CALIFORNIA STATE POLYTECHNIC POMONA

VOLITILE MEMORY EXTRACTION   
IN ANDROID MOBILE DEVICES

A PAPER

SUBMITTED TO DR. HUSAIN

IN PARTIAL FULFILLMENT

OF REQUIREMENTS FOR THE COURSE

ADVANCED INFORMATION SECURITY

CS 599

BY

JESSICA MILNE & ASHLEY BARTON

POMONA CALIFORNIA

DECEMBER 13, 2014

{JessicaMilne90, barton.ash}@gmail.com

**Table of Contents**

**Abstract……………………………………………………………………………………………………………………………………………. 3**

**Related Work……………………………………………………………………………………………………………………………………. 3**

**0.0 Introduction…………………………………………………………………………………………………………………………. 4**

**1.0 Background………………………………………………………………………………………………………………………….. 4**

**1.1 Hardware.………………………………………………………………………………………………………………………. 4**

**1.2 Operating System………….….……………………………………………………………………………………………. 5**

**1.3 Why Memory Forensics?..………………………………………………………………………………………………. 5**

**1.4 Solution Overview….………………………………………………………………………………………………………. 6**

**2.0 Linux Memory Extraction (LiME) …………………………………………………………………………………………. 6**

**2.1 Android Debugging, Explained.………………………………………………………………………………………. 6**

**2.2 LiME Overview.………………………………………………………………………………………………………………. 7**

**2.3 LiME Installation in Linux.………………………………………………………………………………………………. 8**

**2.4 Locate the Kernel Source..………………………………………………………………………………………………. 8**

**2.5 Config….…………………………………………………………………………………………………………………………. 9**

**2.6 Toolchains………………………………………………………….…………………………………………………………. 10**

**2.7 AVD………………………………..……………………………………………………………………………………………. 10**

**2.8 Custom Bootloader………………………………………………………………………………………………………. 10**

**2.8 Extract Memory over TCP ………………………………….…………………………………………………………. 10**

**3.0 Linux Forensics with Volatility.…………………………………………………………………………………………… 11**

**3.1 Linux…..…………………………..……………………………………………………………………………………………. 11**

**3.2 Android SDK and NDK……..……………………………………………………………………………………………. 12**

**3.3 Volatility.………………………..……………………………………………………………………………………………. 12**

**4.0 References…………………..……….……………………………………………………………………………………………. 13**

**Abstract:**

This paper describes the Linux Memory Extractor (LiME) tool used to extracting android memory. We also briefly discuss usage of the Volatility framework and Linux specific internals necessary to know to use Volatility effectively. LiME and Volatility are a great combination of tools used to not only extract memory from an Android device but also to dissect notable information. Since LiME operates as a kernel module, we discuss the process of installing and flashing an Android device in depth. LiME and Volatility are still supported as they are still relatively new forensics tools.

**Related Work:**

Our research showed a couple papers and conference presentations related to this topic[[1]](#footnote-1). One paper[[2]](#footnote-2) by research students at the Edith Cowen University completes an in depth analysis on the internals of LiME and Volatility. The second paper surveys Advanced Forensic Format (AFF) and AImage, competitors to Volatility and LiME respectively. There were a few broad presentations given on LiME and Volatility by the SANS Institute[[3]](#footnote-3) but no research other than the outcomes mentioned in the slides was indicated.

