Chapter 21

# FORENSIC ANALYSIS OF PLUG COMPUTERS

Scott Conrad, Greg Dorn and Philip Craiger

**Abstract** A plug computer is essentially a cross between an embedded computer and a traditional computer, and with many of the same capabilities. However, the architecture of a plug computer makes it difficult to apply commonly used digital forensic methods. This paper describes methods for extracting and analyzing digital evidence from plug computers. Two popular plug computer models are examined, the SheevaPlug and the Pogoplug.

**Keywords:** Plug computers, forensic analysis, SheevaPlug, Pogoplug

## 1. Introduction

Personal digital devices are becoming smaller and cheaper, an example of which is the plug computer. Plug computers are a cross between an embedded device (e.g., smart phone) and a traditional computer (e.g., laptop or desktop). Plug computers have the same general architecture as traditional computers (CPU, RAM, non-volatile memory, system bus, etc.), but are considerably smaller and less powerful. However, the forensic extraction and analysis of digital evidence from plug computers are neither as straightforward nor as simple as for typical desktops and laptops.

An example plug computer is the SheevaPlug [10] shown in Figure 1. It is roughly the size of a large A/C power supply adapter. SheevaPlug's low power consumption and cost (around $65) makes it ideal for use as a small-scale server, such as a home network attached storage (NAS) server.

Since their debut in early 2009, plug computers have generated considerable interest among developers and hobbyists, resulting in numerous applications ranging from file serving to cloud computing. Like desktops

*Figure 1.* SheevaPlug plug computer.

and laptops, plug computers have the potential to be used in nefarious ways, all the while maintaining a low profile because of their small size.

Despite the increasing popularity of plug computers, there is no published research on forensic procedures for extracting and analyzing digital evidence from these devices. This paper attempts to fill the void by discussing digital forensic procedures for two widely distributed plug computers, the SheevaPlug [10] by Marvell Semiconductor, and one of its commercial successors, the Pogoplug by Cloud Engines [12].

## 2. Plug Computer Overview

This section provides an overview of the SheevaPlug and Pogoplug plug computers, which are the focus of this paper.

The SheevaPlug uses a licensed version of ARMv5, an ARM architecture that is commonly found in cell phones. Its Marvell 88F6000 CPU is clocked at 1.2 GHz and has 16 KB of L1 cache and 265 KB of L2 cache, connected via a 64-bit MBus (system bus) clocked at 166 MHz. The SheevaPlug has 512 MB of DDR2 RAM and 512 MB of NAND flash memory, which serves as non-volatile memory (storage). It incorporates several external interfaces: a gigabit Ethernet port, a USB type A port, a mini USB port (used as its JTAG [8] and serial interface), and a secure digital input output (SDIO) flash card slot [10].

The Pogoplug has an almost identical architecture, except that it has no mini USB port and no SDIO flash card slot. The absence of a mini USB port makes it difficult to extract digital evidence from the Pogoplug's NAND flash memory.

A plug computer can be loaded with any operating system that supports the ARM architecture. However, Linux is the most commonly used operating system. Several pre-packaged Linux distributions are available, including Ubuntu, Gentoo, Debian, FreeBSD and Fedora [11].

The primary challenge in performing a forensic analysis of a plug computer is dealing with flash memory (non-volatile storage). Flash memory does not behave in the same manner as more common types of storage media such as a hard drive. For instance, writing to flash memory is done in complete blocks – changing even a single byte requires a complete block (typically 512 K) to be rewritten. This hinders the ability to recover deleted and unallocated data because of the likelihood that slack space will be overwritten; this also creates the need for a special file system that supports the writing of complete blocks.

The two most common file systems used with the SheevaPlug and Pogoplug are the Journaling Flash File System version 2 (JFFS2) and its successor, the Unsorted Block Image File System (UBIFS) [20–22]. These two file systems, as well as most other flash memory specific file systems, are log-structured systems that rely on the Memory Technology Device (MTD) subsystem [17]. Interested readers are referred to [21] for additional details.

## 3. Interfacing with Plug Computers

The Joint Test Action Group (JTAG) [8] interface is an important mechanism for extracting the contents of plug computer memory. The JTAG interface is a debugging tool that allows users to directly control the circuits and chips on a memory board without accessing the operating system. In the case of plug computers, the JTAG interface is used primarily for reloading the boot loader (U-Boot, which is described below) and the operating system, although this can be accomplished over a serial connection using removable memory or a network protocol.

