Forensic Limbo: Towards Subverting Hard Disk Firmware Bootkits

Michael Gruhn*

Department of Computer Science, Friedrich-Alexander University of Erlangen-Nuremberg, Martensstr. 3, 91058 Erlangen, Germany

Abstract

We discuss the problem posed by malicious hard disk firmware towards forensic data acquisition. To this end, we analyzed the Western Digital WD3200AAKX model series (16 different drives) in depth and outline methods for detection and subversion of current state of the art bootkits possibly located in these particular hard disks' EEPROMs. We further extend our analysis to a total of 23 different hard drive models (16 HDDs and 7 SSDs) from 10 different vendors and provide a theoretical discussion on how hard disk rootkits residing in the firmware overlays and/or modules stored in the special storage area on a HDD called the Service Area could be detected. To this end, we outline the various debug interfacing possibilities of the various hard disk drives and how they can be used to perform a live analysis of the hard disk controller, such as dumping its memory over JTAG or UART, or how to access the Service Area via vendor specific commands over SATA.

Keywords: anti-forensics; hard disk firmware rootkit; hard disk firmware bootkit

1. Introduction

In digital forensics data is analyzed. In order to analyze data, it must, however, first be acquired. Because many times forensic investigations, of, e.g., industrial espionage, involve rootkit compromises, this paper addresses persistent data acquisition from potentially rootkit subverted hard drives. This is a task that has not been addressed by the current state of the art in forensic hard drive forensics. Current literature on hard drive forensics always recommends making a copy of the original source drive, while using a write blocker to prevent changes to the original source drive. The current state of the art also recommends the acquisition of the usually hidden HPA and DCO sections. For a non-compromised hard drive, those measures work fine. However, they are not enough to ensure that evidence is not destroyed, lost or never found when the analyzed hard drive has been compromised by a firmware rootkit.

Hence, in this paper, we first give a short overview of what effect a firmware rootkit can have on an investigation. We then demonstrate how firmware bootkits can be detected and we outline several possibilities how even deep firmware rootkits can be detected. We will use a Western Digital WD3200AAKX as running example throughout this paper.

1.1. HDD anatomy

A modern HDD is not just a block device but rather a whole computer system in itself. The symbolic picture in Figure 1 on the following page shows the anatomy of a hard disk drive (HDD). It is loosely based on the Wester Digital WD3200AAKX drive. The picture shows the HDD and selected components in the middle. The components are: the disk, also known as the platter; the read and write head; and the PCB containing the processor, RAM, EEPROM, etc. The figure further points out three topics of interest: persistent storage areas, interfacing possibilities, and anti-forensic threats. These are detailed in the next two sections. The first section outlines the persistent storage areas and their associated anti-forensic threats. The second section outlines the interfacing and verification possibilities.

1.1.1. Persistent storage areas and anti-forensic threats

While the main purpose of a HDD is to provide persistent storage, the HDD itself has also storage areas it can use, which may not be accessible by a normal user of the HDD. The following areas are pointed out in Figure 1 on the next page:

Mask ROM. The mask ROM is integrated into the processor on the HDD's PCB. Because it is programmed by the integrated circuit manufacturer during the manufacturing process directly via the photo*mask* in the photolithography process, it is read-only and can not be modified. It contains code that allows the processor to boot and load the firmware boot loader code from the EEPROM into RAM.

EEPROM. The EEPROM contains the boot loader of the firmware. The boot loader bootstraps the system to the point where it can read from the platter. At this point, it will start loading more firmware from the platter. The

*Corresponding author
Email address: michael.gruhn@cs.fau.de (Michael Gruhn)

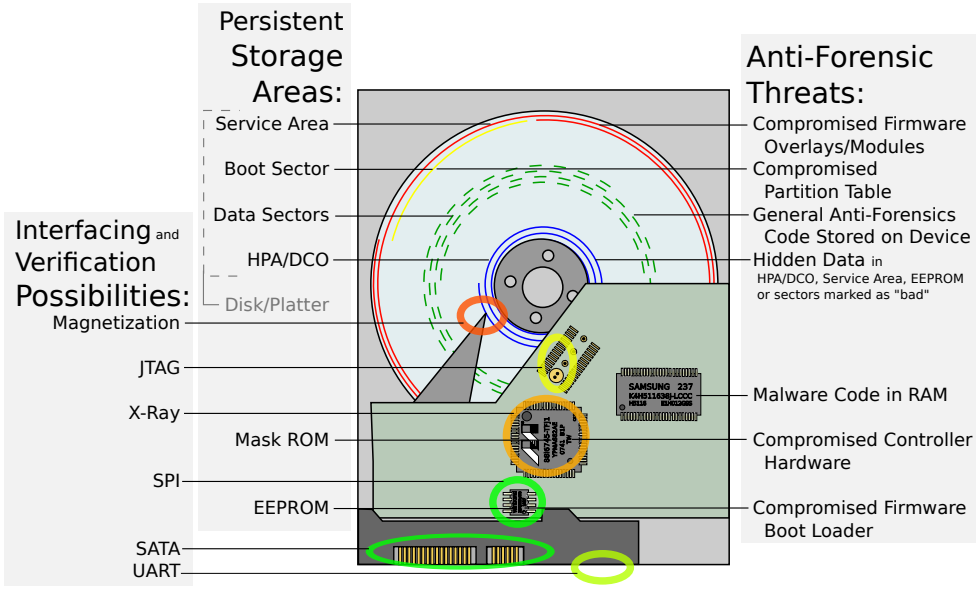


Figure 1: HDD anatomy: storage areas, interfacing possibilities and anti-forensic threats

EEPROM can also be read and written by software via the HDDs SATA connection. This way firmware code can be modified, e.g., to perform legitimate firmware upgrades. This results in the anti-forensic threats associated with a compromised firmware boot loader, also known as a bootkit. It is important to note that this firmware bootkit does not interfere with the boot process of the operating system installed on the HDD, but rather the boot process of the HDD itself. Besides compromising the firmware, the EEPROM could be used as hidden data storage.