**0.0 Introduction**

As Android continues to be a main contender in the mobile space, more and more applications (malicious and otherwise) are being developed. Recent advanced malware has shown a clear attempts to operate more anonymously within the device making its actions less traceable. One way to combat this attempt at erasing a program’s traces is the use of disk and memory forensics. Forensics is the system retrieving snapshots of computer storage (disk forensics) and volatile memory (memory forensics). Unlike disk forensics, however, memory forensics provides a unique insight into the current state of the memory space. Operating system calls within the device itself are more difficult to obfuscate. A greater grasp on memory forensics could potentially mean a greater chance of finding rootkits or seeing if an application has ill-advisable functionality that operates in an insecure manner. Our goal for this paper is to evaluate current forensics tools and the process behind extracting memory from an Android device. Section 1 contains background information regarding how hardware and Operating System components interact with memory. We also begin to make the case as to why memory forensics specifically is useful and what our solution is to getting a memory dump of an android device. The last Section contains information about the two main tools we used to get a memory image and the setup process for installing them on an Android device. In this section we also discuss Linux specific forensics information since Android operates on a Linux kernel.

**1. Background Information**

**1.1 Hardware**

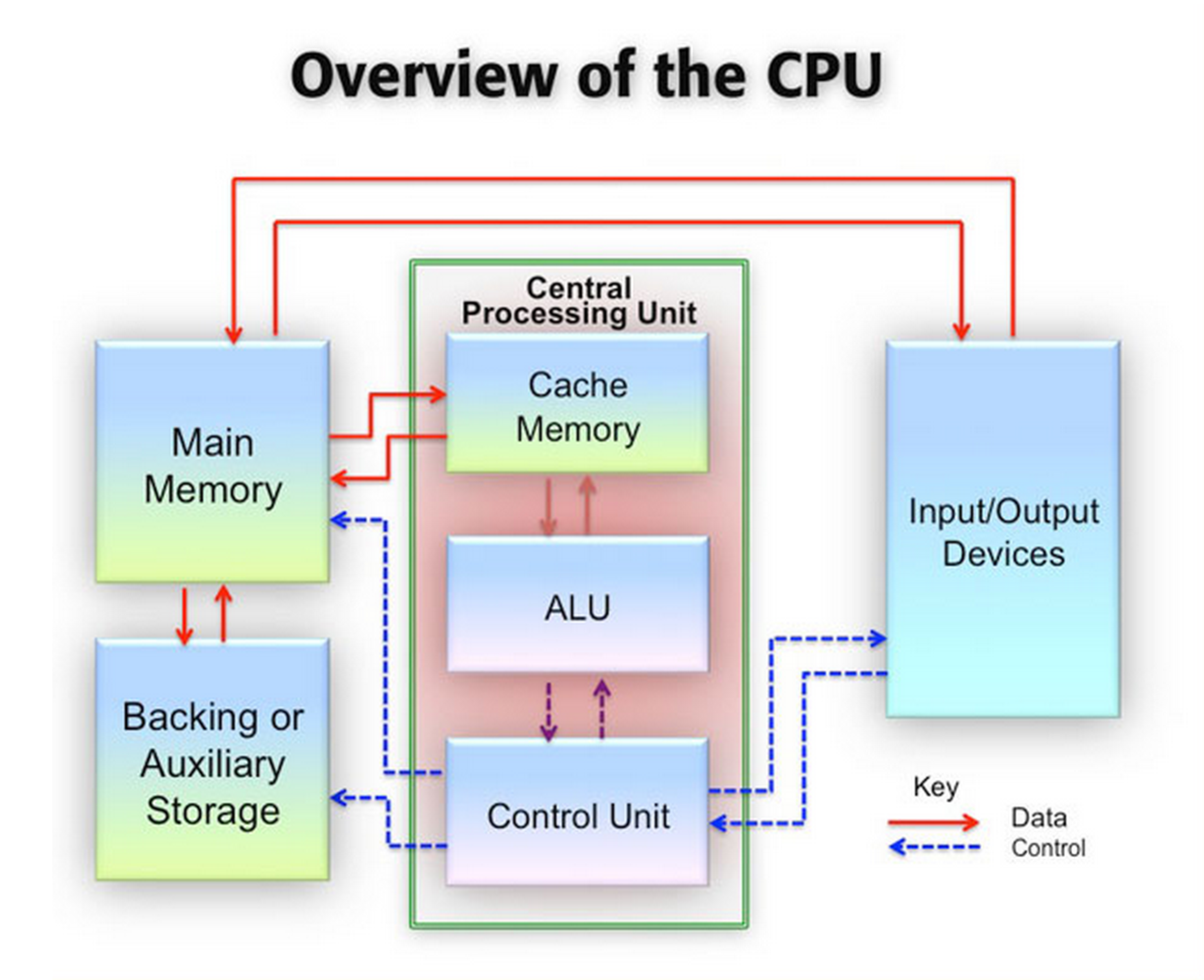
A computer contains a few key elements: the processor, motherboard, main memory, and other peripheral components such as hard drives and video cards. Highlighted below is the functionality of a few key components that heavily influence the state of volatile memory.

