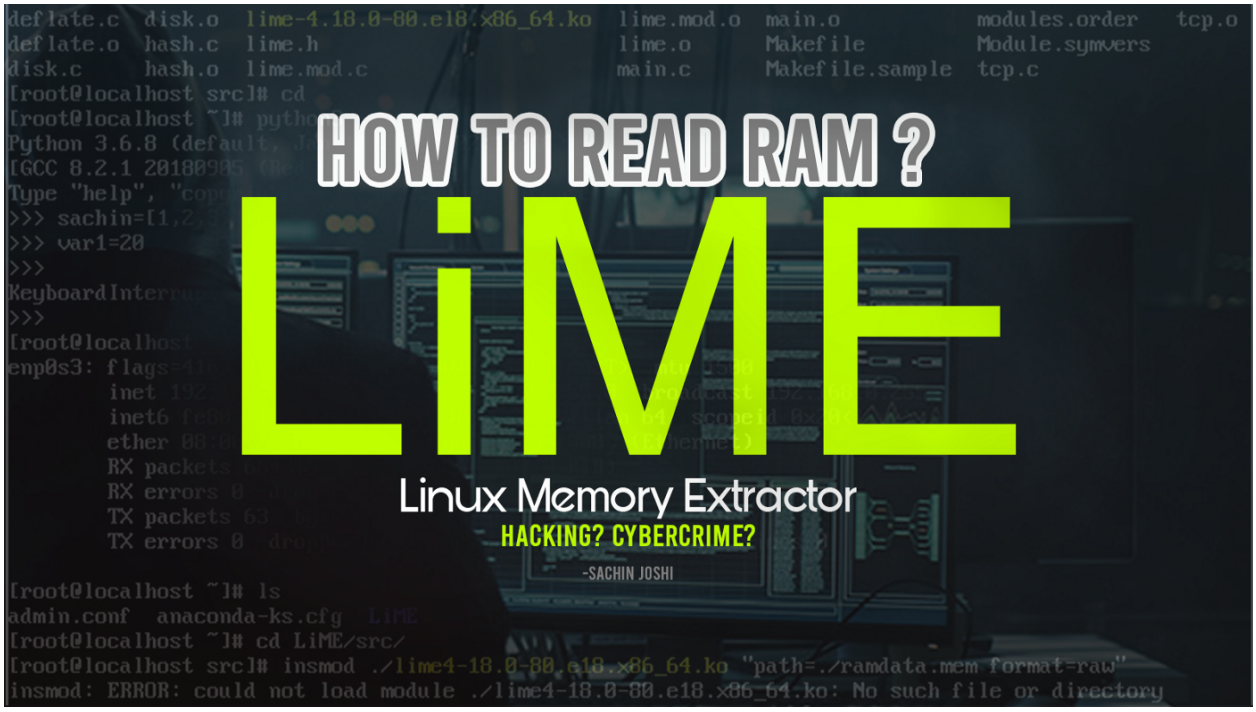
How to read data stored in RAM?(Memory Forensic)



What is RAM and What data RAM contains?

Random-access memory (RAM) is a computer's short-term memory. None of your programs, files, or Netflix streams would work without RAM, which is your computer's working space.

RAM is short for "random access memory" and while it might sound mysterious, RAM is one of the most fundamental elements of computing. RAM is the super-fast and temporary data storage space that a computer needs to access right now or in the next few moments.

Mark my word that ram contains the most valuable data of your Operating System which might or might never be written on Harddisk.

What does RAM contains ?

username passwords

-Recently opened file which has been wiped from disk

-process information

-list of all running processes

-command-line information

-Unencrypted data from an encrypted disk

-keystrokes

-network information

-crypto keys and ton lot of more data.

So it’s basically clear that RAM is one of the most important components in determining your system’s performance. RAM gives applications a place to store and access data on a short-term basis.

Also one of the use cases to read ram data is considered when has hacker done some illegal activity and police need proofs regarding the same, they usually read the read the RAM of hackers machine which actually provides the tree chart.

So then How can one read what data is inside one’s RAM?

There are multiple course of action to read RAM data each has its own use case I will explain one of the methods to read ram data.

The method that I will be using in that we will dump the whole ram data on disk and then we will read ram read data from it. I will show this in Linux-based Operating System.

But in a similar way you can read ram from windows or mac.

