SKIP TO CONTENT

Perform Linux memory forensics with this open source tool

Find out what's going on with applications, network connections, kernel modules, files, and much more with Volatility

By Gaurav Kamathe (Correspondent, Red Hat)

April 27, 2021 | 0 Comments | 15 min read

Image by:

opensource.com

A computer's operating system and applications use the primary memory (or RAM) to perform various tasks. This volatile memory, containing a wealth of information about running applications, network connections, kernel modules, open files, and just about everything else is wiped out each time the computer restarts.

Memory forensics is a way to find and extract this valuable information from memory. Volatility is an open source tool that uses plugins to process this type of information. However, there's a problem: Before you can process this information, you must dump the physical memory into a file, and Volatility does not have this ability.

Therefore, this article has two parts:

- The first part deals with acquiring the physical memory and dumping it into a file.
- The second part uses Volatility to read and process information from this memory dump.

I used the following test system for this tutorial, but it will work on any Linux distribution:

```
$ cat /etc/redhat-release
Red Hat Enterprise Linux release 8.3 (Ootpa)
$
$ uname -r
4.18.0-240.el8.x86_64
$
```

> A note of caution: Part 1 involves compiling and loading a kernel module. Don't worry; it isn't as difficult as it sounds. Some guidelines:
>
> - Follow the steps.
> - Do not try any of these steps on a production system or your primary machine.
> - Always use a test virtual machine (VM) to try things out until you are comfortable using the tools and understand how they work.

Install the required packages

Before you get started, install the requisite tools. If you are using a Debian-based distro, use the equivalent `apt-get` commands. Most of these packages provide the required kernel information and tools to compile the code:

More Linux resources

- Linux commands cheat sheet
- Advanced Linux co

```
$ yum install kernel-headers kernel-devel gcc elfutils-libelf-devel make git libdwarf-tools python2-devel.x86_64-y
```

Part 1: Use LiME to acquire memory and dump it to a file

Before you can begin to analyze memory, you need a memory dump at your disposal. In an actual forensics event, this could come from a compromised or hacked system. Such information is often collected and stored to analyze how the intrusion happened and its impact. Since you probably do not have a memory dump available, you can take a memory dump of your test VM and use that to perform memory forensics.

Linux Memory Extractor (LiME) is a popular tool for acquiring memory on a Linux system. Get LiME with:

```
$ git clone https://github.com/504ensicsLabs/LiME.git
$
$ cd LiME/src/
$
$ ls
deflate.c  disk.c  hash.c  lime.h  main.c  Makefile  Makefile.sample  tcp.c
$
```

Build the LiME kernel module

Run the `make` command inside the `src` folder. This creates a kernel module with a .ko extension. Ideally, the `lime.ko` file will be renamed using the format `lime-<your-kernel-version>.ko` at the end of `make`:

```
$ make
make -C /lib/modules/4.18.0-240.el8.x86_64/build M="/root/LiME/src" modules
make[1]: Entering directory '/usr/src/kernels/4.18.0-240.el8.x86_64'

<< snip >>

make[1]: Leaving directory '/usr/src/kernels/4.18.0-240.el8.x86_64'
strip --strip-unneeded lime.ko
mv lime.ko lime-4.18.0-240.el8.x86_64.ko
$
$
```

```
$ ls -l lime-4.18.0-240.el8.x86_64.ko
-rw-r--r--. 1 root root 25696 Apr 17 14:45 lime-4.18.0-240.el8.x86_64.ko
$
$ file lime-4.18.0-240.el8.x86_64.ko
lime-4.18.0-240.el8.x86_64.ko: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV),
BuildID[sha1]=1d0b5cf932389000d960a7e6b57c428b8e46c9cf, not stripped
$
```

Load the LiME kernel module

Now it's time to load the kernel module to acquire the system memory. The `insmod` command helps load the kernel module; once loaded, the module reads the primary memory (RAM) on your system and dumps the memory's contents to the file provided in the `path` directory on the command line. Another important parameter is `format`; keep the format `lime`, as shown below. After inserting the kernel module, verify that it loaded using the `lsmod` command:

```
$ lsmod  | grep lime
$
$ insmod ./lime-4.18.0-240.el8.x86_64.ko "path=../RHEL8.3_64bit.mem format=lime"
$
$ lsmod  | grep lime
lime                16384  0
$
```

You should see that the file given to the `path` command was created, and the file size is (not surprisingly) the same as the physical memory size (RAM) on your system. Once you have the memory dump, you can remove the kernel module using the `rmmod` command:

```
$
$ ls -l ~/LiME/RHEL8.3_64bit.mem
-r--r--r--. 1 root root 4294544480 Apr 17 14:47 /root/LiME/RHEL8.3_64bit.mem
$
$ du -sh ~/LiME/RHEL8.3_64bit.mem
4.0G    /root/LiME/RHEL8.3_64bit.mem
$
$ free -m
              total        used        free      shared  buff/cache   available
```

```
Mem:        3736        220        366          8       3149       3259
Swap:       4059          8       4051
$
$ rmmod lime
$
$ lsmod  | grep lime
$
```

What's in the memory dump?