The central processing unit (CPU) is what executes the instructions required to run programs and operating system calls. Within the processor, we have caches. Caches are layers extremely fast memory delineated by “levels” (e.g. L1, L2) that are solely used by the processor for fetching frequently used instructions. If what the processor is searching for is not found in a particular cache, it continues through each level until it eventually reaches main memory.

Main memory, normally referred to as Random Access Memory (RAM), is the processors main resource for storing an applications information that it uses frequently. Main memory is referred to as “volatile memory” because it requires power to operate. Once the power is removed (i.e. you turn off your computer) the information stored in main memory is wiped and therefore no longer accessible. With memory forensics specifically, this is important to know since we will need to have the computer/device running to pull current information from main memory.

An important component to understanding output from a memory dump is to understand a computer’s architecture. The architecture generally dictates rules regarding the memory address space such as *paging*. Address spaces are referenced using page tables and directories to pull the relevant information programs require to operate. To save on memory, these page tables are used to swap out application instructions that are seldom being used. When forensics experts take images of RAM, they also include these page files (known as swap space) to see what hibernated programs exist.

Many of the frequently used tools that are useful for memory analysis emulate the devices specific architecture to effectively complete address translations. This is a brief overview of hardware interactions with RAM but those who frequently look into memory forensics must be familiar with many types of architectures and the particulars of different computers and/or devices.

