A visual approach to interpreting NAND flash memory

Dr Bradley L Schatz

Schatz Forensic, Brisbane AU
+61 422 949 039, bradley@schatzforensic.com.au
PO Box 15972 City East, Brisbane, 4002, AU.

ABSTRACT

The research described in this paper proposes methods for visually interpreting the content of raw NAND flash memory images into higher level visual artefacts of assistance in reverse engineering and interpreting flash storage formats. A novel method of reverse engineering the structure and layout of individual memory locations within NAND flash images, based on injecting a known signal into a test NAND environment is also proposed. Omissions in the current theory of operation of flash, in particular the role of flash memory controllers in transforming the raw NAND are identified, clarifying the cause of variations seen between images taken using pseudo physical and raw physical techniques. The effectiveness of the approach is validated against raw NAND images from YAFFS2 based Android phones, taken via JTAG and chip-off methods.

Keywords: Digital forensics, NAND flash, visualisation, reverse engineering, FTL

1. Introduction

The once widespread occurrence of standard interfaces to data storage in the computing field is rapidly eroding as embedded and mobile make up a greater proportion of devices investigated. At the foundations of this shift is the widespread adoption of NAND flash in place of the magnetic spinning disk. NAND storage may be found in applications ranging from mobile and embedded devices to USB flash drives.

NAND flash differs from traditional disk storage. While it is still a random access medium for reads, writing carries with it unique limitations, that, in the context of digital forensics, imply challenges related to diverse interfaces and heterogeneous storage formats, along with the promise of an increased ability to recover deleted information.

A range of tools exist for accessing NAND flash in a variety of formats, and the body of knowledge contains techniques for acquiring and analyzing NAND flash, however the field is a broad one, ranging from interpreting NAND chips extracted from USB drives, to interpreting images of portions of NAND acquired by software running on a smartphone. While the interpretation of flash-specific filesystems has recently received significant attention recently, particularly in regard to the YAFFS2 filesystem used on Android, no research to date addresses the variability observed between such software acquired NAND images with flash acquired directly via hardware methods.

This paper presents two advances on the state of the art in interpreting physical NAND flash images. First, a visual pattern based approach to hypothesizing about the structure and layout of the NAND image is proposed, such that variability might be accounted for in the interpretation process. Second, a method of reverse engineering the structure and layout of individual memory locations within NAND
flash images is proposed. The method is based on injecting a known and useful signal into test NAND-based environment for later interpretation.

2. Theory of operation – NAND flash

This section serves as an overview of NAND flash organization and architecture. The reader is referred to Breeuwsm et.al. (2007) for a more comprehensive overview of NAND flash and acquisition techniques.

The smallest writable unit in flash memory is the Page, which is conceptually subdivided into two parts, the User Data area, and Spare area (also referred to as the Out Of Band or OOB area). At the time of writing sizes of 2048 bytes for the user data area, and 64 bytes for the spare are common. The user data area is generally used for storage of content (for example logical blocks of a block device or filesystem) and the spare area used for storing Error Correction information (ECC) and metadata. ECC is generally observed at constant offsets within the Spare, and the offsets are particular to the software and hardware of the device.

Pages are organized into groups referred to as erase blocks (or simply, blocks). Prior to writing a page to a block, the entire block must be erased. Pages may generally only be written to a block sequentially (no random writes within the block). Writes to NAND flash are a scarce resource. Estimates are that for the current generation of device, a single storage area may only be written to (more correctly, erased and then written to) on the range of 100,000-1,000 times for current (Grupp et al 2012). In early USB flash drives, this limitation resulted in the rapid failure of the storage areas of flash drives most often updated, with file system metadata atimes often being observed as a culprit. This led to the introduction of Flash Translation Layers (FTL), a virtualization of the raw flash storage designed to balance writes across the available raw flash (wear levelling), while limiting the number of program/erase cycles of blocks.

A consequence of these layers is that successive writes to a particular storage area will result in writes to different raw storage areas in the raw NAND flash. The forensic significance of this is compelling: prior versions of a file system or files stored on flash may be recovered with an understanding of the FTL and access to the physical (raw) flash layer.

