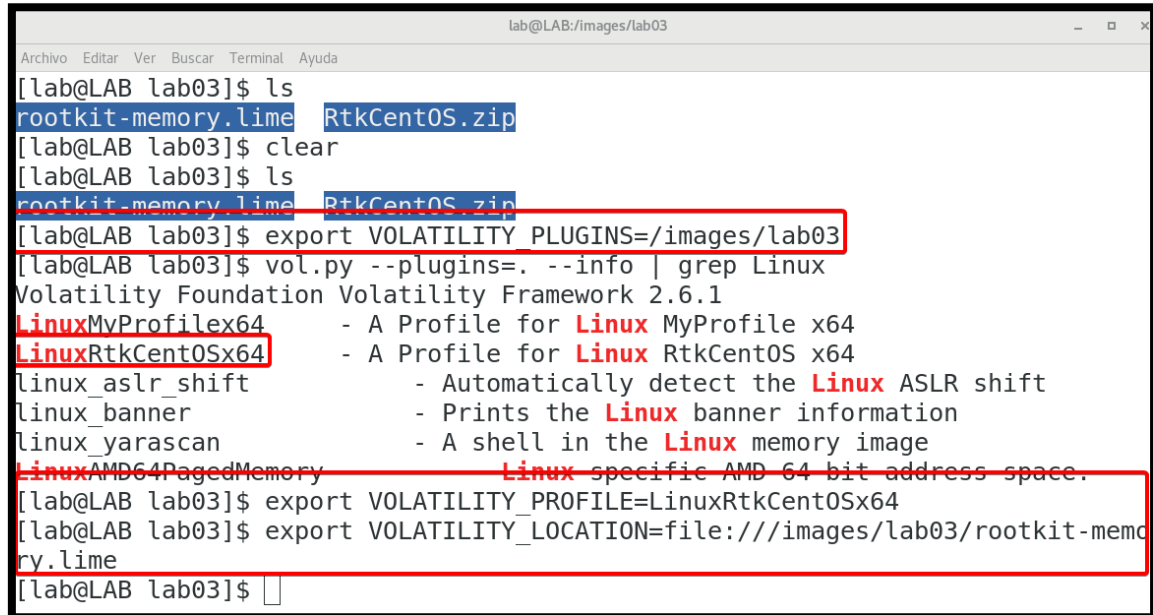
ERIC SERRANO MARÍN ANÁLISIS FORENSE INFORMÁTICO

Contenido

1. Defina con “export” una variable para la ubicación de los plugins, otra para el nombre del perfil y otra para la localización de la memoria. Muestre una captura de pantalla de las 3 variables declaradas..... 3
2. Ejecute “vol.py Linux_banner” y muestre su resultado. Indique qué sistema operativo utilizaba esta memoria..... 3
3. Indique el comando y sus parámetros, suponiendo que usted quiera insertar un módulo, malicioso o no, en el kernel. 3
4. Ejecute el complemento linux_lsmod y muestre únicamente el nombre de todos los módulos ordenados alfabéticamente. Muestre el resultado en dos columnas..... 4
5. Ejecute y muestre el resultado de un complemento que utilice “sysfs”. ¿Por qué no aparece en el listado del apartado 4? 4
6. Ejecute y muestre el resultado de un complemento que “Trabaja la memoria para encontrar módulos del kernel ocultos”..... 4
7. Ejecute un complemento que lea el buffer dmesg y almacene su contenido en “dmesg.txt”. 5
8. Muestre todas las líneas que muestren información del posible módulo rootkit anteriormente descubierto. 5
9. Compruebe que únicamente el módulo analizado es el único que presuntamente está contaminando el kernel..... 5
10. Explique brevemente la utilidad de los comandos anteriores..... 6
11. Con los comandos anteriores, detecte todos los HOOKED producidos. Indique los 3 nombres de las interfaces resultantes y qué significado podrían tener en Linux (intentar descubrir para qué se utilizan a través de su nombre) . 7
12. Ejecute un complemento que recupere el historial de bash en la memoria y muestre el resultado que esté relacionado con el supuesto rootkit. Almacénelo en un archivo..... 7