Service Area. The Service Area is a hidden area on the platter that, unlike the HPA or DCO, can not be used by a normal user. This area is used to store further firmware components called overlays or modules, which are loaded into RAM by the boot loader. On some HDD's some overlays are only loaded on demand and swap other overlays out of RAM. While this area is reserved for usage by the firmware, there are vendor specific commands (VSCs) with which this area can be accessed via the SATA interface. This allows an attacker to modify firmware as well as hide data. After the Service Area begins the regular storage area of the HDD platter — often known as the User Area. However, in Figure 1 we additionally point out different aspects of the User Area, because these different sections of the User Area can be used in different ways to facilitate anti-forensics. But to the hard drive, the User Area is treated as one single storage area.

Boot Sector. The Boot Sector is the first thing loaded by a BIOS when booting from the HDD. Because a professional forensic investigator will not boot from the device, any malware infections, such as bootkits in the boot sector, will not immediately impact his work, unless, of course, the malware is the subject of his investigation.

The Boot Sector usually also contains the partition table, which can be malicious. In CVE-2016-5011, we have shown that it is possible to cause a denial of service (DoS) against various Linux systems with a specially crafted DOS/MBR partition table, that uses an extend partition loop [20], i.e., an extended partition within the DOS/MBR partition table will point back to the DOS/MBR itself causing infinite recursion in the library libblkid when parsing the partition table. Another such example uses GPT partitioning. It can cause a DoS against Windows 7 by setting the number of partition table entries to zero in the GPT header, which will result in the so-called Blue Screen of Death due to an implementation error which causes a division by zero within the kernel. So even by merely connecting a HDD with such a compromised partition table to a vulnerable system can impact the forensic investigator's work negatively.

Data Sectors. The Data Sectors are all sectors that a regular user of the HDD has access to. This is where, e.g., the file system resides, the files within it, etc. Regular anti-forensic measures, typically, reside here, e.g., directory loop attacks [20] or XSS code injection into forensic reports that are generated as HTML files [20].

HPA/DCO. Last but not least the HDD can contain the so-called Host Protected Area (HPA) and/or a Device Configuration Overlay (DCO). Both are hidden from the user. The HPA can be used to store installation files used for system recovery, while the DCO can be used to control over-provisioning of the HDD. Access to both the HPA and DCO can be acquired via ATA commands [7]. Both areas are well known to forensic investigators, hence, they lost their value as data hiding spot for criminals.

1.1.2. Interfacing and Verification Possibilities

Figure 1 on the previous page also shows the various interfacing and verification possibilities we found provided by the analyzed hard drives. We will outline most of them in more detail throughout this paper. Hence, here we only give a brief outline of each.

Magnetization. The first idea to verify the data on the HDD platter seems to be to read the magnetization directly from the platter. While this may have worked on older hard disk drives, newer disk drives typically have a very high integration density rendering this method nearly impossible.

SATA. The usual way today to interface with a HDD is via its Serial AT Attachment (SATA) interface. This method uses the firmware stored on the device to read and write data from and to the platter. In the case of a firmware compromise this interface can not be trusted.

SPI. To interface with the EEPROM the Serial Peripheral Interface (SPI) can be used. Because this interface it very low level a software compromise is not possible.

X-Ray. While x-raying the device is not actually interfacing with it, X-raying can be used to verify the integrity of the PCB and attached microchips.

JTAG. Some hard disks have a Joint Test Action Group (JTAG) interface. A JTAG interface can be used to test and debug the processor on the PCB.

UART. Some hard disks have a Universal Asynchronous Receiver Transmitter (UART) interface, which provides serial communication with the device.

1.2. HDD firmware analysis software

PC-3000 is a commercially available hard and software combination[1], which can be used to repair corrupted firmware. Unfortunately, we do not have access to a PC-3000, hence, we can only make assumptions about it. Presumably, it could possibly be used to restore a proper firmware on a HDD. But it is a proprietary product, hence, little is known about its inner workings, making it a less trustworthy solution than what we are about to present. Also, because the PC-3000's inner workings are unknown, it can not be further developed and adapted to the needs of forensic examiners, e.g., new hard drive support.

1.3. Related work

In 2009 Sutherland et al. [18] outlined an anti-forensic technique in which the ability of the hard disk to mark specific tracks on the hard drive platter as bad was used to hide data from a forensic investigation by marking its corresponding physical sectors as bad, i.e., unusable. At OHM in 2013 Jeroen "Sprite_tm" Domburg's demonstrated possibly the first publicly demonstrated hard disk firmware rootkit [4]. Many other works followed [22, 6, 21]. In 2012 Knauer and Baier [10] demonstrated how the diagnostic UART interface of a Samsung HDD could be used to bypass ATA passwords. They further extended their work in 2014 [1] to anti-forensic data hiding. In 2013 Read et al. [14] used firmware modifications to hide partitions from a forensic investigator. To this end, they used the HDDHackr tool, which can manipulate firmware of ordinary Western Digital HDDs to be used with the Microsoft Xbox 360, instead of more expensive official OEM drives. In 2015 Laan and Dijkhuizen [11] proposed a firewall for specific ATA commands. With their work in place, the proposed hard disk firmware rootkits would not be able to infect the device, because this is done via specific ATA commands, which their so-called *firmwall* would block.