This dump file is just raw data, as you can see using the file command below. You cannot make much sense out of it manually; yes, there are some ASCII strings in there somewhere, but you can't open the file in an editor and read it out. The hexdump output shows that the initial few bytes are EmiL; this is because your request format was "lime" in the command above:

```
$ file ~/LiME/RHEL8.3_64bit.mem
/root/LiME/RHEL8.3_64bit.mem: data
$


$ hexdump -C ~/LiME/RHEL8.3_64bit.mem | head
00000000  45 4d 69 4c 01 00 00 00  00 10 00 00 00 00 00 00  |EMiL............|
00000010  ff fb 09 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  b8 fe 4c cd 21 44 00 32  20 00 00 2a 2a 2a 2a 2a  |..L.!D.2 ..*****|
00000030  2a 2a 2a 2a 2a 2a 2a 2a  2a 2a 2a 2a 2a 2a 2a 2a  |****************|
00000040  2a 2a 2a 2a 2a 2a 2a 2a  2a 2a 2a 2a 2a 20 00 20  |************* . |
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000080  00 00 00 00 00 00 00 00  00 00 00 00 70 78 65 6c  |............pxel|
00000090  69 6e 75 78 2e 30 00 00  00 00 00 00 00 00 00 00  |inux.0..........|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
$
```

Part 2: Get Volatility and use it to analyze your memory dump

Now that you have a sample memory dump to analyze, get the Volatility software with the command below. Volatility has been rewritten in Python 3, but this tutorial uses the original Volatility package, which uses Python 2. If you want to experiment with Volatility 3, download it from the appropriate Git repo and use Python 3 instead of Python 2 in the following commands:

```
$ git clone https://github.com/volatilityfoundation/volatility.git
$
$ cd volatility/
$
$ ls
AUTHORS.txt    contrib      LEGAL.txt    Makefile     PKG-INFO     pyinstaller.spec  resources  tools       vol.py
CHANGELOG.txt  CREDITS.txt  LICENSE.txt  MANIFEST.in  pyinstaller  README.txt        setup.py   volatility
$
```

Volatility uses two Python libraries for some functionality, so please install them using the following commands. Otherwise, you might see some import errors when you run the Volatility tool; you can ignore them unless you are running a plugin that needs these libraries; in that case, the tool will error out:

```
$ pip2 install pycrypto
$ pip2 install distorm3
```

List Volatility's Linux profiles

The first Volatility command you'll want to run lists what Linux profiles are available. The main entry point to running any Volatility commands is the `vol.py` script. Invoke it using the Python 2 interpreter and provide the `--info` option. To narrow down the output, look for strings that begin with Linux. As you can see, not many Linux profiles are listed:

```
$ python2 vol.py --info  | grep ^Linux
Volatility Foundation Volatility Framework 2.6.1
LinuxAMD64PagedMemory          - Linux-specific AMD 64-bit address space.
$
```

Build your own Linux profile

Linux distros are varied and built for various architectures. This why profiles are essential—Volatility must know the system and architecture that the memory dump was acquired from before extracting information. There are Volatility commands to find this information; however, this method is time-consuming. To speed things up, build a custom Linux profile using the following commands.

Move to the `tools/linux` directory within the Volatility repo, and run the `make` command:

```
$ cd tools/linux/
$
$ pwd
/root/volatility/tools/linux
$
$ ls
kcore  Makefile  Makefile.enterprise  module.c
$
$ make
make -C //lib/modules/4.18.0-240.el8.x86_64/build CONFIG_DEBUG_INFO=y M="/root/volatility/tools/linux"
modules
make[1]: Entering directory '/usr/src/kernels/4.18.0-240.el8.x86_64'
<< snip >>
make[1]: Leaving directory '/usr/src/kernels/4.18.0-240.el8.x86_64'
$
```