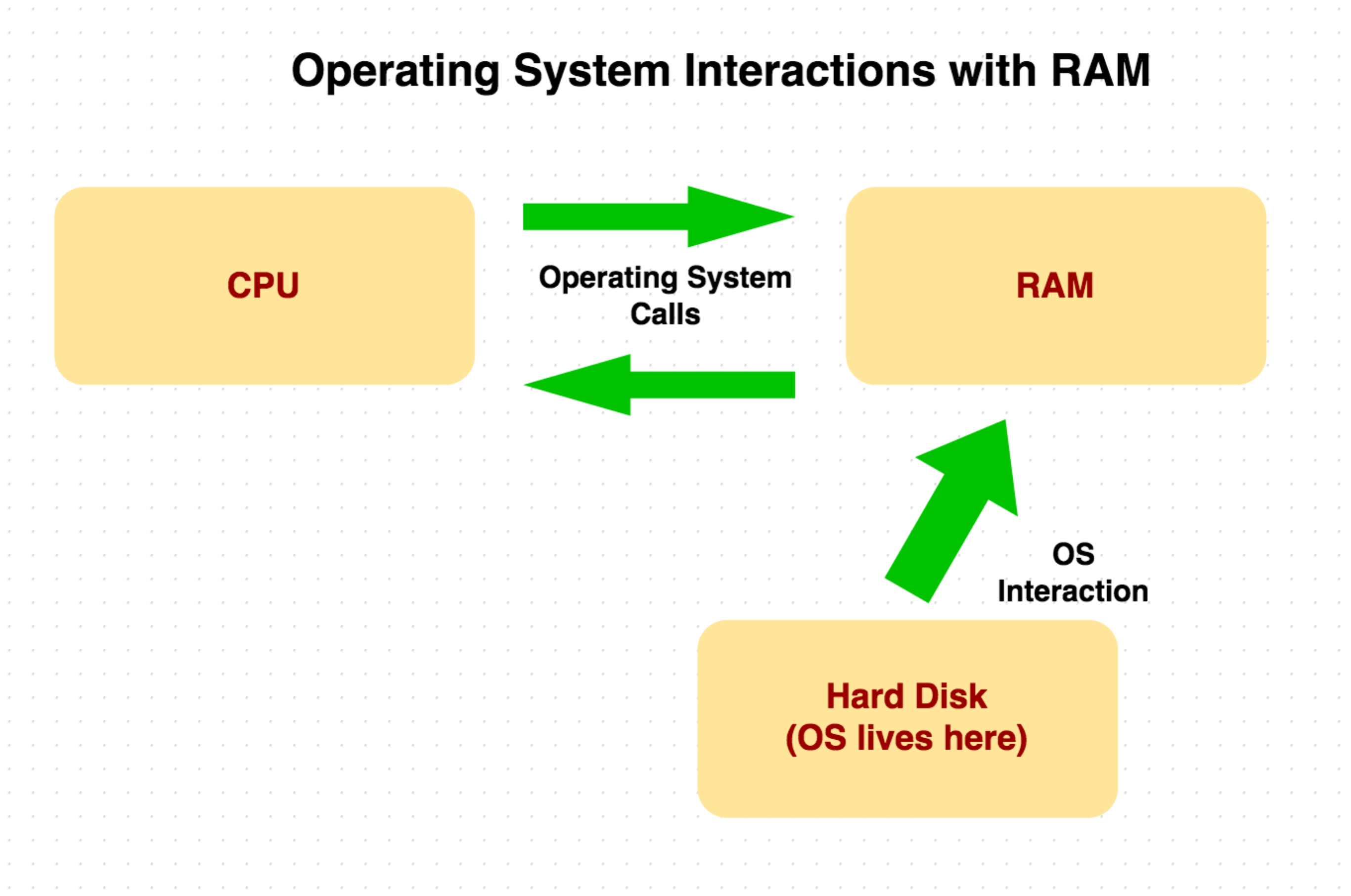


**1.2 Operating Systems**

Equally important to hardware, the operating system dictates a great deal about the current state of the machine. An operating system provides a user with the core services necessary for a computer to manage peripheral devices and handle applications. Much like hardware, operating systems are made up of a few key components: Device Drivers, System Utilities, a kernel and user interfaces. Of these components, the kernel, which loads applications into memory, is what interacts with RAM the most. The kernel also ensures that applications are sharing the CPU efficiently as well as manages whatever storage needs the application might have.

Since Android devices run on a Linux kernel, we will mostly discuss Linux functionality. In Linux, kernels are completely open source and allowed to be customizable unlike Mac OS and Windows which are closed source. We take advantage of Linux’s open source nature by installing our own custom kernel (a *kernel module*) on an Android device to extract memory information (discussed in depth in later sections). Kernel modules are especially useful because they can be loaded and unloaded on demand without having to restart the device or reboot the kernel.

The processes that are started once a kernel loads a program into memory can leave traceable artifacts within memory such as passwords, network sockets, URLs, open files, encryption keys, etc. Additionally, we can even see what sort of processes and threads are running. A process can be useful for forensics experts to gauge what a user or an application was doing at the time the image was taken.



**1.3 Why Memory Forensics?**

Digital forensics as a whole generally falls into three categories, disk forensics, network forensics, and memory forensics.

**Disk forensics** analyzes data saved on storage media such as hard drives, USB devices, memory cards, DVDs, etc. to retrieve information. Disk forensics is useful to find saved files or to see what programs are installed on a given device. Additionally, disk images do not lose information if they are not powered on as compared to volatile memory.