In a forensic investigation, the JTAG interface may be used to directly access and extract the memory contents. This is accomplished by directly controlling the CPU, having the CPU read the contents of a memory bank (RAM or NAND), and forwarding the bits back through the JTAG interface so that they can be captured and stored. The Open On-Chip Debugger (OpenOCD) software may be used for this purpose as it can control many different aspects of a plug computer through the JTAG interface as well as copy data to and from memory [1, 14].

As mentioned above, the Pogoplug does not have a built-in JTAG interface like the SheevaPlug. However, the Pogoplug does have an 8-pin JTAG header on its motherboard, which makes it possible to use JTAG

to acquire the memory contents. To accomplish this, a hardware JTAG adapter must be manually interfaced with the Pogoplug – since the 8-pin header has not been standardized – and then configured to work properly with the ARMv5 processor. Subsequently, OpenOCD or other similar software may be used to communicate with the processor. Although this can be accomplished in theory, we were unable to implement the JTAG interface during our research. Note that a serial interface can also be created by modifying a cable and then attaching the cable to a 4-pin header on the motherboard [13, 19].

The boot loader U-Boot [3, 5, 6] is used to initialize and load the operating system when a plug computer is powered on (assuming, of course, that an operating system is already installed). Note that U-Boot has a short timeout during which a user can manually abort the loading of the operating system via a serial connection. The U-Boot shell works much like the BASH shell in that it can run simple commands such as displaying or altering the current environmental variables; also, it supports simple shell scripts. In most cases, executing the `printenv` command to display environmental variables reveals what the boot loader will do when it attempts to boot the operating system. Interested readers are referred to [6] for additional details about U-Boot.

## 4. Imaging Non-Volatile Memory

Acquiring a bit-for-bit copy of the non-volatile memory is one of the first steps in a forensic investigation. This is a simple process in a typical desktop or laptop computer because the non-volatile memory (hard drive) can be removed and attached to a physical write blocker to obtain a forensic duplicate (image). However, this is difficult to do for a plug computer because the NAND flash memory is soldered to the motherboard. The most direct means of obtaining a copy of the memory is to desolder the chip from the motherboard and move it to a device that is capable of reading their contents. This is dangerous, however, because of the likelihood of losing or corrupting the data on the chip. Additionally, this method requires expensive, specialized equipment and training. For these reasons, we suggest that a desoldering method be employed only as a last resort.

Another method for creating an image is to boot the plug computer and use Linux utilities to create a forensic image. This task is easily accomplished using `dd` to read data directly from the flash device (`/dev/mtd`) and then stream the bits to a second target computer using the `netcat` utility (which reads and writes a stream of bits). Because there is no way to prevent writing to the NAND flash memory – for

example, by inserting a write blocker – this is not a forensically-sound method for creating an image. Furthermore, the act of booting a plug computer (like any other computer) causes the data in memory to change in some way. There is also the possibility that a sophisticated user could use an anti-forensic measure or create daemons to automatically destroy the data if the plug computer is not booted in a specific way.

We have discovered that accessing memory through the JTAG interface is (arguably) forensically acceptable because it prevents nearly all types of writes to the memory. As described above, the JTAG interface allows for direct control over the plug computer via a USB port. This means that memory dumps of the NAND flash memory and RAM can be obtained directly from the plug computer without accessing the operating system or any other software running on the device. Thus, the only risk of unauthorized writes to plug computer memory comes from the JTAG software (OpenOCD) and the boot loader (U-Boot).

Note that the U-Boot environment can directly copy areas of memory to a target. Accordingly, we surmised that it should be possible to use a serial connection to copy data from the internal flash memory to external storage in order to obtain a forensic image, thus eliminating the need to use a JTAG scheme. Unfortunately, our attempts were unsuccessful because the copy command would not allow the target to serve as an external memory device [5].

## 5. Obtaining a Flash Memory Dump

This section describes the process we used to obtain a dump of the internal flash memory of the SheevaPlug. The acquisition computer was an Intel-based desktop computer running Ubuntu 9.10 (Linux kernel v2.6.31) and OpenOCD v0.2.0. We used a USB type A to mini USB data cable to connect the JTAG interface from the SheevaPlug to the acquisition computer.

### 5.1 Connecting to the SheevaPlug

Connecting the laboratory computer to the SheevaPlug plug computer using OpenOCD involves the following steps:

- Connect the laboratory computer to the SheevaPlug using the JTAG interface (mini USB port).
- Power on the SheevaPlug by plugging it into a power outlet.
- Use a terminal emulator application (e.g., `PuTTY`) to connect to the SheevaPlug via the serial port (usually `/dev/ttyUSB1` in Linux).