NAND memory may generally be accessed as raw NAND, or managed NAND. Raw NAND refers to direct access by the CPU of the host computer (GPIO bit banging), or indirectly from the CPU via a discrete flash memory controller. Flash memory controllers may serve a number of roles, ranging from implementing a full FTL to offload of ECC calculation.

Managed NAND generally sits behind a storage layer that mimics an existing block device (i.e. embedded behind microcontrollers in USB thumb drives, MMC camera cards, and SSD drives).

Related work

General tool support for forensic analysis of flash based storage is currently poorly recognized and limited in scope. The data recovery field offers combined hardware/software solutions focused on recovery of storage from USB flash drives and SSD’s. To the authors knowledge none of the products support recovery of prior versions of files from interpreting FTL, and the techniques underling recovery are not open to peer review. The situation is generally the same for the current generation of forensic tools focusing on mobile devices.
The consumption of raw NAND has been on the forensic research agenda for some time, with JTAG and so called “chip-off” approaches (physical chip removal followed by reading with a device programmer) proposed by Willassen (2005) as means for forensic acquisition of raw NAND flash. Breeuwsma (2006) described the theory of operation of JTAG and proposed a technique for identifying JTAG ports on mobile phones. Vidas (2011) proposed an alternative boot environment (akin to a PC LiveCD) based approach to acquisition of Android phone storage.

Unlike in the PC world, where data formats such as filesystems and volume management schemes change only slowly over time, in the embedded space, where NAND dominates, the data formats related to FTL and filesystem are comparatively heterogeneous. Accordingly early interpretation approaches focused on the use of carving (Willassen 2005). Breeuwsma et al (2007) describing a theory of operation for flash memory and presented a detailed overview of overarching challenges of raw flash acquisition and analysis. Techniques were proposed for forming hypotheses identifying which bytes of the spare are related to the FTL, and which bytes of the spare were the Logical Block of the page, and onwards interpreting those pages into a consistent virtual block device for subsequent filesystem analysis with standard tools. Luck and Stokes (2008) built on this work, describing a methodology for constructing prior versions of FAT filesystems, based on a hypothetical FTL approach which assumed that, where there are multiple versions of a page with the same Logical Block, newer versions are found at higher physical block addresses.

The DFRWS 2011 forensic challenge raised a limited awareness of the challenges associated with interpreting flash and stimulated research related to this area, in particular the interpretation of the YAFFS2 filesystem (Manning 2010) and its hybrid Filesystem/FTL. Bang et. al. (2011) observed in attempting to reconstruct the YAFFS2 filesystem that the structure of YAFFS2 FTL related metadata differed from previous studies. Quick and Alzabbi (2011) observed inconsistencies in their ability to interpret images of YAFFS2 partitions generated using differing techniques. Pooters et al (2011) identified a useful property of the YAFFS2 Object Header suitable for carving a subset of filesystem metadata. Schmidt (2011) analysed the YAFFS2 source code, describing many of the subtleties of the data structures involved.

Hoog (2012) and Schatz (2012), identified these inconsistencies as being due to variability in the location of YAFFS2 metadata (known as Packed Tags) stored in the spare section of the page. Both proposed a recovery technique for interpreting the filesystem and prior versions from the user data and packed tags. The approach of Schatz, a preliminary presentation of the research described in this paper, proposed identifying the tag within the spare by examining a variant of Conti’s (2010) byteplot and identified visual patterns as markers of tag data structures. Hoog proposed identifying the packed tag using a heuristic approach, and publicly released a YAFFS2 filesystem implementation as a part of the SleuthKit in 2013.

Outside the Android space, Sigwald (2012) described the theory of operation of raw NAND interpretation on iPhone 4 devices, including software local physical acquisition, FTL interpretation and decryption.

Despite the recent advances in interpreting YAFFS2 flash images, these works have focused on interpretation of what we call “pseudo-physical” NAND images: those acquired from user space via the MTD subsystem API which itself interfaces with either a NAND simulator or a specialized NAND/flash controller driver. Such acquisition is limited in regard to applicability (for example a locked phone may thwart the examiner from running the software necessary for an acquisition), completeness (not providing access to all areas of flash), and fidelity (for example, the MTD
subsystem has been observed to fail to present complete copies of the spare. No prior works have considered interpreting YAFFS2 flash from what we refer to as “raw physical” NAND flash: images taken at a hardware level rather than by software (for example either by JTAG or chip off).

