Acquiring Forensic Memory Images on Linux

2.1 Linux Memory Dump Toolsn

We use the Volatility Framework when performing memory analysis for incident response and forensics. In IIR Vol. 32, "1.4.1 Creating Profiles for the Volatility Framework," we explained the procedure for creating a Volatility profile for Linux[*1].

Here, we explain how to acquire Linux memory images for analysis in Volatility. Several tools for acquiring Linux memory images exist, but here we will look at LiME[*2] and crash[*3]. We will also describe a memory image acquisition method that minimizes the impact on disk forensics. Note that this article assumes your system is running CentOS 7.7-1908. Another tool for acquiring memory images is Linpmem[*4], but it did not work well in our test environment, so we decided to leave it out.

Note that in this article we refer to the machine being examined as the "target host" and the machine on which the examination is performed as the "examination host". To avoid changing data on the target host as much as possible, compilation of the analysis tools, etc., should be done on a separate host.

2.2 What is LiME?

LiME is short for Linux Memory Extractor, and is the tool that Volatility recommends[*5] for acquiring memory images. Since LiME operates as a Linux kernel module, it must be compiled against the same kernel version as that running on the target host.

2.3 Compiling LiME

To compile LiME, you first obtain the LiME source code by running the git command in Figure 1 or by downloading the zip file from LiME's GitHub page and extracting it into a directory. The src subdirectory inside the source code directory contains a Makefile, so running the make command from within that subdirectory will generate the LiME module

```
$ git clone https://github.com/504ensicsLabs/LiME.git
```

Figure 1: Cloning the LiME Git Repository

```
$ cd LiME/src
$ make
$ ls
disk.c  lime-3.10.0-1062.el7.x86_64.ko  lime.o     Makefile.sample  tcp.o
disk.o  lime.h                          main.c     modules.order
hash.c  lime.mod.c                      main.o     Module.symvers
hash.o  lime.mod.o                      Makefile   tcp.c
```

Figure 2: Compiling the LiME Module

```
$ sudo yum install kernel-devel gcc
```

Figure 3: Installing the kernel-devel Package

*1 Internet Infrastructure Review (IIR) Vol. 32, 1.4.1 Creating Profiles for the Volatility Framework (https://www.iij.ad.jp/en/dev/iir/032.html).
*2 LiME (https://github.com/504ensicsLabs/LiME).
*3 crash (https://people.redhat.com/anderson/).
*4 Velocidex/c-aff4 (https://github.com/Velocidex/c-aff4/releases).
*5 Linux · volatilityfoundation/volatility Wiki (https://github.com/volatilityfoundation/volatility/wiki/Linux#acquiring-memory).

(see the red outline in Figure 2). Certain packages are required to compile LiME (kernel-devel, gcc, etc.), so if the make command produces an error, try installing the corresponding packages as shown in Figure 3.

Note that unless you specify a version at install time, the latest version of the kernel-devel package will be installed. If a different version of the kernel is running on the target host, run a search using the yum command, install the appropriate kernel-devel package version, and then compile the LiME module, as shown in Figure 4. You need to pass the KVER option to the make command when doing this.

If the OS version differs, you can generate a LiME module for that version by downloading the corresponding kernel-devel package separately, extracting it using the cpio command, and then running make with the KVER and KDIR options specified, as shown in Figure 5. KVER specifies the kernel version, and KDIR specifies the directory where the kernel-devel package was extracted. You also need to install any other required packages (on our machine, we needed to install elfutils-libelf-devel). We confirmed that the LiME module generated in this manner works on CentOS 8 (1905).

2.4 Dumping Memory to an External Drive

When acquiring memory images, you need to minimize disk write operations on the target host. In particular, the memory image is almost always several GB or more is size, so writing it to the target host's disk would cause many unused areas to be overwritten, which could greatly affect forensic analysis of the disk.