*Figure 2.* Stopping the operating system from booting.

This must be done quickly because the user only has a few seconds to stop the operating system from booting.

- Stop the operating system from booting by pressing any key at the prompt (Figure 2).

- Run OpenOCD on the laboratory computer (Figure 3). The OpenOCD manual [15] describes how to point the software to the correct SheevaPlug configuration file.

*Figure 3.* Running OpenOCD.

- Connect to OpenOCD via `telnet` by issuing the command `telnet localhost 4444`.
- Run the OpenOCD method `sheevaplug_init` over the `telnet` connection. This will reset the plug computer, but immediately halt it and initialize the plug computer to allow control over the NAND flash memory and RAM.
- Locate the flash devices by issuing the command `NAND list` and probe the devices using the command `NAND probe num` where `num` is a number given by `NAND list` (Figure 4).

## 5.2 Obtaining a Dump

The next step is to obtain a memory dump. The following steps are involved:

- Execute the command `NAND dump num filename beginning_offset length` to copy the memory contents from a flash device to a file on the laboratory computer. This action can take a considerable amount of time.
- Execute the command `dump_image filename beginning_offset length` to copy the contents of RAM to a file on the laboratory computer, if desired.
- Power off the plug computer when finished by pulling it out from the power outlet.

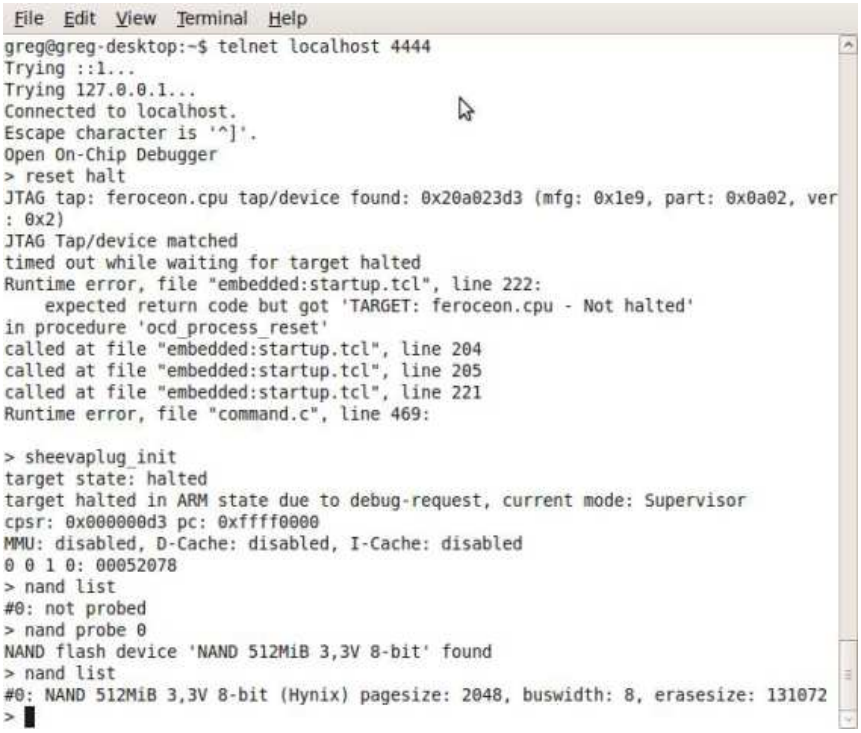


*Figure 4.* Locating and probing NAND flash memory.

This method does not allow the operating system to load, but it still achieves the goal of creating a forensic duplicate of the memory contents. Note, however, that a skilled programmer could alter U-Boot to corrupt the memory of a plug computer if the correct steps are not performed when powering on the device, but this is unlikely because the boot loader is a very small program and it may not be possible to reconfigure it to achieve such an effect.

In our experiments, it took more than a few hours to obtain a memory dump of the RAM and more than a month to obtain a dump of the NAND flash memory. The NAND flash memory dump has a transfer rate of roughly 1 MB per hour and all attempts to accelerate this process were unsuccessful. Writing to the NAND flash, on the other hand, is much faster; writing all 512 MB of memory only took a few hours.

## 5.3 Creating a Serial Connection to Pogoplug

Constructing a cable to establish a serial connection to the Pogoplug requires minimal soldering experience and inexpensive supplies. Infor-

mation about the type of cable required is available at [13] and a description of wires in the cable is available at [10]. The resulting modified cable can be used to connect to the 4-pin header on the Pogoplug's motherboard and establish a serial connection to the device.