**Network Forensics** involves analyzing network traffic. Captured packets can be reassembled and analyzed for information. Network forensics, while useful in many other situations, isn’t necessarily applicable for the purposes of our research.

**Memory forensics** is particularly useful to see what a user or application is currently doing on a system. Even if a file was deleted it can still exist in memory and therefore be available for retrieval. As mentioned previously, memory forensics can hold information such as browsing history, chat logs, passwords, etc. on a device until that memory space is no longer used and potentially overwritten by the operating system. Information retrieved from Linux commands like lsof (list open files), ps (list of process statuses), netstat (network statistics) are not possible to obtain on a disk image but are easily accessible on a memory dump. This type of information can be very useful for malware analysis.

**1.4 Solution Overview**

Despite the availability of other types of memory extractors and parsers, we chose to use LiME and Volatility in part due to their newer technology stack, as well as their active user base. Both of these tools are also fully supported on Android devices. We use LiME to extract the memory from an android device (process mentioned in depth in the next section) and Volatility to parse the information retrieved from LiME.

* LiME – Memory Extractor
  + Installed as a kernel module and therefore minimizes contact between kernelland and userland
  + Useful features like network dump over Android Debug Bridge (ADB)
* Volatility – Parses LiME output
  + Written in Python
  + GPL license
  + Active community with hundreds of plugins
  + Works well with LiME

**2.0 Linux Memory Extractor (LiME)**

**2.1 Android Debugging, Explained.**

You can interface with the device using ADB, android debug bridge. To set this up, you start by accessing the device’s settings. Additional system settings are hidden in the KitKat and Jellybean versions by default. This is to prevent idle changes by laypersons which may cause issues with normal device functionality. They are intended largely to be used by android app developers to easily test and collect functionality metrics. You unlock these settings by accessing “About Tablet”, and then you must click on “Build Number” five times in quick succession. You should see a popup informing you that the developer options have been unlocked. Then, you should see Developer Options available in the main settings screen. At the top of the screen is an enable button, which should be turned “on”. Scroll through this until you see Allow USB Debugging. USB debugging allows your computer to copy data from to your device, install applications without notification to the device user, and read log data.

After plugging in your device through USB to your computer, you now have several options. If you’ve installed the android SDK, you can navigate to the ./sdk/tools directory to find ADB. This allows you to push files to the device, pull from the device, or spawn a shell on the device with ADB shell. If ADB has trouble connecting, make sure the device is properly connected, unlock the device, and watch for a popup which will request access permissions for your computer.

The next step was to extract the memory image from the device. In order to do this, we used LiME, the Linux Memory Extractor. There are few other methods of retrieving memory from the device. Such as /proc/kcore, /proc/fmem, or /dev/mem. We will discuss a few of these methods below.



Figure 1 Graphical breakdown of the android memory structure.   
The kernel is a low level system operation which manages drivers and power.   
Image credit: androidteam.googlecode.com

It was once possible to extract memory on a Linux machine using /dev/mem, before security concerns caused it to be disabled. This device would allow programs such as dd, a copying program, to directly read from and write to RAM. RAM refers to random access memory. One issue with this method was that a tester could accidently access sensitive memory and cause memory corruption issues.   
  
The more modern version of this is known as fmem. This loads a kernel driver which creates a device /dev/fmem which functions in a similar way to /dev/mem. It creates certain protections which make it a better option for modern day forensics professionals. First, it checks whether a physical page resides in main memory before reading it. It accomplishes this by calling the page\_is\_ram function. This prevents certain memory corruption issues which can result from reading unmapped physical addresses or memory which is in use. fmem is an excellent tool for experienced forensic professionals due to the fact that it requires intimate knowledge of memory layouts in order to be used properly. In this paper, our inexperience led us to use a

more beginner friendly tool known as Linux Memory Extractor (LiME).

**2.2 LiME Overview**

LiME works by loading a custom Linux Kernel Module (LKM) into the kernel source files. The term module refers to a separable component of a larger, more complex system. The kernel is a very low level system structure which resides below the operating system.