Further the NSA's IRATEMONK, UNITEDRAKE, STRAITBIZARRE, SLICKERVICAR, SADDLEBACK, ALTEREDCARBON, FAKEDOUBT, PLUCKHAGEN and EASYKRAKEN projects[2] are assumed to be in one way or the other involved with compromising the firmware of HDDs. In 2015 Kaspersky Labs GReAT (Global Research & Analysis Team) discovered the `nls_933w.dll` malware module which is able to reprogram HDD firmware of all major vendors [9]. This malware is believed to be part of the above mentioned NSA tool set. The samples in question have been compiled in 2010 and 2013 meaning the malware was globally active for at least 5 years prior to discovery. This further highlights the lack of forensic capabilities with regard to HDD firmware malware.

And, even though the topic of firmware rootkits and malware has been demonstrated again and again, there is no publication outlining the impacts on digital forensics and/or providing best practice recommendations on how to deal with such firmware anti-forensic malware infections.

1.4. Outline

The outline of this chapter is as follows:

First, in Section 2 on the following page, we examine hard disk firmware bootkits. Next, in Section 3 on page 11 we outline overlay-based hard disk firmware rootkits. Followed by a discussion in Section 4 on page 14 and last but not least a conclusion in Section 5 on page 15.

[1] `http://www.acelaboratory.com/pc3000portable.php`

[2] `https://cryptome.org/2014/01/nsa-codenames.htm`

2. Hard disk firmware bootkit

In this section, we present a survey of hard disk firmware bootkit infections using the Western Digital WD3200AAKX 320 GB HDD as our primary example. In the first Section 2.1, we show two ways how this HDD's firmware can be compromised — only via running software with root privileges on the computer hosting the HDD. Next, we show how we can use only anti-forensic resistant hardware techniques to detect some of the possible firmware manipulations. We further detail how deeper firmware manipulations could be detected as well as briefly outline how those techniques can be applied to other hard drives besides the explanatory WD3200AAKX, namely 23 hard drives by different manufacturers. In Section 2.4 on page 11, we show how manipulated firmware and also manipulated controller hardware can be subverted by a forensic analyst to still access the data stored on the HDD's platter in a forensically sound way. Eventually to conclude this section we give future prospects in Section 2.5 on page 11 into how firmware manipulations can be investigated further.

2.1. Compromising

The firmware of the Western Digital WD3200AAKX is not digitally signed. Neither is the firmware of any other hard drive, to our knowledge. It can be uploaded, i.e., replaced with a new firmware image, via vendor specific ATA commands. These commands are often simply called vendor specific commands (VSCs). The Linux tool `hdparm` provides the argument "`--fwdownload <firmware.bin>`" which implements a convenient interface to these vendor specific commands. It can be used to write the new firmware image `<firmware.bin>` to the EEPROM of the WD3200AAKX. Malware, such as a rootkit, can also issue those vendor specific commands, given sufficient privileges to do so, i.e., it needs administrative privileges. Given this ability the possibilities of firmware manipulation are endless.

2.1.1. Providing persistent root access

In 2013 Jeroen "sprite_tm" Domburg demonstrated a firmware rootkit against a 500 GB Western Digital HDD that allowed for covert root access on UNIX systems persisting even system reinstalls and complete HDD erasure [4]. To this end, he hooked the interrupt table of the Marvel Feroceon ARM processor responsible for the SATA DMA transfers. The hooked interrupt is triggered once the requested data was loaded from platter into the cache of the HDD by the second Feroceon ARM processor on the main controller chip. Instead of instantly issuing the SATA DMA transfer, the hook code would first search the newly loaded data in cache for an ASCII string that indicates the `/etc/shadow` file of a UNIX system was read, i.e., a string having the form "`root:*:*:*:*:*:*:*:*:*\n`", with `*` representing a wildcard character for any ASCII character not containing `:` or no character, and `\n` representing a newline character. Here `root` represents the username, in this case, the root user, and `:` is a separator separating the individual fields. The first field is the username and the second field the password hash. The other fields are additional fields, that are irrelevant for this scenario. The rootkit then exchanges the password hash for a known password hash. This way the attacker injects a known root password into any UNIX system, or any other system employing the `/etc/shadow` password file for that matter, installed on such a compromised HDD — even if the system is reinstalled.

2.1.2. Activating and deactivating the rootkit (from remote)

Further hooks can be added to the firmware to enable and disable this root password overwrite, e.g., a second hook in the interrupt table for the interrupt triggered once data in the cache is supposed to be written to platter can be used to search for commands. These commands can then be triggered simply by writing them to disk. This way the regular root user can use the system with his or her password, but once the attacker wants to access the system all he or she needs to do is to write a secret command string to disk to activate the root password overwrite. sprite_tm used the strings "`HD, live`" to activate and "`HD, dead`" to deactivate his rootkit. An attacker can then inject such a command string, e.g., within the `User-Agent` string of a HTTP request to a server which is then subsequently written into the HTTP access logs of that server. Another example would be a SSH login attempt with a specially crafted username which would then be written to the security logs and thus trigger the command. An attacker may need to trigger multiple data writes to ensure that the commands are actually committed to the HDD and not reside in the kernel's file system cache, though. To this end, an attacker can simply flood the logs by issuing bogus requests. Any such actions needed to get the command to be committed to disk can then be deleted afterwards once the attacker has gained root access. In fact, the firmware rootkit could replace the commands found in the cache with innocent looking data, e.g., if the command to start the rootkit was "`3e91923511ce3ad01dc6ae56d3874dc6a7bef7aec532d...`
`...365a4d83037eca4f20a13374211`" it could be replaced with the valid user agent string "`Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101 ...`
`...Firefox/40.1`" before actually being written to disk.

