Digital Investigation 8 (2012) 175–184

Contents lists available at SciVerse ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Acquisition and analysis of volatile memory from android devices

Joe Sylve [a], Andrew Case [b], Lodovico Marziale [b], Golden G. Richard [a,*]

[a] Department of Computer Science, University of New Orleans, New Orleans, LA 70148, USA
[b] Digital Forensics Solutions, LLC, New Orleans, LA 70130, USA

ARTICLE INFO

Article history:
Received 27 April 2011
Received in revised form 6 September 2011
Accepted 24 October 2011

Keywords:
Android
Memory forensics
Memory analysis
Linux
Mobile device forensics

ABSTRACT

The Android operating system for mobile phones, which is still relatively new, is rapidly gaining market share, with dozens of smartphones and tablets either released or set to be released. In this paper, we present the first methodology and toolset for acquisition and deep analysis of volatile physical memory from Android devices. The paper discusses some of the challenges in performing Android memory acquisition, discusses our new kernel module for dumping memory, named *dmd*, and specifically addresses the difficulties in developing device-independent acquisition tools. Our acquisition tool supports dumping memory to either the SD on the phone or via the network. We also present analysis of kernel structures using newly developed Volatility functionality. The results of this work illustrate the potential that deep memory analysis offers to digital forensics investigators.

1. Introduction

The Android operating system now has a substantial share of the mobile market, and is expected to lead the market by the end of 2011 (Eweek, 2011). The mass adoption of Android and its projected growth make it vital that the forensics community be able to properly acquire and analyze evidence from the platform. While a few research efforts have discussed analysis of Android's file system and analysis of process memory, we are not aware of any work to date that completely acquires physical memory and subsequently performs a coherent analysis of the acquired memory for Android devices. Physical memory analysis is vital to investigations, since it contains a wealth of information that is otherwise unrecoverable. This evidence includes objects relating to both running and terminated processes, open files, network activity, memory mappings, and more. Lack of such information can make certain investigative scenarios impossible, such as when performing incident response or analyzing advanced malware that does not interact with non-volatile storage.

In this paper, we explore the technical issues associated with acquiring physical memory captures from Android-based devices as well as subsequent analysis of the data acquired. We present a methodology for acquiring complete memory captures from Android, code to analyze kernel data structures, and scripts that allow analysis of a number of userland and file system-based activities. We believe that Android will continue to require future forensics research and in order to make our results immediately usable to researchers and investigators, we have integrated support for Android memory analysis into the Volatility Memory Analysis framework (Volatility, 2011). Since Volatility is already used extensively in real investigations, to support research in memory forensics, and in a number of training courses, we hope our results will generate further interest in the Android platform.

2. Related work

The presented research encompasses a number of related work areas as it includes acquisition of memory and a number of analysis techniques.

* Corresponding author. Tel.: +1 504 280 6045.
E-mail address: golden@cs.uno.edu (G.G. Richard).

1742-2876/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2011.10.003

2.1. Linux volatile memory analysis

In the last few years, there has been a substantial amount of memory analysis research targeting Linux. The first systems presented for this purpose were the FATKit (Walters, 2006), (Burdach, 2004), and memparser (Betz, 2005). Inspired by the DFRWS 2008 challenge (DFRWS, 2008), additional efforts were made to extract forensically relevant information from memory captures (e.g., Case et al., 2008). Since then, a number of other research projects have been presented that perform deep analysis of Linux kernel data structures as well as userland information (Case, 2011; Case et al., 2010a, 2010b; Kollar, 2010). The result of these projects is the ability to gather numerous objects and data structures relevant to forensics investigations in an orderly manner. A shortcoming of these projects, however, was their inability to properly handle the vast number of Linux kernel versions and the large number of widely used Linux distributions. Due to the issues investigators face when attempting to analyze one of a large number of Linux kernel versions, a number of recent research projects have attempted to automatically build kernel structure definitions through a combination of static and dynamic analysis (Case et al., 2010a, 2010b; Cozzie et al., 2008; Lin et al., 2010; Slowinska et al., 2011).There has also been recent work by the Volatility developers to automatically generate C kernel structure representations for different Linux kernel versions using debugging information, which is similar to how Volatility handles different versions of the Windows kernel.

While the these projects were able to recover both allocated and de-allocated instances of kernel structures, many of them relied on either following references within data structures or memory scanning using ad-hoc structure signatures. The ability to accurately find data structures to which all references are removed is required in order to find completely freed objects. The problem with current generation scanners, such as those discussed previously, is that the signatures were created based on manual and informal source code review by the project developers. Illustrating serious problems with this approach, including the ease in which malware can bypass such weak signatures, were two publications that used virtual machine introspection and formal methods to construct structure signatures (Dolan-Gavitt et al., 2009; Lin et al., 2011). Using the techniques presented in these publications, forensic investigators are able to scan for instances of data structures with a degree of confidence, since malware is unable to easily bypass the signatures and false negatives and false positives will be minimal.

2.2. Linux memory acquisition

Traditionally, memory captures on Linux were acquired by accessing the */dev/mem* device, which contained a map of the first gigabyte of RAM. This allowed acquisition of 896 MB of physical memory without the need to load code into the kernel. This approach did not work for machines with more than 896 MB of RAM. Due to security concerns, the */dev/mem* device has recently been disabled on all major Linux distributions, as it allowed for reading and writing of kernel memory. In order to capture all physical memory, regardless of size, and to work around the loss of the */dev/mem* device, Ivor Kollar created *fmem* (Kollar, 2010), a loadable kernel module that creates a */dev/fmem* device supporting memory capture. *fmem* has been used in a number of incident response situations and is the defacto Linux memory acquisition tool. Another tool similar to *fmem* is the *crash* (Anderson, 2008) project by Redhat. For reasons we discuss later, the *fmem* module does not work on Android devices.

2.3. Android memory analysis

There are currently three projects that support varying levels of Android memory analysis. The first project, volatilitux (Girault, 2010), provides only limited analysis capabilities, including enumeration of running processes, memory maps, and open files, and does not provide a method to acquire memory from the phone. Our techniques provide both acquisition and analysis capabilities.

The second related work was published in DFRWS 2010 (Thing et al., 2010). This research project avoided the technical issues with capturing physical memory on Android (which we solve in this paper), by focusing on specific, running processes, and using the ptrace functionality of the kernel to dump specific memory regions of a process. The virtual memory captures are then analyzed to discover evidence. While this is a good first step, many important aspects of the Android device's memory are not analyzed, including in-kernel structures, networking information, etc. Another concern is that the approach requires memory to be extracted separately for each process of interest, which requires a number of interactions with the live system and potentially overwrites valuable evidence. We concentrate instead on physical memory acquisition and analysis, which provides a superset of the information contained in the address spaces of individual processes.

Finally, another tool that is capable of extracting process memory is *memfetch* (Zalewski, 2002). This tool dumps a running application's address space, either on demand or when faults (e.g., SIGSEGV) occur. *memfetch* is portable across a variety of Linux distributions, including Android, but cannot acquire physical memory.

3. Acquiring volatile memory

In this section we discuss memory acquisition for Android and our discussion is broken into a number of sections for readability. Section 3.1 explains how to prepare a phone for memory acquisition, Section 3.2 discusses issues with existing acquisition modules, and Section 3.3 discusses portability issues.

3.1. Preparing the phone

Preparation of the phone for memory acquisition requires a number of steps, since Android does not support a memory device that exposes physical memory and furthermore does not provide APIs to support userland memory acquisition applications. This means that