The kernel manages input / output requests from software. It translates the higher level requests from applications into data processing instructions for the Central Processing Unit (CPU). It’s important that the kernel is not modified while the system is in use, because if parts of it were to be overwritten, the device would be rendered inoperable. For that reason, the kernel’s memory location is protected.

Memory extraction is done from within the kernel because it increases the accuracy of the memory image. This is because there are no context switches from the user memory space to the kernel memory space in order to transfer data. LiME also resolves certain issues present in fmem regarding address range mapping. It determines these address ranges automatically by reading the kernel’s iomem\_resource linked list. In contrast, fmem requires the investigator to read this list by hand, which can present issues if the investigator is not skilled in the memory structure of the device.

**2.3 LiME installation in Linux**

LiME[[4]](#footnote-4) memory acquisition can be done via Transmission Control Protocol (TCP) or through the device’s SD card.

LiME requires the user to cross compile a custom kernel and flash it onto the device in order to extract memory. The reason this is necessary is because android devices do not allow custom module support by default. You can test this by testing whether insmod works.

If you compile the LiME module lime.ko and try to push it onto a device with the standard kernel configuration, it informs you that the function is not implemented.

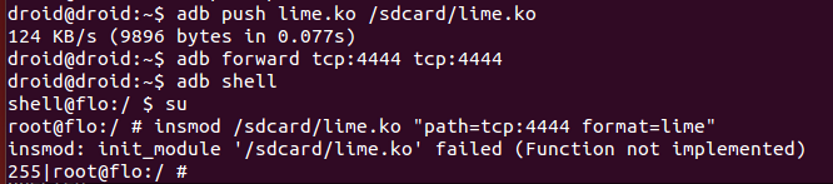


Figure 2 LiME installation attempt, on a device with the default kernel configuration.

insmod is a program which allows the super user to insert a module into the linux kernel. Here, we are inserting lime.ko into the kernel, and adding the options "path=tcp:4444 format=lime".

You can also double check that modules are not supported by checking the output of lsmod.



Figure 3 The output of lsmod on the same device (See fig 2)

Lsmod shows the status of other modules which have been loaded into the kernel. In this case, we see that the /proc/modules file does not exist. This file will exist if and only if the kernel supports loadable modules.

The LiME documentation for android describes some of the steps required in order to create the custom kernel and cross compile it. However, the steps are not straight forward for every device. Some of the issues we encountered include collecting the kernel configuration file, downloading the kernel source files, and locating the correct toolchain. We will explain each of these in detail. Please note that you will need a rooted device to continue with these instructions, and not every device can currently be rooted. Be advised that rooting also voids your manufacturer warranty.

**2.4 Locate the Kernel Source**

“Download and untar the kernel source for your device. This can usually be found on the website of your device manufacturer or by a quick Google search.” This is the casual recommendation from the LiME android documentation.

In our case, finding the kernel source was a more involved process than just a quick google search. You begin by researching details about your device. In this experiment, we were using a Nexus 7, 2013 version. These are manufactured by Asus. The chipset of this model is Qualcomm Snapdragon S4 Pro APQ8064–1 AA. The device name is flo. The operating system version is 4.4.2, also known as Kit Kat.

Asus did not have the kernel source files available on its website. The Nexus 7 downloads offered only drivers and some documentation. After more research, I located the android Git repository, located at android.googlesource.com

The android git repository does not have an abundance of documentation. The android developers guide has you clone the entire thing, which saves them the trouble of enumerating its contents. However, this is not a terribly efficient practice if you are only working with a single device type. Considering the details of mine, there were a few different options which might have held the kernel;

device/asus/flo  
device/asus/flo-kernel  
device/asus/grouper

Several guides described the file structure I was looking for, but none of these repositories held the files that I needed to cross compile. The flo-kernel was only the kernel image, and I needed all the associated files.