## 6. Forensic Analysis of Flash Memory

After the NAND flash memory contents are extracted, there are only a few options available for analysis. We know of no forensic tools, including the most popular forensic suites, that have been developed to specifically analyze the data. Research on the subject of recovering deleted data from NAND flash memory [2, 16] focuses on the physical memory level rather than the logical file system level. Thus, the only reasonable option is physical analysis with a hex editor. The logical level could be replicated by using a laboratory computer to mount the image virtually or by writing the entire NAND flash memory image to another identical plug computer.

We attempted to use the OpenOCD software to write the NAND image to a second plug computer. The command `NAND write num filename offset` copies the NAND image `filename` to the NAND device `num` starting at `offset` in the NAND device. After the copy process is complete, the plug computer can be restarted and, in theory, should boot up normally as with any cloned device. Unfortunately, our attempts at using this method were unsuccessful.

The only successful method that we discovered for logical analysis is to mount an image as a read only device in a Linux environment using software tools that emulate NAND flash memory. The following steps are involved in mounting a JFFS2 file system:

- `mknod /temp/mtdblock 0 b 31 0`.
- `modprobe loop` (may not be necessary).
- `losetup -f` (returns a free loopback device, e.g., `/dev/loop0`).
- `modprobe mtdblock`.
- `modprobe block2mtd`.
- `echo /dev/loop0,128KB /sys/module/block2mtd/parameters /block2mtd`.
- `modprobe jffs2`.
- `mount -t jffs2 /tmp/mtdblock0` (mount point).

Note that this technique only works with the file system partition and not with the U-Boot partition or with the entire NAND flash memory image. If the entire NAND flash image is provided, the U-Boot partition, which usually constitutes the first few megabytes, must be carved out.

## 7. External Storage Considerations

Plug computers were originally designed to serve as network attached storage (NAS) servers, which require external storage media. The external storage could be an external hard drive, USB flash drive or, in the case of the SheevaPlug, a SDIO flash card. The external storage may be formatted with a common file system (e.g., FAT, EXT 2/3/4 or NTFS) or a less common file system (e.g., Minix, FUSE, HFS, HFS+, UFSD or VFAT) [4]. Plug computers also have the ability to format the external storage with most of these file systems, although this is typical of any Linux operating system.

The SheevaPlug and Pogoplug handle external storage differently. The SheevaPlug handles its external storage in the same way as any Linux machine – storage devices are mounted for access and use. The Pogoplug, on the other hand, automatically modifies external storage connected to it by adding its own system files. The files created are `/ceid` and `/.cedata/cedb` and the folder created is `.cedata`. File `.ceid` is a text file that contains a 22-character `diskid` while file `cedb` is an SQLite database file [7]. The database `cedb` contains an entry for every non-hidden file on the external storage; each entry contains the name, path, creation time and data type of the file. The creation time is in the UNIX timestamp format with three additional digits appended to it. Thus, the timestamp corresponding to the date `10 Oct 2010 09:08:07` is `1286701687XXX` with `XXX` as the three additional digits.

Figure 5 shows the `cedb` database as viewed using an SQLite manager. We observed that the `cedb` database contains entries for deleted files. When a file is removed using the Pogoplug software (e.g., via the website interface), the file entry is removed from the database. However, if the file is manually deleted from storage, regardless of whether or not the storage is connected to the Pogoplug or to another machine, the file entry remains in the database.

In general, the forensic analysis of external storage should require little or no special considerations because plug computers almost always use widely available Linux distributions (e.g., Debian or Redhat). However, investigators should be mindful of the circumstances under which external storage may be affected.

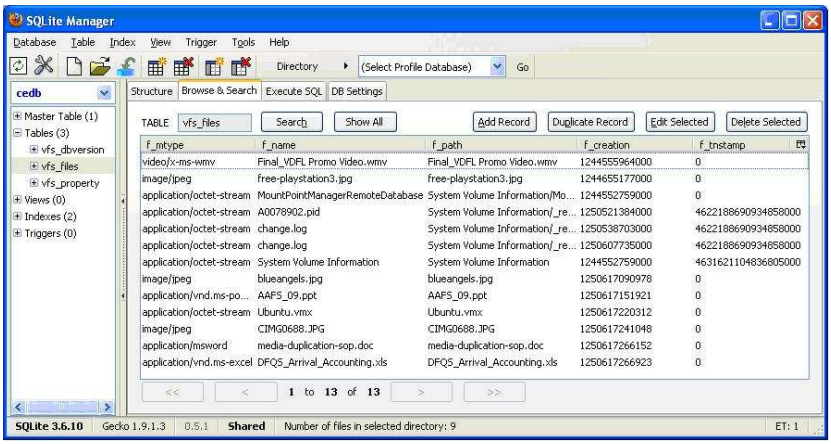


*Figure 5.* View of a `cedb` database.