2.1.3. Other HDD firmware rootkit examples

Zaddach et al. demonstrated a similar exploit to sprite_tm's work in their work *"Implementation and implications of a stealth hard-drive backdoor"* [22]. They further detailed optimizations to control the rootkit in a way which makes the attack more feasible than sprite_tm's prototype. Also, according to talks by the co-authors Goodspeed at 0x07 Sec-T Conference in 2014 [6], Kaminsky[3] at DEF CON in

[3] Even though not named on the publication, he admitted to being an "unindicted co-conspirator" [8, `https://youtu.be/xneBjc8z0DE?t=550`].

2014 [8], and Zaddach himself at REcon in 2014 [21], they used a 500 GB Seagate Barracuda HDD, showing that such firmware rootkits are not just affecting Western Digital drives.

In the following years more and more researchers successfully recreated these HDD firmware bootkits and e.g. like Marcus "MalwareTech" Hutchins even posting online about it[4]. These examples of security researchers recreating sprite_tm's work further show not only the feasibility, especially with regard to implementing their own firmware rootkits, but also the relevance in the field of anti-forensics. Given that multiple independent researchers have been able to recreate the rootkit and even openly posting about it online proves that criminals have access to this kind of information. Forensic scientists should, therefore, expect such firmware compromises when investigating high profile cases of industrial espionage, etc.

Last but not least, the crucial part about sprite_tm's prototype if used in a compromise is that there will be no evidence on the regular data storage areas of the HDD. And any attempt to clean the drive will be futile because the changed root password hash is modified on the fly and never actually written to the platter.

2.2. Interfering with data acquisition

Besides the above-mentioned scenario of persistent root access, trivial but yet efficient anti-forensic measures can be employed. E.g. the partitions of the HDD can be formatted in a way that after the partition table — irrelevant whether DOS/MBR or GPT — one sector is left unallocated, so it is never accessed by the system during regular operation. This sector can then be used by manipulated firmware as a digital trap [6, `https://youtu.be/8Zpb34Qf0NY?t=1095`] and trigger any combination of — but not limited to — the following anti-forensic measures:

- only return zeros when reading specific sectors of the HDD (i.e., sectors containing evidence)
- only return zeros on all reads from the device
- return zeros on all read request and overwriting the requested data with zero on the HDD

All the above measures impact the quality of the forensic results obtained from the device. While the first two are very subtle, they should, in general, be enough to stop forensic expert working according to the current state of the art from discovering evidence. But they still leave the evidence on the device. While measure three is not subtle, it will destroy the evidence. Given that imaging a drive is often automated and takes time, it is unlikely a forensic analyst realizes that the data has been overwritten after the drive was imaged.

In any case, the current state of the art does not consider such attacks and hence does not protect from them. Special equipment such as a write blocker will not prevent the drive itself, but only the investigator, from (accidentally) writing to the disk. Other practices, such as hashing the image, will also not help, because the data was already manipulated on the device. Imaging the device twice (or even three times), first without HPA, then with HPA (and then DCO) enabled, could, depending on the type of anti-forensic trap used, yield discrepancies between each imaging result. For example, the first time the partition table is read the data is read correctly. Then, once the sector after the partition table functioning as the trap is read the partition table will suddenly be zeroed out. However, in case the data was erased during the first imaging process, noticing discrepancies at the second imaging process is too late.

2.3. Detection (in EEPROM)

Detecting manipulated firmware must be done via hardware, i.e., physical access. This is because compromised firmware could simply acknowledge requests that new — non-compromised — firmware has been written successfully when the investigator tries to upload such non-compromised firmware to the device. The compromised firmware could use the Service Area to store a shadow firmware, which is returned on read requests and overwritten on write requests. All, of course, without actually touching the real firmware on the controller.

One possibility to verify firmware with software only access would be to upload firmware with a characteristic behavior, e.g., returning a magic value when reading a specific sector. This way one could make sure that at least this functionality of the firmware is executed and then assume the rest of the uploaded firmware is also executed. However, this gives no 100 % assurance, because compromised firmware could only temporarily load the newly uploaded firmware and at a later point switch back to the compromised firmware, e.g., via a command string written to disk as detailed in Section 2.1.2 on the previous page. This is possible because the Service Area provides enough storage for multiple versions of the firmware.

Here it is critical to acknowledge the fact that the Service Area can only reasonably be read through the controller itself and hence the firmware. This means without physical acquisition of the firmware binary no 100 % assurance can be given. However, in this section we only look at the currently easily available HDD firmware compromise methods, i.e., a bootkit residing within the EEPROM. A deeper HDD compromise down into the system and overlay modules within the Service Area is outlined in Section 3 on page 11.

To detect a bootkit as that follows the scheme outlined by sprite_tm [4], the firmware loader residing in the EEPROM must be verified. For this three steps are needed:

[4] `https://www.malwaretech.com/2015/04/hard-disk-firmware-hacking-part-1.html`

identifying the EEPROM chip, reading the EEPROM chip and last but not least verifying its contents.