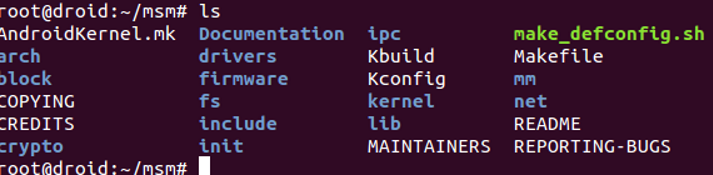


Figure 4 Kernel source files for the Nexus 7

I eventually was able to find the actual kernel source inside;   
/kernel/msm

These are the kernel source files which are designed specifically for Qualcomm chipsets.

The master branch of this repository was empty, however. This is likely because each kernel version does not need to build upon the last one. The branch I ultimately had to clone was android-msm-flo-3.4-kitkat-mr0.

**2.5 Config**

In order to cross compile your kernel, you also need access to the kernel configuration. This is in order to replicate the same setup that was in place before you began. Collecting the kernel config took some effort. In some Linux machines, the kernel config is easy to extract. It is normally located in a file called /proc/config.gz. This file is created if a certain flag is set during the kernel compilation, called CONFIG\_IKCONFIG\_PROC. The LiME documentation informs you to simply pull the file and continue, but if it doesn’t exist there are other methods to collect a version of it.

Our device did not have this file available.It would have been preferable to extract the config from the live device, in order to ensure maximum compatibility. However, since the config is not created automatically, I did not discover a good alternate method to accomplish this. I extracted the config from an equivalent kernel source.

The flo-kernel image is available on the android git repository, and there is a script available which can extract the config from a kernel image on the linux kernel source tree repository. (See references for the link). By running this script on the image in flo-kernel, I was able to extract a version of the config which would suit our purposes.

In order to enable loadable module support, you will need to modify it the config. The config is only a text document with a list of flags and their settings, so it may be tempting to simply change them by editing the file. However, this did not produce the intended result.

The better way to change the kernel config is using make menuconfig. You begin by copying the config file you attained earlier into the root directory of the kernel source, using the name .config. Then, you set a few temporary environment variables.

**2.6 Toolchains**

Next, we need to add the ARM toolchain to our path. The path is a set of directories which contain executable programs. It saves the programmer the effort of describing the full paths of every object that is required to compile code. ARM refers to a family of instruction set architectures (ISA) involved in creating processors.

A toolchain is a set of distinct software development tools which are linked together by specific stages such as the GNU C Compiler, GCC. By creating these links, it aids compilation of certain kinds of complex code.

The NDK has sets of toolchains for many different kinds of chipsets, and several In our case, we used ./ndk/toolchains/arm-linux-androideabi-4.8/prebuilt/linux-x86\_64/arm-linux-androideabi/bin/gcc

As our toolchain. “androidabi” in this context refers to the android Application Binary Interface (ABI). An interface between two program modules, such as a library or operating system, at the level of machine code is called an ABI.

Once you have located the correct toolchain, creating the custom kernel is not too difficult. You simply run make menuconfig and change the required setting. Menuconfig also provides a graphical user interface to streamline the process.

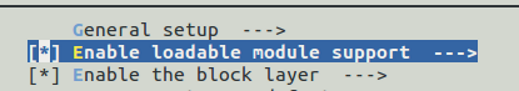


Figure 5 Allowing custom module support in our new kernel.

After that, if all went well, you can run

# make clean && make mrproper

If all goes well, this should create the custom kernel image, which can be found at ./arch/arm/boot/zImage

**2.7 AVD**

Now that we’ve created the zImage of our modified kernel, we can load it into an Android Virtual Device (AVD), and check that the device will boot. Eclipse is a good solution for testing. You only need to install the android software development kit (SDK) as discussed earlier in order to allow it to run AVDs.