## 8. Conclusions

Extracting and analyzing digital evidence in a forensically sound manner are becoming significantly more difficult for low form factor computers. Extracting data from a plug computer to create a forensic image is challenging but, nevertheless, possible. However, analyzing the image is difficult because the lack of automated tools necessitates manual analysis.

Second generation versions of the SheevaPlug and Pogoplug have already been announced, and many new plug computer models are in development. Meanwhile, manufacturers such as Seagate and Iomega have integrated plug computer concepts in their own product lines (e.g., FreeAgent from Seagate [18] and iConnect from Iomega [9]). The digital forensics research and vendor communities must intensify their efforts to keep up with this growing segment of low form factor computers, and develop forensically sound techniques and tools for extracting and analyzing digital evidence from these devices.

## References

[1] Amontec, Open On-Chip Debugger (OpenOCD), Vuippens, Switzerland (www.amontec.com/openocd/doc/index.html).

[2] M. Breeuwsma, M. de Jongh, C. Klaver, R. van der Knijff and M. Roeloffs, Forensic data recovery from flash memory, *Small Scale Digital Device Forensics Journal*, vol. 1(1), 2007.

[3] C. Brune, Lost art of computer programming, Das U-Boot: The universal boot loader (www.cucy.net/lacp/archives/000022.html), 2004.

[4] Cloud Engines, Pogoplug, San Francisco, California (pogoplug.com).

[5] DENX Software Engineering, Memory commands, DENX U-Boot and Linux Guide, Groebenzell, Germany (www.denx.de/wiki/view/DULG/UBootCmdGroupMemory#Section_5.9.2.4).

[6] DENX Software Engineering, U-Boot, Groebenzell, Germany (www.denx.de/wiki/U-Boot).

[7] Hwaci Applied Software Research, SQLite (sqlite.org).

[8] IEEE Standards Association, 1149.1-1990 – IEEE Standard Test Access Port and Boundary-Scan Architecture, Piscataway, New Jersey, 1990.

[9] Iomega, Iomega iConnect wireless data station, San Diego, California (go.iomega.com/en-us/products/network-storage-desktop/wireless-data-station/network-hard-drive-iconnect).

[10] Marvell Semiconductor, SheevaPlug Development Kit Reference Design, Santa Clara, California (www.plugcomputer.org/index.php/us/resources/downloads?func=startdown&id=90), 2010.

[11] plugcomputer.org, Plug Wiki (plugcomputer.org/plugwiki/index.php/Main_Page).

[12] Pogoplugged.com, Forums (www.pogoplugged.com/forums).

[13] Pogoplugged.com, How to find `mkfs`, `jffs` and other tools on Pogoplug (www.pogoplugged.com/forum/thread/11515/How-To-Find-mkfs-jffs2-and-other-tools-on-Pogoplug/?highlight=find+mkfs).

[14] D. Rath, Open On-Chip Debugger (openocd.berlios.de/web).

[15] D. Rath, OpenOCD User's Guide (openocd.berlios.de/doc/html/index.html#Top).

[16] J. Regan, The Forensic Potential of Flash Memory, Master's Thesis, Department of Computer Science, Naval Postgraduate School, Monterrey, California, 2009.

[17] M. Rosenbum and J. Ousterhout, The design and implementation of a log-structured file system, *Proceedings of the Thirteenth ACM Symposium on Operating System Principles*, pp. 1–15, 1991.

[18] Seagate Technology, FreeAgent DockStar, Scotts Valley, California (www.seagate.com/www/en-us/products/network_storage/freeagent_dockstar).

[19] Sun Microelectronics, Introduction to JTAG Boundary Scan, White Paper, Sun Microsystems, Santa Clara, California (www.johnloomis.org/ece446/notes/jtag/wpr-0018-01.pdf), 1997.

[20] UBIfs Wiki (osl.sed.hu/wiki/ubifs/index.php/Main_Page).

[21] D. Woodhouse, JFFS: The Journaling Flash File System (linux-mtd.infradead.org/∼dwmw2/jffs2.pdf).

[22] D. Woodhouse, UBI – Unsorted Block Images (www.linux-mtd.infradead.org/doc/ubi.html).