So if you have physical access to the target host, you can connect a USB stick or mobile SSD containing the LiME module to it and load the LiME module into the kernel from there via the insmod command, as shown in Figure 6. This will let you acquire a memory image while barely writing anything to the target host's disk. The double quotes in Figure

```
$ yum --showduplicates search kernel-devel
$ sudo yum install kernel-devel-3.10.0-1062.1.2.el7.x86_64
$ make KVER=3.10.0-1062.1.2.el7.x86_64
```

Figure 4: Compiling LiME Against a Specific Kernel Version (1)

```
$ curl -OL http://ftp.iij.ad.jp/pub/linux/centos/8.0.1905/BaseOS/x86_64/os/Packages/kernel-devel-4.18.0-80.11.2.el8_0.x86_64.rpm
$ rpm2cpio ./kernel-devel-4.18.0-80.11.2.el8_0.x86_64.rpm | cpio -id
$ sudo yum install elfutils-libelf-devel
$ make KVER=4.18.0-80.11.2.el8_0.x86_64 KDIR=~/src/usr/src/kernels/4.18.0-80.11.2.el8_0.x86_64/
```

Figure 5: Compiling LiME Against a Specific Kernel Version (2)

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=/media/centos77.mem format=lime"
```

Figure 6: Loading the LiME Module

6 contain the LiME module options. In this example, a lime format dump file will be saved to the USB drive mounted at /media under the filename /media/centos77.mem.

The memory dump starts when the LiME module is loaded, but the command prompt does not return until the memory dump is completed. So don't worry if you're unable to enter commands after loading the module; just wait for the dump to complete. More recent servers, in particular, can be expected to take a while because of their large memory capacity. Note that the LiME module remains loaded even after the memory dump finishes, so unload it using the rmmod command, as shown in Figure 7.

2.5 Dumping Memory via the Network

If you do not have physical access to the target host as covered above (for example, if the target host is in a remote location that makes physical access difficult), or if you don't have a large-capacity USB stick, you can combine LiME with other tools to acquire a memory image via the network. Here, we show how to combine it with Netcat, NFS, and SSH. The examination host's IP address is 192.168.232.131, and target host's is 192.168.232.132.

■ Netcat (1)

The LiME module has functionality that enables memory dumps over the network, and we combine this functionality with the Netcat command to acquire a memory image via the network. Once the Netcat command is installed on the examination host, the commands in Figures 8 and 9 will connect the examination host to the port on which the LiME module is listening (4444/tcp) to acquire the memory image data.

■ Netcat (2)

The above procedure will result in data equivalent to the capacity of memory being received. If the target host is a server, that data may take up several dozen GB or more, so

```
$ lsmod | grep lime
$ sudo rmmod lime
```

Figure 7: Unloading the LiME Module

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
```

Figure 8: Listening on 4444/tcp with LiME (Target Host)

```
$ nc 192.168.232.132 4444 > centos77.mem
```

Figure 9: Acquiring a Memory Image via the Network Using Netcat (Examination Host)

```
$ nc -l 5555 > memorydump.lime.gz
```

Figure 10: Command Executed on the Examination Host

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
Switch to another login session to execute the following command
$ nc localhost 4444 | gzip -c | nc 192.168.232.131 5555
```

Figure 11: Commands Executed on the Target Host

```
$ sudo yum install nfs-utils
$ sudo mkdir /mnt/nfsserv/
$ chown -R nfsnobody:nfsnobody /mnt/nfsserv/
$ sudo vi /etc/exports
$ sudo systemctl start nfs.service
$ sudo systemctl status nfs.service
```

Figure 12: Setting the NFS Server (Examination Host)

```
/mnt/nfsserv/ 192.168.232.132(rw,all_squash)
```

Figure 13: NFS Export Settings on the Examination Host (/etc/exports)

we want to compress it as much as possible. If Netcat is also installed on the target host, executing the commands in Figures 10 and 11 will result in the memory image being compressed using gzip when it is transferred to the examination host. As mentioned, the command prompt does not return when the LiME module is loaded, so the command following the nc command needs to be run from a separate login session.

In this example, we first use Netcat on the examination host to listen on 5555/tcp. Next, we use Netcat on the target host to connect to 4444/tcp, where the LiME module is listening, acquire a memory image, compress it using gzip, and then transfer it to the examination host via Netcat.

■ NFS

If the target host can mount an NFS volume, set up the examination host on a network that the target host has access to. Then export the NFS volume on the examination host with read/write capability. By copying LiME to this NFS volume and mounting it as an NFS from the target host, you can acquire a memory image without creating any files on the target host (Figures 12, 13, 14, and 15).

■ SSH

If you cannot use Netcat or NFS, you can use SSH instead. By executing the commands in Figure 16, you can transfer a memory image to the examination host via SSH. However, this method can only be used in bash. As with the Netcat (2) instructions, the exec command onward must be run from a separate login session.

2.6 What is Crash?

The crash command is a tool for analyzing Linux memory images. The primary purpose of the tool is to perform analysis, but a module can also be used to perform memory dumps. But as there is no RPM package of the crash memory dump module, we have to compile it from source