**2.8 Custom Bootloader**

Now that we are sure that we have a working kernel, we need to flash it onto the device. The usual method is to download all 16 GB of the Android Open Source Project (AOSP), and then figure out where you can inject your kernel into that directory tree.

A bootloader is a device which is executed before the Operating System (OS). It contains the instructions to boot the operating system kernel. Because it is so low-level, the bootloader is extremely specific to your processor. Every motherboard has its own bootloader. Android in particular has a lot of variance in the processing hardware on the device. In comparison, Apple iOS devices share a lot more hardware similarities.

Loading the kernel involves more than just pointing your bootloader to it. You must actually create a custom bootloader as well as a RAM disk in order to continue.

Android bootloaders are locked by default, to prevent user customization. Until it is unlocked, it is almost impossible to flash custom ROMs onto the device ROM refers to read only memory. It is a form of non-volatile memory which usually stores critical system instructions.

Unlocking the bootloader is quite simple. Begin by booting the device in bootloader mode. This is normally a device-specific key combination. Then, connect the device via USB, and type

# fastboot oem unlock

Then, you will see the change reflected on your device display. Now we can create and flash the custom bootloader.

We created a custom boot image to load our kernel. A boot image is a kind of disk image containing the contents and structure of the operating system, utilities, boot and data recovery programs. It allows the hardware to boot.

Pete at akeo.io created a custom unpack tool which improves upon the default android one. The android default requires a lot of research in order to use it, because it does not provide the parameters needed to unpack the boot image.

Unpacking will create the boot.img files that we need to flash onto the device. Then, you can simply run this command while the device is in bootloader mode.

# ./fastboot boot customboot.img

**2.9 Extract Memory over TCP**

After these steps are followed, you should be able to collect the android memory dump from LiME.

$ adb push lime.ko /sdcard/lime.ko  
$ adb forward tcp:4444 tcp:4444  
$ adb shell  
$ su  
# insmod /sdcard/lime.ko "path=tcp:4444 format=lime"

# exit  
$nc localhost 4444 > ram.lime

**3. Linux Forensics with Volatility**

**4.0 References**

Heriyanto, Andri P. "Procedures And Tools For Acquisition And Analysis Of Volatile Memory On Android Smartphones." *Http://ro.ecu.edu.au/*. Australian Digital Forensics Conference, 2 Dec. 2013. Web. 15 Dec. 2014. <http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1122&context=adf>.

"Android Mind Reading: Acquistion and Analysis with LiME and Volitality." *Digital-Forensics SANS*. Web. 15 Dec. 2014. <https://digital-forensics.sans.org/summit-archives/2012/android-mind-reading-memory-acquisition-and-analysis-with-lime-and-volatility.pdf>.

*Unmasking Careto Through Memory Analysis*. SecTor 2014, 2014. Film.

http://2014.video.sector.ca/video/110388398

Torvalds, Linus. "Kernel/git/torvalds/linux.git - Linux Kernel Source Tree."*Kernel/git/torvalds/linux.git - Linux Kernel Source Tree*. Web. 15 Dec. 2014. <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/scripts/extract-ikconfig>.

"Pete's Blog." *: Compiling and Running Your Own Android Kernel on the Nexus 7 2013.* Web. 15 Dec. 2014. <http://pete.akeo.ie/2013/10/compiling-and-running-your-own-android.html>.

1. http://2014.video.sector.ca/video/110388398 [↑](#footnote-ref-1)
2. http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1122&context=adf [↑](#footnote-ref-2)
3. https://digital-forensics.sans.org/summit-archives/2012/android-mind-reading-memory-acquisition-and-analysis-with-lime-and-volatility.pdf [↑](#footnote-ref-3)
4. There is a package named “lime” available through the Advanced Packaging Tool (APT) in Linux. However, be advised that this is actually an unrelated set of web development tools which coincidentally uses a similar name. [↑](#footnote-ref-4)