2.3.1. Identifying the EEPROM

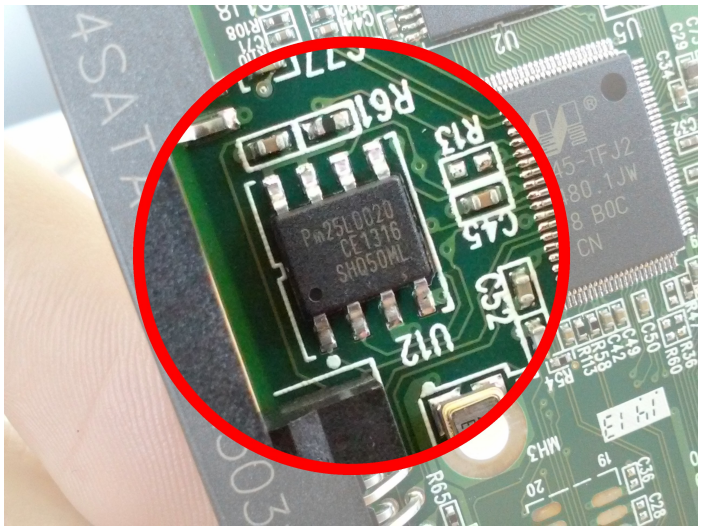


Figure 2: Identifying the EEPROM of a Western Digital WD3200AAKX HDD

The EEPROM can be identified by reading the part number on the chip. Because the marking is often very small it is advised to read the part number from a picture. Figure 2 shows a picture of the Western Digital WD3200AAKX EEPROM, taken with an inexpensive off-the-shelf consumer smartphone (LG L70). The direction of light is important to maximize the readability of the markings. As can be seen from the picture the EEPROM of this Western Digital WD3200AAKX is a Programmable Microelectronics Corporation Pm25LD020 chip. A list of chips used within different HDDs can be seen later in Table 1 on page 16.

2.3.2. Reading the EEPROM

The process of reading the firmware on our Western Digital WD3200AAKX example HDD is straightforward and can be done without desoldering the EEPROM chip. We used a commercially available SOIC-8 programming clamp and the Autoelectric MiniPRO TL866CS[5] programmer. Once the controller PCB has been removed from the HDD the SOIC-8 programming clamp can simply be clipped to the EEPROM chip as can be seen in Figure 3. This process is referred to as in-circuit programming. Figure 4 depicts the reading of a desoldered EEPROM chip.

To increase transparency, we used the open source `minipro` software by Valentin Dudouyt[6] instead of the official but proprietary software by Autoelectric. The MiniPRO programmer supports over 10 k different EEPROM chips. It only supports 3.3 V SPI EEPROM chips,

Figure 3: Reading an EEPROM with an in-circuit programming clamp without desoldering

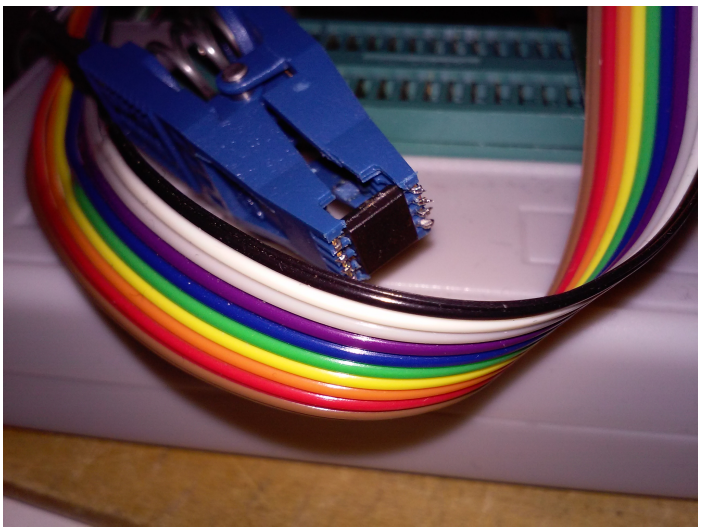


Figure 4: Reading the EEPROM after it has been desoldered from the HDD's PCB

however, some EEPROM chips are 1.8 V. To read those 1.8 V chips — or other chips unreadable by the MiniPRO — we used a Bus Pirate v4.0 by Dangerous Prototypes[7] in combination with the open source `flashrom` software[8].

Table 1 on page 16 shows all HDDs we tested for compatibility with this EEPROM acquisition method. The table lists the HDD Vendor, HDD Model, the EEPROM chip used and whether or not we were able to read the EEPROM in-circuit. We were able to acquire the firmware portion stored in the EEPROM of all the devices containing an EEPROM. However, only on about half, we could acquire the EEPROM in-circuit. Nevertheless, we demonstrate the practicability of our proposal. Reading out the EEPROM in-circuit does not pose an unreasonable

[5] `http://www.autoelectric.cn`
[6] `https://github.com/vdudouyt/minipro` (0.1)
[7] `http://dangerousprototypes.com/docs/Bus_Pirate`
[8] `https://www.flashrom.org/Flashrom` (0.9.8)

overhead in HDD data acquisition. We, therefore, propose that this procedure should be best practice when imaging HDDs of systems under suspicion of malware and/or rootkit compromises. Because the potential benefits, as we will show in the next sections, outweighs the effort to read the EEPROM. In case the EEPROM can not be read in-circuit, we, however, rather recommend imaging the drive without desoldering it first — unless there are strong indications that the drive is actually compromised. Even though we done numerous de- and resolders of EEPROM chips without problems the risk of damage to either the EEPROM, the PCB or even the controller chip exists and should be considered. Also a de- and resolder for every imaging procedure is not economical.

Even though in-circuit reading on the WD3200AAKX was straightforward, for other hard drives it was sometimes challenging to read the EEPROM in-circuit, i.e., without desoldering. In certain board layouts applying power to the EEPROM via the SPI interface would power-up the whole board, including the processor, causing the programmer to, either abort due to over-current protection, or return all ones on reading the chip ID or data. To remedy the problem, it sometimes is possible to hold the board in reset, i.e., pull down the system reset pin of the board, as can be seen in Figure 5. Further Figure 6 shows what we believe to be the reset pins and grounds of the HDDs listed in Table 1 on page 16 requiring this procedure for in-circuit reading. We are reasonably certain these are the respective correct reset pins. However, once connecting the marked contacts, we were able to read the EEPROM without any problems.