13. En qué directorio supuestamente estaba/trabajaba el rootkit. ¿A qué hora/s se insertó en el kernel? 7
2017-07-07 15:52:45 UTC+0000..... 7
14. Con la ayuda de “linux_find_file” de volatity, encuentre el path completo y los nombres de los archivos incluidos del rootkit..... 8
15. Busque las 3 llamadas al sistema en el historial del bash del apartado 11.
8
16. De una de ellas encuentra el parámetro “-31”, ¿para qué se utiliza el -31?
8
17. Utilice todos estos comandos para intentar averiguar información del PID 1078. También utilice “netstat” en memoria..... 9
18. Deduzca qué es posible que realice el rootkit a través del PID 1078 y por qué luego ejecuta kill.11

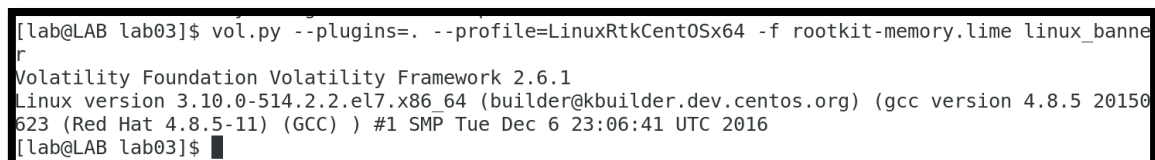
1. Defina con “export” una variable para la ubicación de los plugins, otra para el nombre del perfil y otra para la localización de la memoria. Muestre una captura de pantalla de las 3 variables declaradas.



```
lab@LAB: /images/lab03
[lab@LAB lab03]$ ls
rootkit-memory.lime  RtkCentOS.zip
[lab@LAB lab03]$ clear
[lab@LAB lab03]$ ls
rootkit-memory.lime  RtkCentOS.zip
[lab@LAB lab03]$ export VOLATILITY_PLUGINS=/images/lab03
[lab@LAB lab03]$ vol.py --plugins=. --info | grep Linux
Volatility Foundation Volatility Framework 2.6.1
LinuxMyProfilex64 - A Profile for Linux MyProfile x64
LinuxRtkCentOSx64 - A Profile for Linux RtkCentOS x64
linux_aslr_shift - Automatically detect the Linux ASLR shift
linux_banner - Prints the Linux banner information
linux_yarascan - A shell in the Linux memory image
LinuxAMD64PagedMemory - Linux specific AMD 64 bit address space.
[lab@LAB lab03]$ export VOLATILITY_PROFILE=LinuxRtkCentOSx64
[lab@LAB lab03]$ export VOLATILITY_LOCATION=file:///images/lab03/rootkit-memory.lime
[lab@LAB lab03]$
```

2. Ejecute “vol.py Linux_banner” y muestre su resultado. Indique qué sistema operativo utilizaba esta memoria.

La memoria estaba en un sistema operativo CentOS, con la versión del kernel 3.10.0-514.2.2.el7.x86_64.



```
[lab@LAB lab03]$ vol.py --plugins=. --profile=LinuxRtkCentOSx64 -f rootkit-memory.lime linux_banner
Volatility Foundation Volatility Framework 2.6.1
Linux version 3.10.0-514.2.2.el7.x86_64 (builder@kbuilder.dev.centos.org) (gcc version 4.8.5 20150623 (Red Hat 4.8.5-11) (GCC) ) #1 SMP Tue Dec 6 23:06:41 UTC 2016
[lab@LAB lab03]$
```