The contribution of this paper is an investigation of the relationship between physical NAND flash and pseudo-physical NAND acquisition approaches, the outcomes of which identify important omissions in the current theory of operation of flash. These clarify the difference between images taken using both of these techniques. Novel methods of identifying the structure and organization of raw flash memory is proposed which facilitates the translation of arbitrary raw NAND flash into a normalized “cooked” version suitable for consumption by existing tools.

3. Method - The modelled byteplot

The fundamental hypothesis of this work is that by rendering a physical ("raw") or pseudo physical NAND image into a visual representation (a rendering), one might identify characteristics in the representation which assist in interpreting the upper layer file system components contained within the image.

We adopt the byteplot of Conti et.al. (2010), and propose extending the approach to visually discriminate between the structurally different areas identified by our theory of operation (the user data and the spare). The width of the rendering is based on the hypothesized (or known) size of the NAND page. Each byte of the NAND image is rendered as a greyscale value, such that an entire page will share the same horizontal line. The subsequent page is then rendered directly underneath the prior page’s rendering. Borders (rendered in red) are placed between the user data section and spare, as well as before the user data section and after the spare. We call this proposed rendering a “modelled byteplot”.

Table 1 depicts the layout of pixels from a NAND image of geometry (2048+64) i.e. user data of 2048 bytes and spare of 64 bytes.

<table>
<thead>
<tr>
<th>User data bytes</th>
<th>Spare bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ... 2047</td>
<td>2048 ... 2111</td>
</tr>
<tr>
<td>2112 ... 4159</td>
<td>4160 ... 4224</td>
</tr>
<tr>
<td>4224 ...</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Byteplot rendering of model

An example of the proposed modelled byteplot, applied to the DFRWS 2011 challenge, case 2, under a hypothetical NAND geometry of (2048+64) is presented as Figure 1. This particular image has been chosen to function as a control due to the relative degree of scrutiny that has been applied to the image.

Figure 1: Portion of modelled byteplot for DFRWS 2011 challenge, image 2

YAFFS2 filesystem metadata in the page

Examination of the above modelled byteplot indicates an obvious pattern on the horizontal plane within the user data section: short mostly wholly black lines followed by long, mostly unbroken white lines. Closer examination yields the white sections are wholly 0xFF values, and the solid black
sections wholly 0x00 values. An examination of the carving criteria of Pooters et. al. (2011) indicated that the YAFFS2 Object Header is stored in the user data section, with a null padded filename of up to 256 bytes size near the beginning, and bytes 512-until the end of the user data segment padded with 0xFF bytes.

Based on the above one may hypothesize that the aforementioned visual artefact (which we refer to as Object Header Striations) correspond to Object Headers. A manual examination of a small set of the data corresponding to these were verified as consistent with containing Object Header data.

Examination of the byteplot additionally indicated that there was a clear delineation between user data and the spare. Object header striations terminated exactly at the user data/spare boundary, as did all other user data.

**Error correction visual artefacts**

Returning to the theory of operation in regard to the presence and placement of ECC information, we form the following sub hypothesis: ECC information stored in the spare will be visible as high entropy data and will be readily distinguishable from what we hypothesise will be relatively low entropy metadata belonging to the filesystem or FTL stored in the spare.

In order to test this hypothesis, three sets of modelled byteplots for the DFRWS 2011 Challenge Case 2, we generated based on page size hypotheses of (512+16), (1024+32), and (2048+64). Enlarged portions of the spare from these are presented in Table 2.

<table>
<thead>
<tr>
<th>Page (512+16)</th>
<th>Page (1024+32)</th>
<th>Page (2048+64)</th>
</tr>
</thead>
</table>

*Table 2: DFRWS 2011 Challenge Case 2 under differing spare hypotheses*

Examining the renderings under these various page hypotheses, one can clearly identify, under the (2048+64) hypothesis, a vertical band of high entropy data in the spare (between the two solid vertical white bands) and a band of low entropy data (the largely black vertical band). A wider examination of the remainder of the rendering indicated a consistent pattern these bands. In comparison, the (512+16) and (1024+32) hypotheses are not consistent with the entropy hypothesis.