Tools Required for dumping ram data on disk:

Linux based O.S

- LiME
- Linux Memory Grabber
- fmem

MAC O.S

- MACMemoryReader
- Goldfish
- OSXPMem

Windows O.S

- FTK Imager
- Winen

There are many tools I just listed few f themto know more these tools click on the link below:

Top memory dump tools for digital forensics

WINDOWS: Process Hacker: This is an open-source process monitoring application that is very useful to run while the…

hackernewsdog.com

Let's get started,

We will use LiMe (Linux Memory Extractor) to dump ram data on the disk.Since we are using linux operating system.

A Loadable Kernel Module (LKM) which allows for volatile memory acquisition from Linux and Linux-based devices, such as Android. This makes LiME unique as it is the first tool that allows for full memory captures on Android devices. It also minimizes its interaction between user and kernel space processes during acquisition, which allows it to produce memory captures that are more forensically sound than those of other tools designed for Linux memory acquisition.

GitHub - 504ensicsLabs/LiME: LiME (formerly DMD) is a Loadable Kernel Module (LKM), which allows…

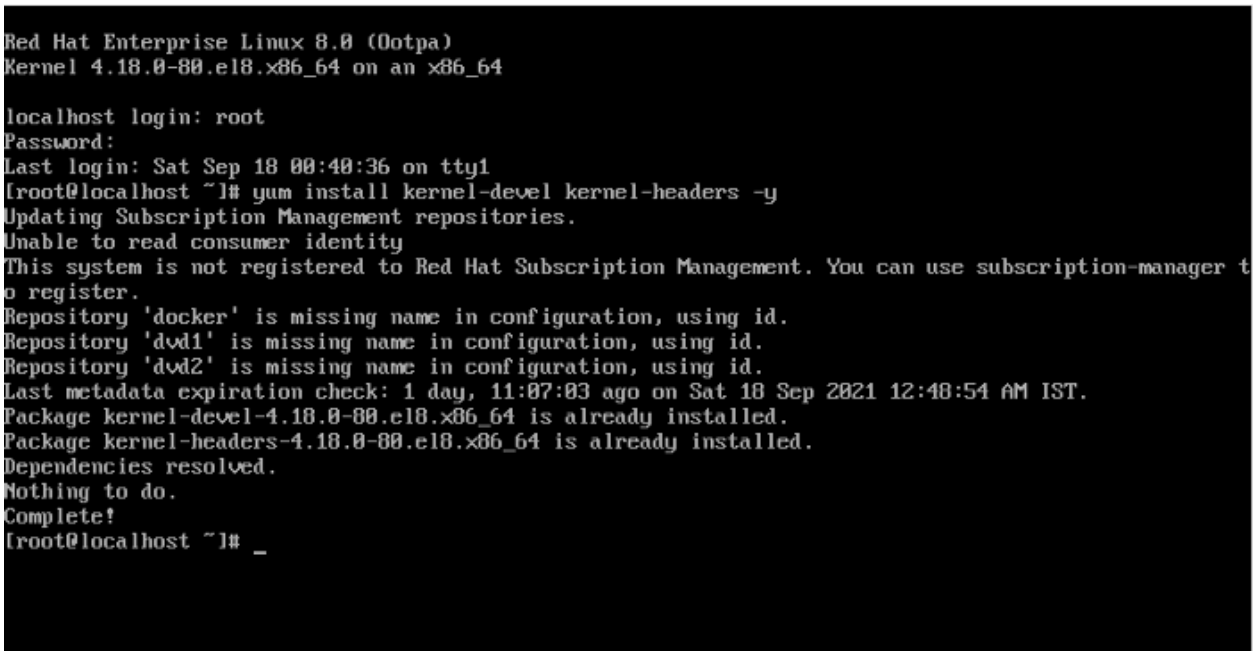
LiME (formerly DMD) is a Loadable Kernel Module (LKM), which allows the acquisition of volatile memory from Linux and…

github.com

We can simply download the source code and compile it to binary files with make. To perform ram acquisition but you can do this on any Linux based O.S.

Also install kernel headers to do ram acquisition.

```
yum install kernel-devel kernel-headers -y
```



I already have the package installed.