3. Indique el comando y sus parámetros, suponiendo que usted quiera insertar un módulo, malicioso o no, en el kernel.

insmod /ruta/al/módulo.ko

4. Ejecute el complemento `linux_lsmod` y muestre únicamente el nombre de todos los módulos ordenados alfabéticamente. Muestre el resultado en dos columnas.

```
[lab@LAB lab03]$ vol.py --plugins=. --profile=LinuxRtkCentOSx64 -f rootkit-memory.lime linux_lsmod | sort -k2 | pr -2 -t
Volatility Foundation Volatility Framework 2.6.1
ffffffffffa05b1000 ablk_helper 13597 fffffffffffa0162d80 libata 247095
ffffffffffa05d0c0 ac97_bus 12730 fffffffffffa041ff80 libnvdimm 126631
ffffffffffa05d1240 aesni_intel 69884 fffffffffffa075a000 lime 18110
ffffffffffa0051640 ata_generic 12910 fffffffffffa069d0e0 llc 14552
ffffffffffa011b240 ata_piix 35038 fffffffffffa02f9880 lockd 93573
ffffffffffa036e680 auth_rpcgss 59323 fffffffffffa05c0080 lrw 13286
ffffffffffa04d0280 bluetooth 534918 fffffffffffa00b1080 mbcache 14958
ffffffffffa03e3560 bnep 19625 fffffffffffa006c040 mptbase 105960
ffffffffffa06bdc80 bridge 107106 fffffffffffa007f200 mptscsih 40150
ffffffffffa0463000 btbcm 14040 fffffffffffa0075420 mptspi 22542
```

5. Ejecute y muestre el resultado de un complemento que utilice “sysfs”. ¿Por qué no aparece en el listado del apartado 4?

No aparece en el listado porque se trata de un rootkit, y está diseñado para ocultar su presencia en el sistema. Aunque `diamorphine` es un módulo malicioso y su propósito es ocultar su presencia, es posible que esté diseñado para no aparecer en la lista de módulos cargados cuando se usa `Linux_lsmod`. Sin embargo, al utilizar herramientas de análisis forense como `Linux_check_modules`, es posible detectarla.

```
[lab@LAB lab03]$ vol.py --plugins=. --profile=LinuxRtkCentOSx64 -f rootkit-memory.lime linux_check_modules
Volatility Foundation Volatility Framework 2.6.1
Module Address      Core Address      Init Address Module Name
-----
0xfffffffffa0747000 0xfffffffffa0745000 0x0 diamorphine
[lab@LAB lab03]$
```

6. Ejecute y muestre el resultado de un complemento que “Trabaja la memoria para encontrar módulos del kernel ocultos”

```
[lab@LAB lab03]$ vol.py linux_hidden_modules
Volatility Foundation Volatility Framework 2.6.1
Offset (V)          Name
-----
0xfffffffffa0747000 diamorphine
[lab@LAB lab03]$
```

7. Ejecute un complemento que lea el buffer dmesg y almacene su contenido en “dmesg.txt”.

```
[lab@LAB lab03]$ vol.py linux_dmesg > dmesg.txt
Volatility Foundation Volatility Framework 2.6.1
[lab@LAB lab03]$ ls
dmesg.txt  rootkit-memory.lime  RtkCentOS.zip
[lab@LAB lab03]$
```

8. Muestre todas las líneas que muestren información del posible módulo rootkit anteriormente descubierto.

```
[lab@LAB lab03]$ cat dmesg.txt | grep diamorphine
[24885243126951.24885] diamorphine: loading out-of-tree module taints kernel.
[24885243234803.24885] diamorphine: module verification failed: signature and/or required key missing -
tainting kernel
[24885306579597.24885] systemd-udevd[10995]: no db file to read /run/udev/data/+module:diamorphine: No
such file or directory
[24885306638248.24885] systemd-udevd[10995]: unable to access usb_interface device of '/sys/module/diam
orphine'
```

9. Compruebe que únicamente el módulo analizado es el único que presuntamente está contaminando el kernel.

```
[lab@LAB lab03]$ cat dmesg.txt | grep taint
[24885243126951.24885] diamorphine: loading out-of-tree module taints kernel.
[24885243234803.24885] diamorphine: module verification failed: signature and/or required key missing -
tainting kernel
[lab@LAB lab03]$
```