The experiment with a Droid-Eris image was repeated based on page hypotheses of (512+16), (1024+32), and (2048+64). Enlarged portions of the spare are presented as Table 3.

---

Examining the renderings under these various page hypotheses, one can identify clearly two vertical bands of high entropy data in the spare under the (2048+64) hypothesis along with two low entropy bands. Similar to the aforementioned image, a wider examination of the remainder of the rendering indicated a consistent pattern these bands and in comparison, the (512+16) and (1024+32) hypotheses are not consistent with the entropy hypothesis.

**YAFFS2 Filesystem/FTL visual artefacts**

A closer examination of the low entropy bands above, under the (2048+64) page size hypothesis, indicated visual artefacts which appeared as vertical lines that start off dim and become incrementally brighter downwards along the y axis. An examination of a subset of these indicated that the upper beginning of the line started with byte value 0x1, and incremented by 1 each page (line). Based on the assumption that the filesystems in the images would contain a significant number of large files, and that those files would, early in the life of the device, generally be laid out sequentially within flash blocks, we formed the hypothesis that these visual artefacts be related to the logical addresses of each page, similar to the Logical Block Numbers of Breeuwsma et al (2007) and Logical Sector Numbers observed by Luck & Stokes (2008).

An examination of the YAFFS2 yaffs_packed_tags2_tags_only data structure yielded an addressing related field that applies to each file: the Chunk ID. For a particular file (or Object in YAFFS terminology) Pages subsequent to Object headers are addressed from Chunk ID 1 and onwards. The structure is summarized in Table 4.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Width</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>u32 (Little Endian)</td>
<td>seq_number</td>
<td>Constant for all object pages in the same block. Incremented for new versions of a page.</td>
</tr>
<tr>
<td>4</td>
<td>u32 (Little Endian)</td>
<td>obj_id</td>
<td>Constant for all pages related to a particular object (used for other purposes in Object Header page).</td>
</tr>
<tr>
<td>8</td>
<td>u32 (Little Endian)</td>
<td>chunk_id</td>
<td>For data chunks the address, for Object headers extended tags</td>
</tr>
<tr>
<td>12</td>
<td>u32 (Little Endian)</td>
<td>n_bytes</td>
<td>Many uses, generally sizeof(user data) in pages corresponding to large files.</td>
</tr>
</tbody>
</table>

**Table 4: the yaffs_packed_tags2_tags_only data structure**

A further reading of the YAFFS2 documentation (Manning 2010), and Schmidt et.al. (2011), yielded the following hypotheses regarding the bytes surrounding the vertical brightening line, for the length
of the line, under the assumption that the artefacts are related to a whole file written to the filesystem in one transaction:

1. An object header striation will occur on the page immediately prior to the set of pages corresponding to the vertical brightening line; and
2. The three vertical pixel lines to the right of the brightening line should locally be constant (being the three higher valued endian bytes of the Chunk ID); and
3. The 4 vertical pixel lines immediately to the left of the brightening line are the Object ID field and should be constant for the length of the vertical brightening line (with the exception of the value corresponding to the 0-valued beginning of the brightening line; and
4. The next 4 bytes to the left again of the hypothesized Object ID area are the Sequence Number and should remain constant for the length of the block.

The above hypotheses are general in nature and will only hold to the extent that the file pages are contiguous and have not bridged to a new flash block.

In order to test these hypotheses a candidate vertical band of low entropy data matching the above criteria was visually identified and the number of pixels to the start of the hypothesized yaffs_packed_tags2_tags_only data structure measured. This offset was used as the packed tags location offset within the spare, and using a self-developed implementation of YAFFS2, the filesystem interpreted. The file structure and content were found to be self-consistent.