Also make sure you install the git package

```
yum install git
```

After installing your screen will look like this,

```
Installed:
  git-2.18.1-3.el8.x86_64                          perl-IO-Socket-IP-0.39-5.el8.noarch
  perl-IO-Socket-SSL-2.060-2.el8.noarch            perl-Mozilla-CA-20160104-7.el8.noarch
  git-core-2.18.1-3.el8.x86_64                     git-core-doc-2.18.1-3.el8.noarch
  perl-Digest-1.17-395.el8.noarch                  perl-Digest-MD5-2.55-396.el8.x86_64
  perl-Error-1:0.17025-2.el8.noarch                perl-Git-2.18.1-3.el8.noarch
  perl-Net-SSLeay-1.85-6.el8.x86_64                perl-TermReadKey-2.37-7.el8.x86_64
  perl-URI-1.73-3.el8.noarch                       perl-libnet-3.11-3.el8.noarch
  emacs-filesystem-1:26.1-5.el8.noarch             perl-Data-Dumper-2.167-399.el8.x86_64
  perl-Encode-4:2.97-3.el8.x86_64                  perl-File-Temp-0.230.600-1.el8.noarch
  perl-Getopt-Long-1:2.50-4.el8.noarch             perl-HTTP-Tiny-0.074-1.el8.noarch
  perl-MIME-Base64-3.15-396.el8.x86_64             perl-Pod-Escapes-1:1.07-395.el8.noarch
  perl-Pod-Perldoc-3.28-396.el8.noarch             perl-Pod-Simple-1:3.35-395.el8.noarch
  perl-Pod-Usage-4:1.69-395.el8.noarch             perl-Storable-1:3.11-3.el8.x86_64
  perl-Term-ANSIColor-4.06-396.el8.noarch          perl-Term-Cap-1.17-395.el8.noarch
  perl-Text-ParseWords-3.30-395.el8.noarch         perl-Time-Local-1:1.280-1.el8.noarch
  perl-podlators-4.11-1.el8.noarch

Complete!
[root@localhost ~]#
```

Now we have to clone the GitHub repo of LiME

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

```
[root@localhost ~]# git clone https://github.com/504ensicsLabs/LiME.git
Cloning into 'LiME'...
remote: Enumerating objects: 362, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 362 (delta 4), reused 10 (delta 4), pack-reused 349
Receiving objects: 100% (362/362), 1.61 MiB | 534.00 KiB/s, done.
Resolving deltas: 100% (193/193), done.
[root@localhost ~]#
```

Now we can compile the source code of LiME… first, we need to navigate to the src directory

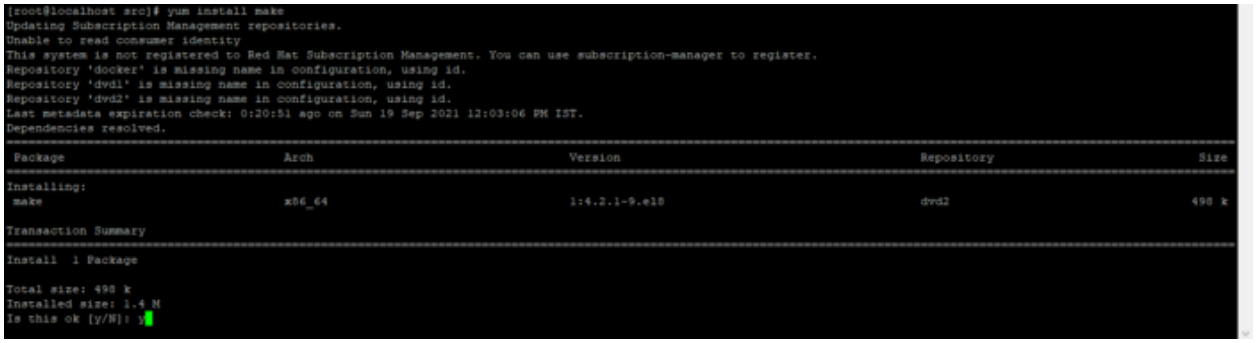
```
cd LiMe/src
```

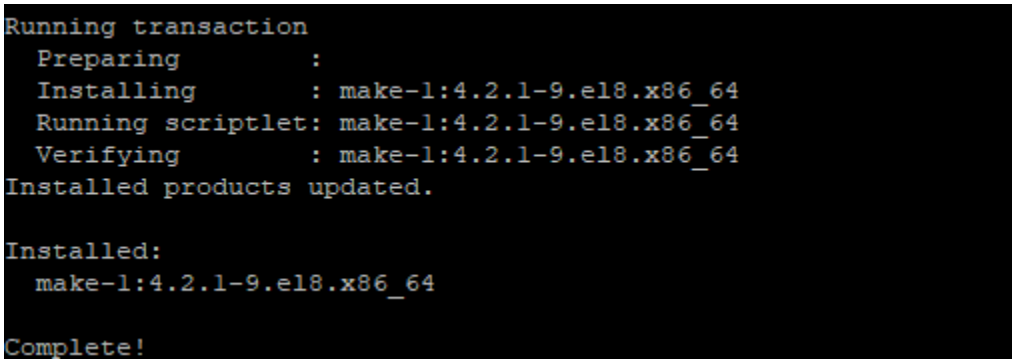
```
[root@localhost ~]# ls
admin.conf  anaconda-ks.cfg  LiME
[root@localhost ~]# cd LiME/src
[root@localhost src]# ls
deflate.c  disk.c  hash.c  lime.h  main.c  Makefile  Makefile.sample  tcp.c
[root@localhost src]#
```

“Make” is typically used to **build executable programs and libraries from source code**. Generally though, Make is applicable to any process that involves executing arbitrary commands to transform a source file to a target result.