10. Explique brevemente la utilidad de los comandos anteriores.

COMANDOS USADOS ANTERIORMENTE	
export PLUGINS/PROFILE/LOCATION	Establecen las variables de entorno para configurar Volatility, así no tener que poner siempre el perfil, ni el archivo.
vol.py linux_banner	Extrae el banner de Linux del volcado de memoria, esto proporciona información básica de la versión y la arquitectura del kernel de Linux.
vol.py linux_lsmod	Lista los módulos del kernel cargados en el sistema. Muestra información detallada sobre cada módulo.
vol.py linux_check_modules	Verifica la integridad de los módulos del kernel cargados en el sistema, buscando signos de manipulación u ocultamiento de módulos.
vol.py linux_hidden_modules	Identifica los módulos del kernel que podrían estar ocultos o enmascarados para evadir la detección tradicional.
vol.py linux_dmesg > dmesg.txt	Extrae el registro de mensajes del kernel (dmesg) del volcado y lo redirige a un archivo de texto.
cat dmesg.txt grep taint	Busca en el archivo dmesg.txt la palabra taint, que significa corrupción, infección...
vol.py linux_check_idt	Analiza Tabla de Descriptores de Interrupción (IDT) en sistemas Linux en busca de posibles signos de manipulación o actividad maliciosa.
vol.py linux_check_syscall	Examina las tablas de sistema en sistemas Linux para detectar modificaciones o discrepancias que puedan indicar la presencia de actividad maliciosa o manipulación de llamadas al sistema.

11. Con los comandos anteriores, detecte todos los HOOKED producidos. Indique los 3 nombres de las interfaces resultantes y qué significado podrían tener en Linux (intentar descubrir para qué se utilizan a través de su nombre)

```
[lab@LAB lab03]$ vol.py linux_check_syscall | grep HOOKED
Volatility Foundation Volatility Framework 2.6.1
64bit          62          0xfffffffffa0745540 HOOKED: diamorphine/hacked_kill
64bit          78          0xfffffffffa0745080 HOOKED: diamorphine/hacked_getdents
64bit          217         0xfffffffffa0745230 HOOKED: diamorphine/hacked_getdents64
```

- **Diamorphine/hacked_kill**: modifica la llamada al sistema 'kill' (que lo que hace es matar un proceso), para que los procesos maliciosos eviten ser terminados o alteren su comportamiento.
- **Hacked_gents y getdents64**, estas llamadas se alteran para ocultar ciertos archivos o directorios del sistema, dificultando la detección de actividades maliciosas.

12. Ejecute un complemento que recupere el historial de bash en la memoria y muestre el resultado que esté relacionado con el supuesto rootkit. Almacénelo en un archivo.

```
10435 bash 2017-07-07 15:57:30 UTC+0000 insmod diamorphine.ko
[lab@LAB lab03]$ vol.py linux_bash | grep diamorphine > bash_diamorphine.txt
Volatility Foundation Volatility Framework 2.6.1
[lab@LAB lab03]$ cat bash_diamorphine.txt
10210 bash 2017-07-07 15:50:49 UTC+0000 vi diamorphine.h
10435 bash 2017-07-07 15:52:45 UTC+0000 cd zaq123edcx-diamorphine/
10435 bash 2017-07-07 15:52:45 UTC+0000 insmod diamorphine.ko
10435 bash 2017-07-07 15:52:45 UTC+0000 insmod diamorphine.ko
10435 bash 2017-07-07 15:52:45 UTC+0000 cd zaq123edcx-diamorphine/
10435 bash 2017-07-07 15:52:45 UTC+0000 insmod diamorphine.ko
10435 bash 2017-07-07 15:52:45 UTC+0000 cd zaq123edcx-diamorphine/
10435 bash 2017-07-07 15:52:45 UTC+0000 insmod diamorphine.ko
10435 bash 2017-07-07 15:52:45 UTC+0000 cd zaq123edcx-diamorphine/
10435 bash 2017-07-07 15:52:45 UTC+0000 insmod diamorphine.ko
10435 bash 2017-07-07 15:52:45 UTC+0000 cd ~sans/zaq123edcx-diamorphine/
10435 bash 2017-07-07 15:57:30 UTC+0000 insmod diamorphine.ko
[lab@LAB lab03]$
```

13. En qué directorio supuestamente estaba/trabajaba el rootkit. ¿A qué hora/s se insertó en el kernel?

2017-07-07 15:52:45 UTC+0000

```
10435 bash 2017-07-07 15:52:45 UTC+0000 cd zaq123edcx-diamorphine/
10435 bash 2017-07-07 15:52:45 UTC+0000 insmod diamorphine.ko
10435 bash 2017-07-07 15:52:45 UTC+0000 insmod diamorphine.ko
```