Based on the above hypotheses and experimental confirmations, we define the following lemmas:

**Lemma 1:** Object header striations continue all the way to the user data/spare boundary.

**Lemma 2:** Object header striations are constant 0xFF (white) from offset 512 until the border of the spare.

**Lemma 3:** There is a clear delineation between user data and the spare at the boundary.

**Lemma 4:** ECC is clearly identifiable within the spare as high entropy bands.

**Lemma 5:** Vertical brightening lines in a low entropy bands in the spare are consistent with the low endian byte of the Chunk ID of contiguous pages of a file.

**Lemma 6:** Packed tags are only found within the spare.

### 5. Evaluation – interpretation of raw NAND

Having identified the lemmas of the prior experiments, the following experiment was undertaken to verify whether the lemmas hold for physically acquired NAND images obtained from a HTC Desire mobile phone using the following techniques:

- **JTAG:** the phone is dismantled, JTAG test pads identified, and a JTAG flasher box (RIFF Box) is attached via a jig. Software corresponding to the JTAG box (JTAG Manager) is then used to dump the NAND content via the JTAG interface.
- **Chip off:** the phone is dismantled, and the NAND flash identified (KA100O015M-AJTT), removed, cleaned, and re-balled. The flash chip is then read using a universal programmer (UP-828).
Using only the information in the dump and the proposed technique, it is possible to make effective hypotheses regarding which ranges contain YAFFS2 filesystems and which do not, based on the presence of object header striations. However, no visual artefacts were identified which effectively distinguish between one YAFFS2 filesystem and more than one YAFFS2 filesystems adjacent to each other.

Our investigations suggest that currently, the most straightforward way to get the MTD flash partitioning is from the kernel log, obtained via dmesg:

```
<5>[ 10.258911] Creating 6 MTD partitions on "msm_nand":
<5>[ 10.263946] 0x00001ff60000-0x000020000000 : "misc"
<5>[ 10.270080] 0x000004240000-0x000004740000 : "recovery"
<5>[ 10.279846] 0x000004740000-0x0000049c0000 : "boot"
<5>[ 10.283508] 0x0000049c0000-0x0000143c0000 : "system"
<5>[ 10.556365] 0x0000143c0000-0x000016bc0000 : "cache"
<5>[ 10.600402] 0x000016bc0000-0x00001ff60000 : "userdata"
```

It remains an open question whether this MTD debug output is widely produced by Android YAFFS2 implementations, and whether this is a generally applicable technique.

The JTAG Manager software indicated that the spare data was to be saved at the end of the image, which implies that all of the user data is stored concatenated followed by all of the spare data concatenated (end spare images). Nanddump on the other hand produces images which interleave the user data with the spare (inline spare images). This former format has the benefit of supporting existing carving applications without modification.

The JTAG acquired image was rendered using the modeled byteplot technique formerly described. No visual artefacts matching the former lemmas were identified. A variant of the image was created, reorganizing the spares to the inline format formerly described and rendered again. Table 5 contains the two renderings. Object header striations are clearly visible in the normalized version.

![JTAG image with (2048+64,end spare) hypothesis](image1)

![Normalized JTAG image (2048+64, inline spare)](image2)

*Table 5: Visual differences between inband spare and spare-at-end raw images*

A closer examination of the normalized JTAG image (an excerpt is presented as Figure 2) revealed that the following lemmas do not hold:

- The object header striations terminate prior to the user data/spare boundary (Lemma 1)
- Vertical brightening lines are visible outside the spare. (Lemma 6)
An examination of the chip off dump of the NAND flash, a portion of which is presented in Figure 3, similarly presented a situation where the aforementioned lemmas do not hold:

- The object header striations extend over the data/spare boundary intruding into the spare, and are regularly interrupted by high entropy signal. (Lemma 2)

Additionally, regular evenly spaced vertical white lines are visible in ever user data segment, which is inconsistent with the observed modelled byteplots generated from pseudo-physical images.

Using a bitmap editor, the widths of the white vertical artefacts were measured, and based on the hypothesis that these artefacts are not user data, were shifted to the spare. The resulting transformed image was then inspected and the lemmas appeared to hold. A packed tags location hypothesis was formed on this basis and the transformed image interpreted using the YAFFS2 file system implementation described earlier and found to be internally consistent.
Based on the kernel dmesg output, we identified the msm_nand.c driver, which, based on an examination of the relevant MTD kernel source file\(^2\), to be related to:

“on-chip NAND flash controller driver on Qualcomm's MSM and QSD platforms.”