You should see a new `module.dwarf` file. You also need the `System.map` file from the `/boot` directory, as it contains all of the symbols related to the currently running kernel:

```
$ ls
kcore  Makefile  Makefile.enterprise  module.c  module.dwarf
$
$ ls -l module.dwarf
-rw-r--r--. 1 root root 3987904 Apr 17 15:17 module.dwarf
$
$ ls -l /boot/System.map-4.18.0-240.el8.x86_64
-rw-------. 1 root root 4032815 Sep 23  2020 /boot/System.map-4.18.0-240.el8.x86_64
$
$
```

To create a custom profile, move back to the Volatility directory and run the command below. The first argument provides a custom .zip with a file name of your choice. I used the operating system and kernel versions in the name. The next argument is the module.dwarf file created above, and the final argument is the System.map file from the /boot directory:

```
$
$ cd volatility/
$
$ zip volatility/plugins/overlays/linux/Redhat8.3_4.18.0-240.zip tools/linux/module.dwarf /boot/System.map-4.18.0-240.el8.x86_64
  adding: tools/linux/module.dwarf (deflated 91%)
  adding: boot/System.map-4.18.0-240.el8.x86_64 (deflated 79%)
$
```

Your custom profile is now ready, so verify the .zip file was created at the location given above. If you want to know if Volatility detects this custom profile, run the --info command again. This time, you should see the new profile listed below:

```
$
$ ls -l volatility/plugins/overlays/linux/Redhat8.3_4.18.0-240.zip
-rw-r--r--. 1 root root 1190360 Apr 17 15:20 volatility/plugins/overlays/linux/Redhat8.3_4.18.0-240.zip
$
$
$ python2 vol.py --info  | grep Redhat
Volatility Foundation Volatility Framework 2.6.1
LinuxRedhat8_3_4_18_0-240x64 - A Profile for Linux Redhat8.3_4.18.0-240 x64
$
$
```

Start using Volatility

Now you are all set to do some actual memory forensics. Remember, Volatility is made up of custom plugins that you can run against a memory dump to get information. The command's general format is:

```
python2 vol.py -f <memory-dump-file-taken-by-Lime> <plugin-name> --profile=<name-of-our-custom-profile>
```

Armed with this information, run the linux_banner plugin to see if you can identify the correct distro information from the memory dump:

```
$ python2 vol.py -f ~/LiME/RHEL8.3_64bit.mem linux_banner --profile=LinuxRedhat8_3_4_18_0-240x64
Volatility Foundation Volatility Framework 2.6.1
```

```
Linux version 4.18.0-240.el8.x86_64 (mockbuild@vm09.test.com) (gcc version 8.3.1 20191121 (Red Hat 8.3.1-5))
(GCC)) #1 SMP Wed Sep 23 05:13:10 EDT 2020
$
```

Find Linux plugins

That worked well, so now you're probably curious about how to find all the names of all the Linux plugins. There is an easy trick: run the `--info` command and `grep` for the `linux_` string. There are a variety of plugins available for different uses. Here is a partial list:

```
$ python2 vol.py --info  | grep linux_
Volatility Foundation Volatility Framework 2.6.1
linux_apihooks             - Checks for userland apihooks
linux_arp                  - Print the ARP table
linux_aslr_shift           - Automatically detect the Linux ASLR shift

<< snip >>

linux_banner               - Prints the Linux banner information
linux_vma_cache            - Gather VMAs from the vm_area_struct cache
linux_volshell             - Shell in the memory image
linux_yarascan             - A shell in the Linux memory image
$
```

Check which processes were running on the system when you took the memory dump using the linux_psaux plugin. Notice the last command in the list: it's the `insmod` command you ran before the dump:

```
$ python2 vol.py -f ~/LiME/RHEL8.3_64bit.mem linux_psaux --profile=LinuxRedhat8_3_4_18_0-240x64
Volatility Foundation Volatility Framework 2.6.1
Pid    Uid    Gid    Arguments
1      0      0      /usr/lib/systemd/systemd --switched-root --system --deserialize 18
2      0      0      [kthreadd]
3      0      0      [rcu_gp]
4      0      0      [rcu_par_gp]
861    0      0      /usr/libexec/platform-python -Es /usr/sbin/tuned -l -P
869    0      0      /usr/bin/rhsmcertd
```

```
875    0      0      /usr/libexec/sssd/sssd_be --domain implicit_files --uid 0 --gid 0 --logger=files
878    0      0      /usr/libexec/sssd/sssd_nss --uid 0 --gid 0 --logger=files

<<< snip >>>

11064  89     89     qmgr -l -t unix -u
227148 0      0      [kworker/0:0]
227298 0      0      -bash
227374 0      0      [kworker/u2:1]
227375 0      0      [kworker/0:2]
227884 0      0      [kworker/0:3]
228573 0      0      insmod ./lime-4.18.0-240.el8.x86_64.ko path=../RHEL8.3_64bit.mem format=lime
228576 0      0
$
```