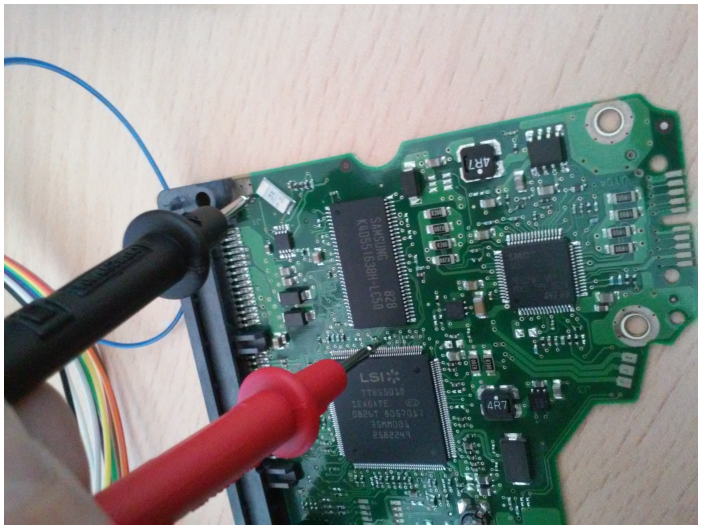


Figure 5: Holding a board in reset by pulling the processor's reset pin

Whether a board needs to be held in reset for in-circuit programming to work is indicated in Table 1 on page 16 with the [rst] footnote. On other occasions, when the error message from the `minipro` software is "`IO error: expected 7 bytes but 5 bytes transferred`", we found multiple quick successive read attempts to cause the reading to work. In this case presumably, the applied voltage supplied to the EEPROM charged the decoupling capacitors which are spread over the circuit board. Consequently, after these had been charged up, the EEPROM reached its required minimum supply voltage and was able to operate correctly. In the big table, we indicate this via the [rep] footnote.

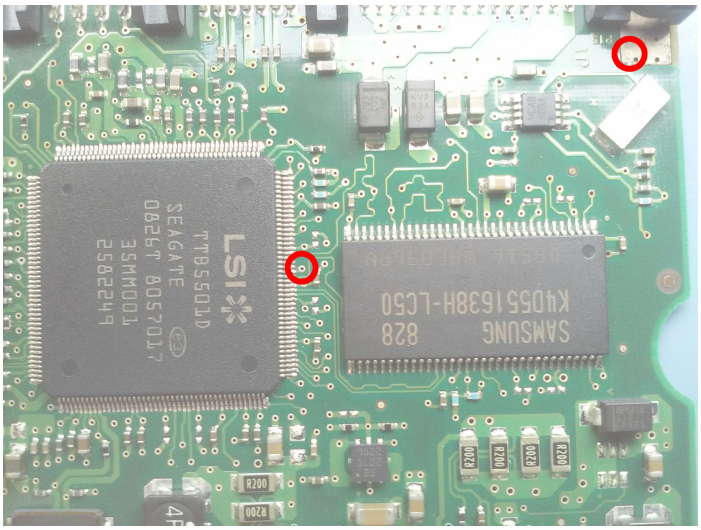


Figure 6: Reset and ground pins to be connected for in-circuit reading from an ST31000340NS

To verify a successful read we always performed three readouts and checked for identical bit patterns. We found this to be very important, because even slightly defective or loose contacts in either the SPI cables or the probe cables which hold the board in reset can cause read errors, which can go by unnoticed unless checked for.

Not all EEPROMs could be read in-circuit. Interestingly we were only unable to read the EEPROM of HDDs by Toshiba and HGST. These HDD's board layouts were basically identical. This is presumably because as of 2012 the HGST's 3.5" HDD division is owned by Toshiba. However, all the EEPROMs of all the HDDs listed in Table 1 on page 16 can be read by desoldering them. Further in case a HDD is not listed as in-circuit-programmable in Table 1 on page 16 does not mean it is in general impossible to do so, but rather that we did not manage to do so with our equipment.

2.3.3. Verifying EEPROM contents

In order to determine a procedure on how to verify different firmware obtained from the EEPROM, we used our example drive the Western Digital WD3200AAKX. More specifically we used 16 different ones. We extracted the EEPROM of each and then compared the retrieved contents. From this sample set we concluded that the firmware

retrieved from the EEPROM can not be compared directly, i.e., via a hash or bitwise comparison. This is because the EEPROM also stores information that is different for each drive. The 16 analyzed HDDs had 4 different firmware revisions with each revision being found on 8 and 5 drives respectively and 3 firmware revisions being found only on 1 drive each.

Figure 7 on the following page shows cross-comparisons, via `hexcompare`, of different firmware binaries from different WD3200AAKX HDDs. The blue and/or green parts are identical between the two binaries. The red areas mark differences.

Figure 7a on the next page shows a comparison of the EEPROM contents of drives using the same firmware revision. Here we can observe that overall the contents is almost identical. Only some byte sequences at the end of the EEPROM are different. These byte sequences are within so-called ROYL ROM modules. ROYL refers to the drive and firmware architecture used by Western Digital for the WD3200AAKX hard disk. The ROM refers to the type of module, i.e., in this case, a module residing in the EEP*ROM*. And last the term module refers to a part of the firmware which is organized modular in so-called modules. Here the difference is located in the modules with IDs 0x0a, 0x0d, 0x30, and 0x47. Module 0x0d contains identity information. Module 0x30 is the Service Area translator, i.e., where on the platter is the Service Area. Module 0x47 is responsible for surface adaptives, i.e., how far is the head away from the platter, etc. All these modules contain drive specific information, and hence, are different for each drive. We have also found module 0x0b (Module Directory) in the EEPROM as well. However, it was identical within each firmware revision.