The CPU of the phone is a Qualcomm QSD8250.

The following lines of code in the driver suggest that the flash controller is configured to place the bad block marker at offset 465 within each 516 byte page, with pages per 2k NAND. We suspect that the term “page” here is being used in an overloaded manner, and have not pursued this further.

\[
\begin{align*}
(3 << 6) & \quad /* 4 codeword per page for 2k nand */ \\
| (516 << 9) & \quad /* 516 user data bytes */ \\
| (465 << 6) & \quad /* Bad block marker location */ \\
| (0 << 16) & \quad /* Bad block in user data area */ 
\end{align*}
\]

The above evidence suggests an inbuilt flash controller as the cause of the relocation inconsistencies in the chip-off image.

6. Method – Identifying variability in arbitrary raw NAND

The prior experiments identified a set of lemmas regarding the visual artefacts associated with the YAFFS2 filesystem which assist in:

a) Distinguishing between an in-band spare image and end spare image; and

b) Identify a consistent page size model; and

c) Identifying relocations.

However, in absence of a YAFFS2 filesystem within the image, the majority of these lemmas do not hold. General means of identifying the above are required. We propose placing a carefully crafted known signal into an unknown NAND flash geometry in order to assist in identifying these characteristics, both by visual inspection using the modelled byteplot and by automated means.

Identification of page size and image format

The signal we inject is a set of specifically designed files (which we dub “tracers”), such that when stored directly as a file on the a flash device and the file pages contiguously laid out on flash, the subsequent rendering of a modelled byteplot will yield a readily recognizable depiction indicating the corresponding user data size, when modelled with the correct page size theory.

Table 6 summarises the tracer files. Each file is based on a grid pattern in the background composed of 32x32 pixel squares of alternatively full white (0xFF) and grey (0xB2). The numerals are black (0x00). The files were produced as 8-bit greyscale images using a bitmap editor, exported as a non-runlength encoded Sun Rasterfile, and then the header removed using a hex editor.

<table>
<thead>
<tr>
<th>Name</th>
<th>512 tracer</th>
<th>1024 tracer</th>
<th>2048 tracer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Targeted page size</td>
<td>(512+16) Page size</td>
<td>(1024+32) Page size</td>
<td>(2048+64) Page size</td>
</tr>
</tbody>
</table>

\(^2\)/drivers/mtd/devices/Kconfig
Table 6: Summary of proposed tracers

<table>
<thead>
<tr>
<th>Appearance</th>
<th>512</th>
<th>1024</th>
<th>2048</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions and size</td>
<td>512x512 grid</td>
<td>1024x1024 grid</td>
<td>2048x2048 grid</td>
</tr>
<tr>
<td>256K size</td>
<td>1024K size</td>
<td>4M size</td>
<td></td>
</tr>
</tbody>
</table>

Table 7 depicts the modelled byteplots associated with the tracers when placed into an emulated NAND (nandsim) with page size (512+16). Only the 512 tracer appears consistent in the rendering, both in terms of grid dimensions, and appearance of numerals.

Table 7: Tracer renderings under various page hypotheses against a (512+16) original page size

<table>
<thead>
<tr>
<th>Tracer rendering</th>
<th>24</th>
<th>24 10</th>
<th>24 10 24 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page hypothesis</td>
<td>512+12</td>
<td>1024+32</td>
<td>2048+64</td>
</tr>
</tbody>
</table>

Automated identification of relocations

The tracers are designed such that, in the presence of simple relocations of vertical byte columns, as has been described formerly, those relocations should be readily perceivable as deformations of the grid and discontinuities in the curves of the numerals.