```
$ sudo firewall-cmd --permanent --add-service=nfs
$ sudo firewall-cmd --reload
$ sudo exportfs -v
```

Figure 14: Configuring the Examination Host's Firewall and Checking the NFS Export

```
$ sudo mount -t nfs 192.168.232.131:/mnt/nfsserv/ /mnt/
$ sudo insmod /mnt/lime-3.10.0-1062.el7.x86_64.ko "path=/mnt/centos77.mem format=lime"
```

Figure 15: Mounting the NFS and Acquiring a Memory Image (Target Host)

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
Switch to another login session to execute the following command
$ exec 5<>/dev/tcp/127.0.0.1/4444; cat <&5 | ssh -c user@192.168.232.131 'cat > centos77.mem'
```

Figure 16: Commands Executed on the Target Host

(Figure 17). The crash command also requires a kernel with debug symbols, so we install the kernel debugging package (Figure 18). Next, we copy the three files shown in Figure 19 to the same directory on a USB stick. On the target host, we execute crash as shown in Figure 20 to acquire a memory image.

If the kernel version of the target and examination hosts differ, then search for the appropriate version of the kernel-debuginfo package as demonstrated in Figure 4 (yum --showduplicates search). Download and extract the package as shown in FIgure 5 (curl, rpm2cpio, cpio commands), and copy the vmlinux file. We also recommend using a crash version that corresponds to your OS version (for example, crash-7.2.3-18 is provided in CentOS 8.0).

2.7 Analyzing the Memory Image

To analyze a Linux memory image using Volatility[*5], you need a profile corresponding to the Linux kernel version. As noted at the beginning, the procedure for creating a Volatility profile is described in IIR Vol. 32[*1]. Please refer there if you are unsure.

While we were writing this article, Lorenzo Martínez opened a Bitbucket repository that automatically generates and publishes LiME modules and Linux profiles for Volatility[*7]. This repository is updated whenever new Linux kernel packages are released. Note, though, that the OSs that it automatically generates for are CentOS 5, 6, 7, 8 and Ubuntu 14.04 LTS, 16.04 LTS, 18.04 LTS. You can find and download the appropriate LiME module and

```
$ sudo yum install crash crash-devel
$ yumdownloader --source crash
$ rpm -ivh crash-7.2.3-10.el7.src.rpm
$ cd rpmbuild/SPECS
$ rpmbuild -bp crash.spec
$ cd ../BUILD/crash-7.2.3
$ make extensions
```

Figure 17: Installing the crash Source Package and Compiling the Module

```
・/usr/bin/crash
・rpmbuild/BUILD/crash-7.2.3/extensions/snap.so
・/usr/lib/debug/usr/lib/modules/3.10.0-1062.el7.x86_64/vmlinux
```

Figure 19: Files Required for a Memory Dump

```
$ sudo yum install --enablerepo=base-debuginfo kernel-debuginfo-3.10.0-1062.el7.x86_64
```

Figure 18: Installing a Kernel with Debug Symbols

*6 The Volatility Foundation - Open Source Memory Forensics (https://www.volatilityfoundation.org/).

*7 Lorenzo Martínez' tweet (https://twitter.com/lawwait/status/1181469996821700609).

Volatility profile by filtering the repository webpage by kernel version.

Also, it does not support architectures other than x86_64, so if you are using a Linux distribution or architecture other than those covered by the autogenerated files, you will need to prepare the files yourself. The same applies if you are using a customized Linux kernel. If using crash, you will also need your own customized kernel with debug symbols.

2.8 Tips for Acquiring Memory Images

In Linux kernel version 2.4 upward, the tmpfs file system is available. Data on a tmpfs file system resides only in memory and is lost if the host is shut down or restarted, so tmpfs is usually only used for temporary directories and the like with files one is not concerned about losing. It has been observed, however, that attackers take advantage of these properties by using tmpfs as an anti-disk forensics haven for files.