As is already clear at this point a bitwise comparison is, even between the same firmware revisions, not possible, as the EEPROM already contains data individual to that specific hard disk, namely the surface adaptives and SA Translator modules. However, because the position of these modules is the same for the same firmware revision, they can simply be excluded from the comparison. Using these modules to store a rootkit or other compromising code is not possible, because the variable data is first too small to hold meaning full exploitation code and second not executed. Hence, if all other portions of the EEPROM contents are identical to known good EEPROM contents, it is safe to assess that the EEPROM can also be considered good and not compromised.

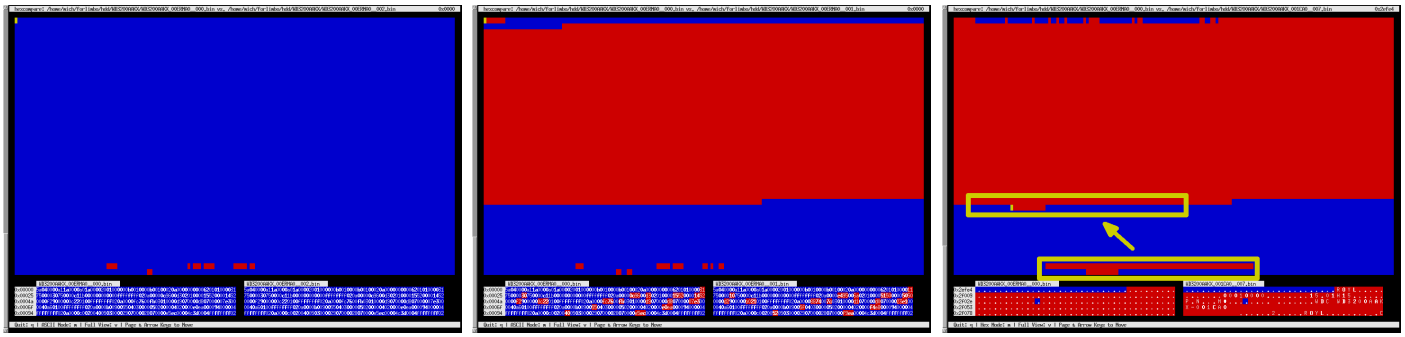
Figure 7b on the following page depicts a comparison of two different firmware revisions. Here the same modules as before contain differences, but also the actual firmware at the start of the EEPROM is different. Starting with the section header. It is different because some sections changed in size. Next, comes the bootstrap code, which is responsible for decompressing the other firmware sections. As can clearly be seen from the large blue section after the small initial red section this bootstrap code did not change between these firmware revisions. The following sections, however, are all completely red. Here it is important to note that this does not mean that the entire firmware changed between these revisions. Because these sections are compressed even small changes lead to different compression tables and a bitwise completely different data stream. The empty space after the sections containing the firmware is blue again, because they, for both revisions are filled with 0xff indicating empty flash blocks.

Figure 7c on the next page depicts the comparison of two different firmware revisions. Unlike the previous example, the changes between these firmware revisions can be characterized as major. First, the position of some ROYL modules changed. The modules are now located more closely after the last firmware section. The yellow frames and the arrow indicate from where to where the modules moved within the EEPROM.

Another noteworthy change is within the bootstrap code. Unlike before, the bootstrap code actually changed. Because the bootstrap code is not compressed, the unchanged sections remain identical. However, because adding and removing code changes the offset of all code following, the bootstrap code eventually also differs in our non-offset compensated bitwise comparison and is marked as red. For further discussion, it should be remembered that there are still unchanged parts of the bootstrap code showing up as blue, though.

Last but not least, it is important to acknowledge that even though the header changed, as before, its length remained the same. This means no additional sections have been added, nor have sections been removed. This indicates that the number of sections seems to be a rather fixed entity of the Western Digital ROYL firmware.

Figure 8 on page 10 shows comparisons of clean EEPROM firmware contents with infected EEPROM firmware contents. As before, a blue output indicates that there is no difference between the clean and infected EEPROM contents, while red indicates a difference. First, in Figure 8a on page 10 we compare the same EEPROM contents but before and after being infected with the rootkit as per sprite_tm's proof of concept rootkit code [3]. Unlike before almost the entire bits of the EPROM contents changed, save the empty part and the ROYL modules. This is because sprite_tm's bootkit adds itself as an additional section to the firmware. This causes the header at the beginning of the EEPROM to be extended by 32 bytes. This causes the sections following the header to be shifted by that 32-byte offset, thus, changing the EEPROM content from the non-infected EEPROM. Further, because the sections are shifted, their beginning addresses within the EEPROM change as well. This, in turn, causes the header fields containing these addresses to also change, which causes the



(a) Same firmware revisions (b) Different firmware revisions (c) More drastically different firmware revisions