While such directly perceivable artefacts are immediately useful in identifying the presence of relocations, there is insufficient information in the signal to effectively identifying what has been moved and where. For this reason the first two horizontal lines of the tracers (which we refer to as the “ruler”) are designed not for far scale visibility, but for both rapidly locating the tracer and identifying individual columns. Taking the upper pixel and the adjacent lower pixel, $y_0$ and $y_1$, the byte values of these pixels are encoded such that the decoded pair yields the original tracer $x$ offset of the pixel in the user data section of the page.
The exceptions to the above are for bytes $x_0$-$x_{255}$ of the first row of the tracer, which are treated as if they contained 0, while they may otherwise be used for storing a signature. The signature is used to rapidly perform a bytwise search for the tracer in a NAND image. In the case of these tracers, the unique signature is of the form http://schatzforensic.com/512-grid-512-label.raw and are listed in Table 6. By searching for the signature and then interpreting the relevant two lines (pages) of associated data, that one is able to identify, for the user data content, at what offsets in the page the values are actually stored.

### 7. Evaluation – Identifying variability in a JTAG NAND image

In order to evaluate the proposed approach, a different HTC Desire phone, which has a known (2048+64) page size, was pre-prepared with the three tracer files, by placing them in the /data/local/tmp/ folder using the Android Debug Bridge. No rooting was required. The phone was then dismantled and a JTAG image taken per the prior experiment.

We predict that the 2048 tracer should appear largely unaltered in the modelled byteplot under a (2046+64, end spare) page hypothesis, and that any relocations should result in visual deformations to the rendering.

Three sets of normalised modelled byteplots were generated, covering page hypotheses of (2048+64, end spare), (1024+32, end spare) and (512+16, end spare). Figure 4 presents an extract under the (2048+64, end spare) page hypothesis, with the 2048 tracer clearly distinguishable in the modelled byteplot. The 2048 tracer appears consistent in the rendering, both in terms of grid dimensions, and appearance of numerals. However the 1024 and 512 tracers are inconsistent with their original and intended appearance, being compressed on their y axis and repeating their numerals.

![Figure 4: Modelled byteplot of JTAG containing tracer files](image)

### Table 8: Tracer byte relocation identifiers

<table>
<thead>
<tr>
<th>Pixel offset</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>..</th>
<th>255</th>
<th>256</th>
<th>257</th>
<th>258</th>
<th>..</th>
<th>511</th>
<th>512</th>
<th>513</th>
<th>..</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_0$ value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>..</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>..</td>
</tr>
<tr>
<td>$y_1$ value</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>..</td>
<td>255</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>..</td>
<td>255</td>
<td>0</td>
<td>1</td>
<td>..</td>
</tr>
</tbody>
</table>

Original tracer offset ($y_0 << 8) + $y_1$

<table>
<thead>
<tr>
<th>Original tracer offset</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>244</th>
<th>256</th>
<th>257</th>
<th>258</th>
<th>511</th>
<th>512</th>
<th>513</th>
<th>..</th>
</tr>
</thead>
</table>

Renderings under (1024+32, end spare) and (512+16, end spare) page hypotheses do not produce any visually recognizable tracers.

The portion of the rendering corresponding to the 2048 tracer was examined for deviations from the original signal. An extract of this is presented as Figure 5. Two classes of deviation were identified as expected; deformations of the grid and discontinuities in the curves. Specifically, the curve of the left hand edge of the 8 is discontinuous, and the vertical grid behind the left hand side of the 4 is compressed (it no longer appears square).

Figure 5: Relocations of data manifest as deformations and discontinuities in the tracer

In order to test the effectiveness of the ruler portion of the tracer, we built a proof of concept tool that automatically searches for the ruler signature, and when identified, then interprets the corresponding original tracer offset values under the associated page hypothesis. We ran the tool over the JTAG image under the aforementioned page hypothesis.

With reference to Table 9, which reproduces a portion of the output of the tool, starting from offset 0 are stored original ruler values 0-511, and beginning at offset 512, values 516-1027. We interpret this to be that there is a discontinuity of 4 values (512-515) which have been otherwise relocated. These values may be found at offset 2048, which suggests that they have been relocated to the spare.