14. Con la ayuda de “linux_find_file” de volatility, encuentre el path completo y los nombres de los archivos incluidos del rootkit.

```
[lab@LAB lab03]$ vol.py linux_find_file -L | grep diamorphine
Volatility Foundation Volatility Framework 2.6.1
30969 0xffff880046540b90 /sys/module/diamorphine
30971 0xffff880046542500 /sys/module/diamorphine/uevent
1308506 0xffff8800484b9900 /home/sans/zaq123edcx-diamorphine
1308565 0xffff880024419900 /home/sans/zaq123edcx-diamorphine/.diamorphine.ko.cmd
1308564 0xffff8800244194f8 /home/sans/zaq123edcx-diamorphine/.diamorphine.mod.o.cmd
1308563 0xffff8800244190f0 /home/sans/zaq123edcx-diamorphine/diamorphine.ko
1308556 0xffff880024418ce8 /home/sans/zaq123edcx-diamorphine/diamorphine.mod.o
1308555 0xffff8800244188e0 /home/sans/zaq123edcx-diamorphine/Module.symvers
1308554 0xffff8800244184d8 /home/sans/zaq123edcx-diamorphine/diamorphine.mod.c
1308553 0xffff8800244180d0 /home/sans/zaq123edcx-diamorphine/modules.order
1308552 0xffff8800467fd980 /home/sans/zaq123edcx-diamorphine/.diamorphine.o.cmd
1308549 0xffff8800244079c0 /home/sans/zaq123edcx-diamorphine/diamorphine.h
1308530 0xffff8800244075b8 /home/sans/zaq123edcx-diamorphine/diamorphine.o
1308523 0xffff8800244071b0 /home/sans/zaq123edcx-diamorphine/diamorphine.c
1308516 0xffff880024406da8 /home/sans/zaq123edcx-diamorphine/README.md
1308515 0xffff8800244069a0 /home/sans/zaq123edcx-diamorphine/Makefile
1308514 0xffff880024406598 /home/sans/zaq123edcx-diamorphine/LICENSE.txt
1308527 0xffff8800484b9d08 /home/sans/zaq123edcx-diamorphine/.tmp_versions
[lab@LAB lab03]$
```

15. Busque las 3 llamadas al sistema en el historial del bash del apartado 11.

```
[lab@LAB lab03]$ vol.py linux_bash | grep kill
Volatility Foundation Volatility Framework 2.6.1
10435 bash 2017-07-07 15:52:45 UTC+0000 pkill -USR1 auditd
10435 bash 2017-07-07 15:52:45 UTC+0000 kill -31 1082
10435 bash 2017-07-07 15:52:45 UTC+0000 kill -31 1090
10435 bash 2017-07-07 15:52:45 UTC+0000 kill -31 14256
10435 bash 2017-07-07 15:52:45 UTC+0000 kill -31 1079
10435 bash 2017-07-07 15:57:43 UTC+0000 kill -31 1078
```

16. De una de ellas encuentra el parámetro “-31”, ¿para qué se utiliza el -31?

El parámetro -31 en el comando kill corresponde a la señal SIGUSR1 en Linux y se utiliza para enviar una señal específica al proceso especificado, solicitando un comportamiento particular asociado con esa señal. La acción exacta que realiza el proceso en respuesta a la señal depende de cómo esté diseñado para manejar esa señal.

17. Utilice todos estos comandos para intentar averiguar información del PID 1078. También utilice “netstat” en memoria.

El comando **linux_pslist** nos dice que el proceso con el ID 1078 se llama sshd (el demonio SSH). También nos da información como que el PPid que es 1.

```
[lab@LAB lab03]$ vol.py linux_pslist -p 1078
Volatility Foundation Volatility Framework 2.6.1
Offset      Name      Pid      PPid      Uid      Gid      DTB      Start Time
-----
0xffff8800757e4e70 sshd      1078      1          0          0      0x00000000780fd000 0
[lab@LAB lab03]$
```

Con **linux_psaux** tenemos información más detallada, como que la línea de comando utilizada para iniciar el proceso es '/usr/sbin/sshd'.

```
[lab@LAB lab03]$ vol.py linux_psaux -p 1078
Volatility Foundation Volatility Framework 2.6.1
Pid      Uid      Gid      Arguments
1078     0        0        /usr/sbin/sshd
[lab@LAB lab03]$
```