Install make first

```
yum install make
```





Now we can simply type the “**make**” command it will compile the source code and give us a loadable kernel object file

```
make
```

```
[root@localhost ~]# ls
admin.conf  anaconda-ks.cfg  LiME
[root@localhost ~]# cd LiME/src
[root@localhost src]# ls
deflate.c  disk.c  hash.c  lime.h  main.c  Makefile  Makefile.sample  tcp.c
[root@localhost src]# make
make -C /lib/modules/4.18.0-80.el8.x86_64/build M="/root/LiME/src" modules
make[1]: Entering directory '/usr/src/kernels/4.18.0-80.el8.x86_64'
arch/x86/Makefile:184: *** Compiler lacks asm-goto support.  Stop.
make[1]: Leaving directory '/usr/src/kernels/4.18.0-80.el8.x86_64'
make: *** [Makefile:35: default] Error 2
[root@localhost src]#
```

if you get this error make sure you install two more package/module

```
yum groupinstall "Development tools"
yum install elfutils-libelf-devel
```

After running this command again hit make keyword

```
make
```

```
[root@localhost src]# make
make -C /lib/modules/4.18.0-80.el8.x86_64/build M="/root/LiME/src" modules
make[1]: Entering directory '/usr/src/kernels/4.18.0-80.el8.x86_64'
  CC [M]  /root/LiME/src/tcp.o
/root/LiME/src/tcp.c: In function 'setup_tcp':
/root/LiME/src/tcp.c:75:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after
-statement]
     int opt = 1;
     ^~~
  CC [M]  /root/LiME/src/disk.o
  CC [M]  /root/LiME/src/main.o
  CC [M]  /root/LiME/src/hash.o
  CC [M]  /root/LiME/src/deflate.o
  LD [M]  /root/LiME/src/lime.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/LiME/src/lime.mod.o
  LD [M]  /root/LiME/src/lime.ko
make[1]: Leaving directory '/usr/src/kernels/4.18.0-80.el8.x86_64'
strip --strip-unneeded lime.ko
mv lime.ko lime-4.18.0-80.el8.x86_64.ko
[root@localhost src]#
```

Here,what we have done is that we have compile the LiMe for a specific kernel as loadable kernel object.

```
[root@localhost src]# make
make -C /lib/modules/4.18.0-80.el8.x86_64/build M="/root/LiME/src" modules
make[1]: Entering directory '/usr/src/kernels/4.18.0-80.el8.x86_64'
  CC [M]  /root/LiME/src/tcp.o
/root/LiME/src/tcp.c: In function 'setup_tcp':
/root/LiME/src/tcp.c:75:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after
-statement]
     int opt = 1;
     ^~~
  CC [M]  /root/LiME/src/disk.o
  CC [M]  /root/LiME/src/main.o
  CC [M]  /root/LiME/src/hash.o
  CC [M]  /root/LiME/src/deflate.o
  LD [M]  /root/LiME/src/lime.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/LiME/src/lime.mod.o
  LD [M]  /root/LiME/src/lime.ko
make[1]: Leaving directory '/usr/src/kernels/4.18.0-80.el8.x86_64'
strip --strip-unneeded lime.ko
mv lime.ko lime-4.18.0-80.el8.x86_64.ko
[root@localhost src]# ls
deflate.c  disk.o  lime-4.18.0-80.el8.x86_64.ko  lime.mod.o  main.o          modules.order   tcp.o
deflate.o  hash.c  lime.h                        lime.o      Makefile        Module.symvers
disk.c     hash.o  lime.mod.c                    main.c      Makefile.sample  tcp.c
[root@localhost src]# _
```