<table>
<thead>
<tr>
<th>Offset 0: 0 - 511 (512)</th>
<th>Offset 2041: 1028 (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset 512: 516 - 1027 (512)</td>
<td>Offset 2042: 0 (1)</td>
</tr>
<tr>
<td>Offset 1024: 1032 - 1543 (512)</td>
<td>Offset 2043: 0 (1)</td>
</tr>
<tr>
<td>Offset 1536: 1548 - 2047 (500)</td>
<td>Offset 2044: 258 (1)</td>
</tr>
<tr>
<td>Offset 2036: 17733 (1)</td>
<td>Offset 2045: 0 (1)</td>
</tr>
<tr>
<td>Offset 2037: 5911 (1)</td>
<td>Offset 2046: 0 (1)</td>
</tr>
<tr>
<td>Offset 2038: 0 (1)</td>
<td>Offset 2047: 0 (1)</td>
</tr>
<tr>
<td>Offset 2039: 0 (1)</td>
<td>Offset 2048: 512 - 515 (4)</td>
</tr>
<tr>
<td>Offset 2040: -6169 (1)</td>
<td>Offset 2052: 21465 (1)</td>
</tr>
</tbody>
</table>

Table 9: Abridged original tracer offset values

This technique effectively identifies relocations of the user data portion of the page, however, given that the related spare bytes are not controlled by the user, the locations of the remaining columns in the spare is still an open question. Figure 6 depicts the first attempted normalized version of the image taking into account the above relocations and simply shifting the remaining misplaced columns to the spare.

In this instance we hypothesized, based on the vertical address blooms and low entropy bands, that the packed tags were located at spare offset 0, however the size of the low entropy band was only 12
bytes wide, rather than the expected 16 bytes. We hypothesized that the remaining 4 bytes between the two rightmost high entropy bands (the 4 rightmost bytes).

![Image of byteplot](image)

*Figure 6: Portion of modelled byteplot of partially normalised relocations*

We tested this relocation hypothesis by generating a normalized inband NAND partition image under the hypothesis and accessed the image using the Sleuthkit. We were able to interpret the filesystem without error, and extract the original tracer using the icat command. The hash of the extracted tracer was compared with that of the original and found to be the same.

8. Limitations & future work

While the proposed techniques and artefacts in this work have been tested and found to be valid under the methodology applied, the testing has been limited in scope to primarily two phones of the same model, and more testing is warranted, across a wider range of YAFFS2 based subject devices and acquisition tools (JTAG tools and universal programmers). Similarly, while we have relied on internal consistency as a measure success in our testing, a more complete verification would involve comparing the hashes of the user data section between images taken using varying methods.

The general relevance of raw NAND flash in the Android space appears to be on the decline, due to an apparent shift to using managed eMMC based flash. Such storage hides the NAND flash behind an MMC based block storage abstraction and microcontroller, leaving achieving direct access to the uncooked NAND a less straight forward proposition. However, despite this trend, raw NAND continues to be observed in low-end phones and tablets running even recent (4.0+) versions of Android.

In the wider context, raw NAND remains prevalent in the embedded and mobile space, for example in USB storage, SSD’s and smart phones such as the iPhone 4 and 5. Unaddressed challenges to this approach inherent in this space include XOR based encodings and encryption being used to increase the reliability of high density NAND.

In future work we intend to apply the techniques identified towards reverse engineering the FTL used in other phones and USB flash drives utilizing raw NAND, and to explore methods of gaining access to the raw NAND found in managed NAND devices.
9. Conclusions

This paper presented two advances on the state of the art in interpreting physical NAND flash images. A visual pattern based approach to hypothesizing about the structure, internal features and layout of a physical NAND image was proposed, such that variability might be accounted for in the interpretation process. A novel method of reverse engineering the structure and layout of individual memory locations within NAND flash images, based on injecting a known signal into a test NAND environment was also proposed. Omissions in the current theory of operation of flash, in particular the role of flash memory controllers in transforming the raw NAND were identified, clarifying the cause of variations seen between images taken using pseudo physical and raw physical techniques.

While the proposed techniques were only evaluated against a small sample of raw NAND images related to the YAFFS2 filesystem, we believe that the techniques may be of wider applicability in interpreting raw NAND images from other systems.

The code for generating the modelled byteplots and the tracer images will be published online via http://schatzforensic.com/nand/.

10. References