For this reason, Volatility provides the Linux-specific linux_tmpfs command. This command is used to restore files held in a tmpfs. Figure 21 illustrates tmpfs files mounted on the /home/user/tmp directory being restored. The command results in the file called “tmpfs_example.txt” being restored, and evidently it contains the string “hello!!”

If available memory is running low, however, the contents of the tmpfs are swapped out, making it impossible to restore the data in the tmpfs via a memory dump (Figure 22). A potential countermeasure in this case is to temporarily

```
Move to the directory where the files were copied and then execute these
commands.

$ sudo ./crash ./vmlinux
(Then from within the crash command prompt)
extend ./snap.so
snap centos77.mem
```

Figure 20: Acquiring a Memory Image

```
$ hexdump -C ~/vol_output/swapout/tmpfs_example.txt
00000000  00 00 00 00 00 00 00 00                           |........|
00000008
```

Figure 22: Example of a Failure to Restore tmpfs Data due to a Swap-Out

```
$ python2 ./vol.py --profile=LinuxCentOS77x64 -f ~/tmpfs_swapoff.mem linux_tmpfs -L
Volatility Foundation Volatility Framework 2.6.1
1 -> /sys/fs/cgroup
2 -> /run
3 -> /home/user/tmp
4 -> /dev/shm

$ python2 ./vol.py --profile=LinuxCentOS77x64 -f ~/tmpfs_swapoff.mem linux_tmpfs -S 3 -D ~/vol_output/
$ hexdump -C ~/vol_output/tmpfs_example.txt
00000000  68 65 6c 6c 6f 21 21 0a                           |hello!!.|
00000008
```

Figure 21: Restoring and Examining the Contents of Files Stored in a tmpfs

disable the swap. This means forcing data that has been swapped out to be swapped in. Whether or not this can be done, however, depends on memory usage on the target host. If data has been swapped out because of a temporary increase in memory usage, you may be able to subsequently disable the swap if memory has since been freed up. This means a situation like that illustrated in Figure 23, for example, where the value of Swap used is lower than free Mem. Analysis of the memory image acquired after disabling the swap confirms that the tmpfs data can be restored, as shown in Figure 24.

This method, however, does overwrite data in unused areas of memory. Because unused memory areas may still contain useful data, a better way to perform Linux memory forensics is to dump memory once before disabling the swap and once again after.

2.9 Volatility 3

Version 2 of Volatility has been in use for a long time, but during the writing of this article, a public beta version of Volatility 3 was released[*8]. Of course we did not check it with all types of memory images, but in our setup, we were able to analyze Windows RAW memory images and CentOS memory images acquired using LiME. A big change is that the option to specify a profile, which was required in Volatility 2, is no longer present. Instead, Volatility 3 infers OS type and version from the memory image being analyzed and refers to the corresponding symbol table. For example, when a Windows memory image is read in, Volatility 3 automatically downloads the PDB file from Microsoft and analyzes it to construct a symbol table that it can reference. However, you still need to prepare symbol tables for macOS and Linux in advance, as in Volatility 2. You can use the symbol tables provided by the Volatility developers, but the symbol tables for Linux lack information relative to those for Windows and macOS, so in most cases, you'll need to provide the table yourself.

Symbol tables are created using a tool called dwarf2json[*9]. As dwarf2json is written in Go, we first install the golang package and then build dwarf2json. We also install the kernel-debuginfo package because we need a Linux kernel with symbols. Executing dwarf2json with a Linux kernel with

```
$ free
              total        used        free      shared  buff/cache   available
Mem:        1863248       75920      307192         768     1480136     1591860
Swap:       2097148       56776     2040372
$ sudo swapoff -a
$ free
              total        used        free      shared  buff/cache   available
Mem:        1863248      118804      247652        9800     1496792     1539936
Swap:             0           0           0
(After dumping memory, reenable the swap)
$ sudo swapon -a
```

Figure 23: Checking Memory Usage, and Disabling and Enabling the Swap

```
$ hexdump -C ~/vol_output/swapoff/tmpfs_example.txt
00000000  68 65 6c 6c 6f 21 21 0a                           |hello!!.|
00000008
```