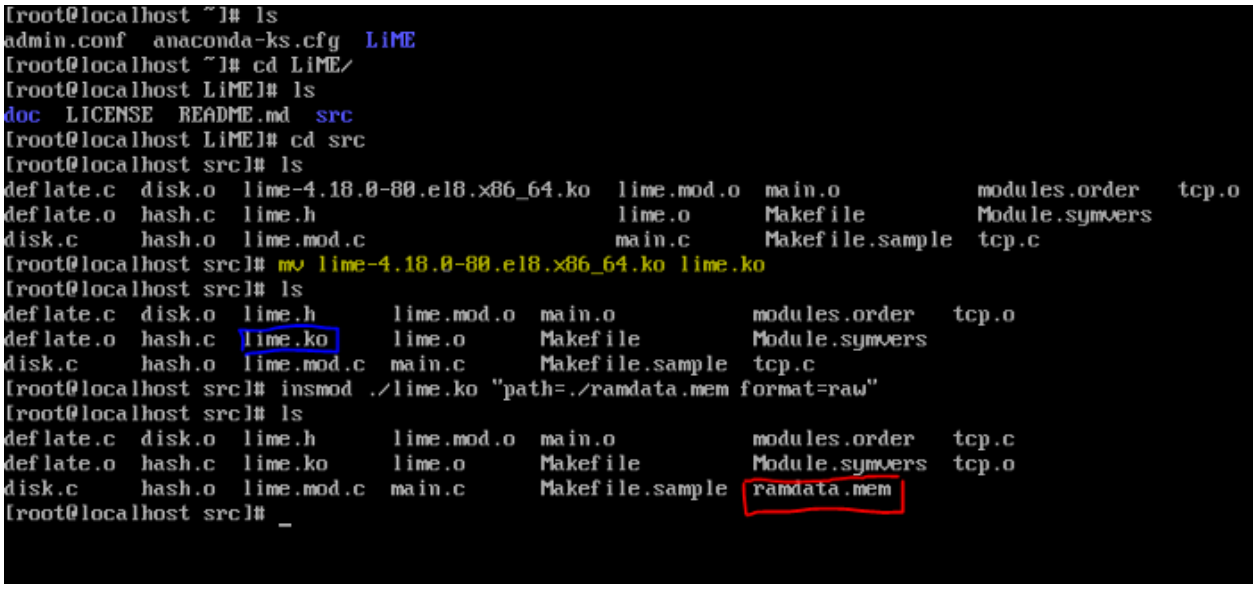
But before we have to generate some data in ram so once we dump ram data we can verify with it.

```
[root@localhost src]# python3
Python 3.6.8 (default, Jan 11 2019, 02:17:16)
[GCC 8.2.1 20180905 (Red Hat 8.2.1-3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> sachin=[1,2,3,"testing"]
>>> x=5
>>>
```

Now let's insert the kernel object we will provide the path and the format in which we want to save the image as

```
insmod ./lime-4.14.198-152.320.amzn2.x86_64.ko "path=./ramdata.mem
format=raw"
```

Depending on the ram size and disk I/O speed it will take time to dump ram data. you can give any name to folder like I have provided "ramdata.mem"



NOTE: “When you compile LiME will append the kernel version to the file name. Make sure you are using the full .ko file name when using insmod, or rename the .ko file to “lime.ko”

In the above image we have created a “ramdata.mem” file this contains all ram data at that point of time now we can verify it that the python variable we had created earlier

Type this command to check if variable value resides in ram or not

```
cat ramdata.mem | strings | grep "x=5"
```

```
[root@localhost src]# cat ramdata.mem | strings | grep "x=5"
x=53

c<x=5m
>>> x=5
gx=5
24;i=3cb;b=efc77e6eba2944f6af18ff2f09355655;m=3166ba6;t=5ccad678266a8;x=513b28dbefab9ab
    >>> vtor.check('tuple(max=5)', (1, 2, 3, 4, 5, 6))
    >>> vtor.check('tuple(min=3, max=5)', (1, 2, 3, 4))
    >>> vtor.check('list(max=5)', (1, 2, 3, 4, 5, 6))
    >>> vtor.check('list(min=3, max=5)', (1, 2, 3, 4))
c<x=5m
x=53
    >>> vtor.check('list(max=5)', (1, 2, 3, 4, 5, 6))
    >>> vtor.check('list(min=3, max=5)', (1, 2, 3, 4))
x=53
--dhcp-lease-max=50
x=53
x=5/
x=5/
x=52
x=52
 x=5
 x=5
0x=5
0x=5
```

we can cat the ramdata.mem and pipe it to strings because ram contains data in binary or other encodings so strings will convert it into a string and then we can grep with the variable name.

Now we have verified that value and variable is stored in the RAM memory, we can different tools and can do more analysis here to get details about CPU caches or every network connection details, socket information, website info, caches, tokens, passwords, usernames, encrypted disk data and a lot of other things.