Con **linux_pstree** no vemos nada nuevo.

```
[lab@LAB lab03]$ vol.py linux_pstree -p 1078
Volatility Foundation Volatility Framework 2.6.1
Name      Pid      Uid
sshd      1078
[lab@LAB lab03]$
```

El plugin **linux_pslist_cache** no me funciona, ya que solo funciona en sistemas que usen SLAB (Segregated Linked Allocation Buffers), es un método de gestión de memoria.

```
[lab@LAB lab03]$ vol.py linux_pslist_cache -p 1078
Volatility Foundation Volatility Framework 2.6.1
Offset      Name      Pid      PPid      Uid      Gid      DTB      Start Time
-----
INFO : volatility.debug : SLUB is currently unsupported.
[lab@LAB lab03]$
```

linux_pidhashtable busca y muestra información específica sobre en este caso el proceso con PID 1078.

```
[lab@LAB lab03]$ vol.py linux_pidhashtable | grep 1078
Volatility Foundation Volatility Framework 2.6.1
0xfffff8800757e4e70 sshd 1078 1 0 0 0x00000000780fd000 0
```

EL COMANDO LINUX_PSVIEW TIENE VARIAS COLUMNAS

pslist	True = se puede ver el proceso usando pslist. False = no se puede ver.
psscan	Indica lo mismo, si el proceso se puede ver usando esta técnica.
pid_hash	Indica si el proceso está presente en la tabla del hash de PID del kernel.
kmem_cache	Indica si el proceso está presente en la caché de memoria del kernel.
parents	Indica si se puede encontrar el proceso padre del proceso especificado.
leaders	Indica si el proceso es líder de un grupo de procesos.

Y EL RESULTADO DE NUESTRO COMANDO HA SIDO EL SIGUIENTE

pslist	psscan	pid_hash	kmem_cache	parents	leaders
True	True	True	False	False	True

```
[lab@LAB lab03]$ cat psxview.txt | grep 1078
0x00000000757e4e70 sshd 1078 True True True False False True
```

linux_lsof proporciona información detallada sobre los descriptores de archivo asociados al proceso sshd con el PID 1078. Este comando es útil para comprender qué archivos y recursos está utilizando el proceso.

```
Volatility Foundation Volatility Framework 2.6.1
```

Offset	Name	Pid	FD	Path
0xffff8800757e4e70	sshd	1078	0	/dev
0xffff8800757e4e70	sshd	1078	1	/dev
0xffff8800757e4e70	sshd	1078	2	/dev
0xffff8800757e4e70	sshd	1078	3	socket:[20785]
0xffff8800757e4e70	sshd	1078	4	socket:[20787]

Por último, **linux_netstat**, nos proporciona información sobre las conexiones de red asociadas al proceso con el PID 1078. Esta salida nos indica que el proceso SSH está escuchando en los puertos TCP 22, tanto en IPv4 como en IPv6.

```
[lab@LAB lab03]$ vol.py linux_netstat -p 1078
```

```
Volatility Foundation Volatility Framework 2.6.1
```

TCP	0.0.0.0	:	22 0.0.0.0	:	0 LISTEN	sshd/1078
TCP	:::	:	22 :::	:	0 LISTEN	sshd/1078

18. Deduzca qué es posible que realice el rootkit a través del PID 1078 y por qué luego ejecuta kill.

El rootkit utiliza el proceso SSH con el PID 1078 para establecer una puerta trasera, ya que el demonio SSH es una herramienta legítima que permite conexiones remotas seguras al sistema. Los atacantes suelen aprovecharse de este proceso para establecer conexiones no autorizadas y mantener el acceso al sistema de forma discreta.

Después está el comando Kill ejecutado, que sugiere que el atacante está intentando ocultar sus actividades maliciosas. Ya que con este comando el atacante estaría tratando de detener procesos de monitoreo o herramientas de seguridad que podrían detectar su presencia.

Estas evidencias (creo) respaldan que el proceso SSH (sshd) con el PID 1078 está siendo utilizado por un rootkit para establecer una puerta trasera en el sistema, y que el comando "kill" se utiliza para ocultar y proteger las actividades del atacante después de la intrusión.