Want to know about the system's network stats? Run the linux_netstat plugin to find the state of the network connections during the memory dump:

```
$ python2 vol.py -f ~/LiME/RHEL8.3_64bit.mem linux_netstat --profile=LinuxRedhat8_3_4_18_0-240x64
Volatility Foundation Volatility Framework 2.6.1
UNIX 18113           systemd/1     /run/systemd/private
UNIX 11411           systemd/1     /run/systemd/notify
UNIX 11413           systemd/1     /run/systemd/cgroups-agent
UNIX 11415           systemd/1
UNIX 11416           systemd/1
<< snip>>
$
```

Next, use the linux_mount plugin to see which filesystems were mounted during the memory dump:

```
$ python2 vol.py -f ~/LiME/RHEL8.3_64bit.mem linux_mount --profile=LinuxRedhat8_3_4_18_0-240x64
Volatility Foundation Volatility Framework 2.6.1
tmpfs                 /sys/fs/cgroup                tmpfs       ro,nosuid,nodev,noexec
cgroup                /sys/fs/cgroup/pids           cgroup      rw,relatime,nosuid,nodev,noexec
systemd-1             /proc/sys/fs/binfmt_misc      autofs      rw,relatime
```

```
sunrpc             /var/lib/nfs/rpc_pipefs          rpc_pipefs   rw,relatime
/dev/mapper/rhel_kvm--03--guest11-root /                          xfs        rw,relatime
tmpfs              /dev/shm                  tmpfs       rw,nosuid,nodev
selinuxfs            /sys/fs/selinux               selinuxfs    rw,relatime
<< snip>>

cgroup             /sys/fs/cgroup/net_cls,net_prio    cgroup       rw,relatime,nosuid,nodev,noexec
cgroup             /sys/fs/cgroup/cpu,cpuacct        cgroup       rw,relatime,nosuid,nodev,noexec
bpf               /sys/fs/bpf                 bpf        rw,relatime,nosuid,nodev,noexec
cgroup             /sys/fs/cgroup/memory            cgroup      ro,relatime,nosuid,nodev,noexec
cgroup             /sys/fs/cgroup/cpuset            cgroup      rw,relatime,nosuid,nodev,noexec
mqueue              /dev/mqueue                 mqueue      rw,relatime
$
```

Curious what kernel modules were loaded? Volatility has a plugin for that too, aptly named linux_lsmod:

```
$ python2 vol.py -f ~/LiME/RHEL8.3_64bit.mem linux_lsmod --profile=LinuxRedhat8_3_4_18_0-240x64
Volatility Foundation Volatility Framework 2.6.1
ffffffffc0535040 lime 20480
ffffffffc0530540 binfmt_misc 20480
ffffffffc05e8040 sunrpc 479232
<< snip >>
ffffffffc04f9540 nfit 65536
ffffffffc0266280 dm_mirror 28672
ffffffffc025e040 dm_region_hash 20480
ffffffffc0258180 dm_log 20480
ffffffffc024bbc0 dm_mod 151552
$
```

Want to find all the commands the user ran that were stored in the Bash history? Run the linux_bash plugin:

```
$ python2 vol.py -f ~/LiME/RHEL8.3_64bit.mem linux_bash --profile=LinuxRedhat8_3_4_18_0-240x64 -v
Volatility Foundation Volatility Framework 2.6.1
Pid      Name              Command Time              Command
```

```
-------- ------------------- ----------------------------- -------
    227221 bash            2021-04-17 18:38:24 UTC+0000   lsmod
    227221 bash            2021-04-17 18:38:24 UTC+0000   rm -f .log
    227221 bash            2021-04-17 18:38:24 UTC+0000   ls -l /etc/zzz
    227221 bash            2021-04-17 18:38:24 UTC+0000   cat ~/.vimrc
    227221 bash            2021-04-17 18:38:24 UTC+0000   ls
    227221 bash            2021-04-17 18:38:24 UTC+0000   cat /proc/817/cwd
    227221 bash            2021-04-17 18:38:24 UTC+0000   ls -l /proc/817/cwd
    227221 bash            2021-04-17 18:38:24 UTC+0000   ls /proc/817/
<< snip >>
    227298 bash            2021-04-17 18:40:30 UTC+0000   gcc prt.c
    227298 bash            2021-04-17 18:40:30 UTC+0000   ls
    227298 bash            2021-04-17 18:40:30 UTC+0000   ./a.out
    227298 bash            2021-04-17 18:40:30 UTC+0000   vim prt.c
    227298 bash            2021-04-17 18:40:30 UTC+0000   gcc prt.c
    227298 bash            2021-04-17 18:40:30 UTC+0000   ./a.out
    227298 bash            2021-04-17 18:40:30 UTC+0000   ls
$
```