Figure 24: tmpfs Data Restored from a Memory Image After Disabling the Swap

```
$ sudo yum install epel-release
$ sudo yum install golang
$ git clone https://github.com/volatilityfoundation/dwarf2json.git
$ cd dwarf2json
$ go build
$ sudo yum install --enablerepo=base-debuginfo kernel-debuginfo-3.10.0-1062.1.2.el7.x86_64
$ ./dwarf2json linux --elf /usr/lib/debug/usr/lib/modules/3.10.0-1062.1.2.el7.x86_64/vmlinux > centos77-3.10.0-1062.1.2.el7.x86_64.json
$ xz -z centos77-3.10.0-1062.1.2.el7.x86_64.json
$ wget https://downloads.volatilityfoundation.org/volatility3/symbols/linux.zip
$ zip ./linux.zip ./centos77-3.10.0-1062.1.2.el7.x86_64.json.xz
```

Figure 25: Building dwarf2json and Generating a Symbol Table

*8 Volatility Labs: Announcing the Volatility 3 Public Beta! (https://volatility-labs.blogspot.com/2019/10/announcing-volatility-3-public-beta.html).

*9 dwarf2json (https://github.com/volatilityfoundation/dwarf2json).

symbols specified will generate a symbol table, so we add this to the file (linux.zip) that contains the symbol table distributed by the developers. See Figure 25 for the specific commands. Copy the generated symbol table to the specified Volatility 3 subdirectory (Figure 26).

Volatility 3 is executed using this command line format: "python3 vol.py -f <memory image file> <plugin>". Figure 27 shows the results of analyzing a CentOS 7.7 memory image using the pstree plugin. The format has changed slightly from that of Volatility 2, with the * character now used to denote process nesting. The method for specifying the plugin to be executed has also changed. For the pstree plugin, you specify "linux_pstree" in Volatility 2, but in Volatility 3, you specify "linux.pstree.PsTree". A list of available plugins can be found by running "python3 vol.py -h".

While we did get it to work properly, we also identified some bugs. As noted above, OS type and version are inferred from the contents of the memory image specified on the command line, but it can fail to recognize the correct Linux kernel version with some memory images, causing the analysis to fail. And with Windows memory images, sometimes analysis of the downloaded PDB fails, preventing you from advancing to the memory image analysis stage of the process.

The Volatility development team has announced an official Volatility 3 version will be released in August 2020. Support for Volatility 2 will continue for one year after that through August 2021, but with Volatility 3 set to become mainstream ahead, it's probably a good idea to get accustomed to the new usage and configuration methods before the official release hits.

```
$ cd ..
$ sudo yum install python3
$ git clone https://github.com/volatilityfoundation/volatility3.git
$ pip3 install --user pefile yara-python capstone
$ cp ./dwarf2json/linux.zip ./volatility3/volatility/symbols/
```

Figure 26: Installing Volatility 3 and Copying the Symbol Tables

```
$ cd volatility3
$ python3 vol.py -f ~/centos77.mem linux.pstree.PsTree
Volatility 3 Framework 1.0.0-beta.1
Progress:   23.13               Scanning LimeLayer using RegExScanner
PID     PPID    COMM

1       0       systemd
* 833   1       login
** 1695 833     bash
* 841   1       firewalld
* 843   1       NetworkManager
** 992  843     dhclient
* 558   1       systemd-journal
* 593   1       systemd-udevd
* 818   1       dbus-daemon
* 1171  1       sshd
** 1720 1171    sshd
*** 1724        1720    sshd
**** 1725       1724    bash
***** 1751      1725    sudo
****** 1753     1751    insmod
* 1172  1       tuned
* 821   1       systemd-logind
* 822   1       irqbalance
* 823   1       polkitd
* 1174  1       rsyslogd
* 1397  1       master
** 1402 1397    pickup
** 1405 1397    qmgr
(Subsequent output omitted)
```

Figure 27: Running the pstree Plugin on a Linux Memory Image

Minoru Kobayashi

Forensic Investigator, Office of Emergency Response and Clearinghouse for Security Information, Advanced Security Division, IIJ
Mr. Kobayashi is a member of IIJ-SECT, mainly dealing with digital forensics. He works to improve incident response capabilities and in-house technical capabilities.
He gives lectures and training sessions at security events both in Japan and abroad, including Black Hat, FIRST TC, JSAC, and Security Camp events.