Want to know what files were opened by which processes? Use the linux_lsof plugin to list that information:

```
$ python2 vol.py -f ~/LiME/RHEL8.3_64bit.mem linux_lsof --profile=LinuxRedhat8_3_4_18_0-240x64
Volatility Foundation Volatility Framework 2.6.1
Offset             Name                           Pid      FD       Path
------------------ ------------------------------ -------- -------- ----
0xffff9c83fb1e9f40 rsyslogd                       71194        0 /dev/null
0xffff9c83fb1e9f40 rsyslogd                       71194        1 /dev/null
0xffff9c83fb1e9f40 rsyslogd                       71194        2 /dev/null
0xffff9c83fb1e9f40 rsyslogd                       71194        3 /dev/urandom
0xffff9c83fb1e9f40 rsyslogd                       71194        4 socket:[83565]
0xffff9c83fb1e9f40 rsyslogd                       71194        5 /var/log/messages
0xffff9c83fb1e9f40 rsyslogd                       71194        6 anon_inode:[9063]
0xffff9c83fb1e9f40 rsyslogd                       71194        7 /var/log/secure
```

```
<< snip >>

0xffff9c8365761f40 insmod                    228573       0 /dev/pts/0
0xffff9c8365761f40 insmod                    228573       1 /dev/pts/0
0xffff9c8365761f40 insmod                    228573       2 /dev/pts/0
0xffff9c8365761f40 insmod                    228573       3 /root/LiME/src/lime-4.18.0-240.el8.x86_64.ko
$
```

Access the Linux plugins scripts location

You can get a lot more information by reading the memory dump and processing the information. If you know Python and are curious how this information was processed, go to the directory where all the plugins are stored, pick one that interests you, and see how Volatility gets this information:

```
$ ls volatility/plugins/linux/
apihooks.py          common.py          kernel_opened_files.py   malfind.py         psaux.py
apihooks.pyc         common.pyc         kernel_opened_files.pyc  malfind.pyc        psaux.pyc
arp.py               cpuinfo.py         keyboard_notifiers.py    mount_cache.py     psenv.py
arp.pyc              cpuinfo.pyc        keyboard_notifiers.pyc   mount_cache.pyc    psenv.pyc
aslr_shift.py        dentry_cache.py    ld_env.py                mount.py           pslist_cache.py
aslr_shift.pyc       dentry_cache.pyc   ld_env.pyc               mount.pyc          pslist_cache.pyc
<< snip >>
check_syscall_arm.py    __init__.py     lsmod.py                 proc_maps.py       tty_check.py
check_syscall_arm.pyc   __init__.pyc    lsmod.pyc                proc_maps.pyc      tty_check.pyc
check_syscall.py        iomem.py        lsof.py                  proc_maps_rb.py    vma_cache.py
check_syscall.pyc       iomem.pyc       lsof.pyc                 proc_maps_rb.pyc   vma_cache.pyc
$
$
```

One reason I like Volatility is that it provides a lot of security plugins. This information would be difficult to acquire manually:

```
linux_hidden_modules       - Carves memory to find hidden kernel modules
linux_malfind              - Looks for suspicious process mappings
linux_truecrypt_passphrase - Recovers cached Truecrypt passphrases
```

Volatility also allows you to open a shell within the memory dump, so instead of running all the commands above, you can run shell commands instead and get the same information:

```
$ python2 vol.py -f ~/LiME/RHEL8.3_64bit.mem linux_volshell --profile=LinuxRedhat8_3_4_18_0-240x64 -v
Volatility Foundation Volatility Framework 2.6.1
Current context: process systemd, pid=1 DTB=0x1042dc000
Welcome to volshell! Current memory image is:
file:///root/LiME/RHEL8.3_64bit.mem
To get help, type 'hh()'
>>>
>>> sc()
Current context: process systemd, pid=1 DTB=0x1042dc000
>>>
```

Next steps

Memory forensics is a good way to learn more about Linux internals. Try all of Volatility's plugins and study their output in detail. Then think about ways this information can help you identify an intrusion or a security issue. Dive into how the plugins work, and maybe even try to improve them. And if you didn't find a plugin for what you want to do, write one and submit it to Volatility so others can use it, too.

Flag: DCTF{pdfs_are_easy_to_crack}