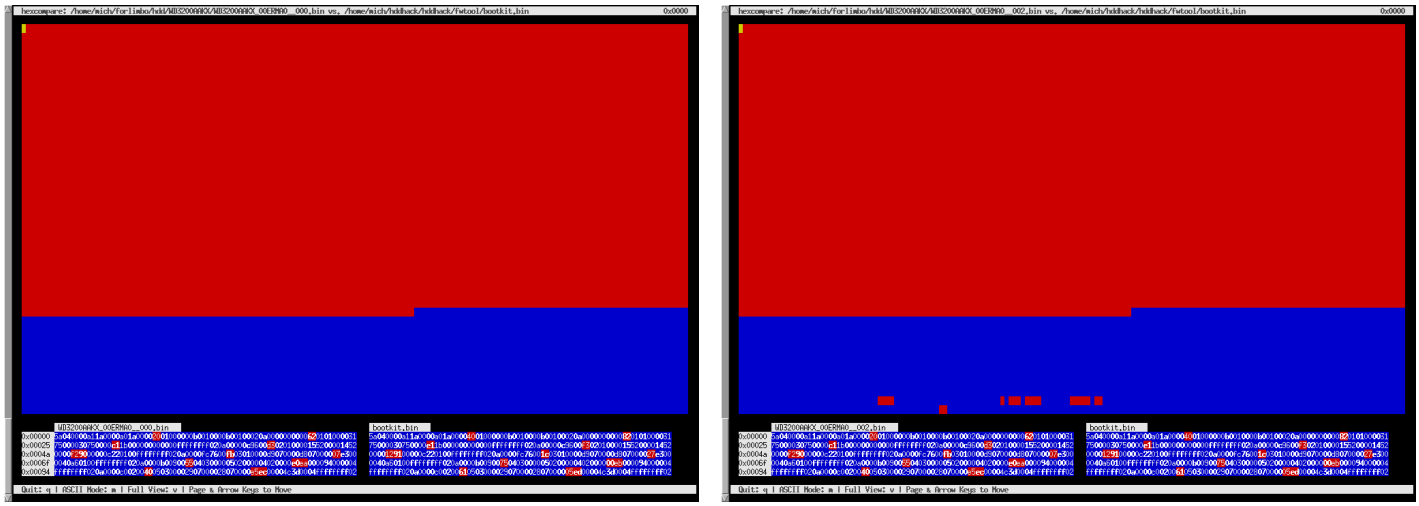
```
$ binwalk -W -i *000.bin *002.bin

OFFSET      WD3200AAKX_00ERMA0__000.bin                                     WD3200AAKX_00ERMA0__002.bin
------------------------------------------------------------------------------
*
0x0003D700  04 00 1E 00 0A 00 01 00 2F 6A D6 7E 30 30 30 31 |......../j.~0001| \ 04 00 1E 00 0A 00 01 00 1F 68 DE 88 30 30 30 31 |.........h..0001|
*
0x0003D720  03 00 7C 42 7C 41 4B 36 53 53 59 51 55 55 00 00 |..|B|AK6SSYQUU..| / 03 00 7C 42 7C 44 4B 37 59 52 51 46 55 55 00 00 |..|BIDK7YRQFUU..|
0x0003D730  FF FF FF FF FF FF 00 00 E0 F5 52 4F 59 4C 01 00 |..........ROYL..| \ FF FF FF FF FF FF 00 00 EA F5 52 4F 59 4C 01 00 |..........ROYL..|
0x0003D740  1E 00 30 00 02 00 A4 3B F4 4C 30 30 31 36 30 30 |..0....;.L001600| / 1E 00 30 00 02 00 A1 9A EC 4A 30 30 31 36 30 30 |..0.....J001600|
*
0x0003D760  00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 |................| \ 00 00 01 00 00 00 01 00 00 00 00 00 00 00 01 |................|
0x0003D770  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................| / 01 01 01 01 01 01 0D 00 00 00 F4 9E 05 00 00 00 |................|
*
0x0003DB40  1E 00 4F 00 01 00 DD 52 83 DE 30 30 30 35 30 30 |..O....R..000500| \ 1E 00 4F 00 01 00 36 4F 43 BB 30 30 30 35 30 30 |..O...6OC.000500|
*
0x0003DB80  00 00 00 00 00 00 00 00 00 00 55 AA 02 00 21 40 |..........U...!@| / 00 00 00 00 00 00 00 00 00 00 55 AA 02 00 7C 40 |..........U...|@|
0x0003DB90  C5 3E 00 00 00 00 00 00 00 00 00 00 00 00 55 AA |.>............U.| \ 39 42 00 00 00 00 00 00 00 00 00 00 00 00 55 AA |9B............U.|
0x0003DBA0  02 00 C6 00 5B DE A9 FF E1 FB F6 FF 13 00 1A 00 |....[...........| / 02 00 AD 00 CB EC 31 00 22 FC 03 00 F6 FF F2 FF |......1."...... |
0x0003DBB0  FB FF 00 00 02 00 69 FD E7 E5 EC FF D1 FB DB FF |......i.........| \ 0A 00 03 00 FB FF 55 FF 68 F7 A4 00 F0 FB FF FF |......U.h.......|
0x0003DBC0  23 00 FA FF F7 FF 03 00 08 00 01 00 50 49 4B 53 |#...........PIKS| / E3 FF 27 00 E6 FF 03 00 EC FF 01 00 50 49 4B 53 |..'.........PIKS|
*
0x0003DC00  52 4F 59 4C 01 00 1E 00 47 00 02 00 2C 11 D0 9A |ROYL....G...,...| \ 52 4F 59 4C 01 00 1E 00 47 00 02 00 54 4F 9F 98 |ROYL....G...TO..|
*
0x0003DC20  C0 01 00 00 00 00 00 00 00 00 00 00 00 00 36 4B |..............6K| / C0 01 00 00 00 00 00 00 00 00 00 00 00 00 37 4B |..............7K|
0x0003DC30  B1 45 10 80 72 00 0F 0D 10 0D 6D B0 BC 07 43 00 |.E..r.....m...C.| \ D1 45 1A 80 72 10 0F 0B 0D 0C 6D B0 BC 07 43 00 |.E..r.....m...C.|
0x0003DC40  40 DA 1F C0 01 03 07 28 4C 03 09 04 02 01 02 02 |@......(L.......| / 40 DA 2B C0 01 04 0A 2D 4C 0B 07 04 02 01 02 02 |@.+....-L.......|
0x0003DC50  C0 09 00 00 00 84 24 00 00 FF 20 00 80 40 E1 45 |......$..... .@.E| \ C0 09 00 00 00 84 20 00 00 FF 20 00 80 40 91 45 |...... ... .@.E|
0x0003DC60  1A 80 72 11 0F 0E 0F 0D 6D B0 BC 07 43 00 40 DA |..r.....m...C.@.| / 12 80 72 16 0F 0B 10 0C 6D B0 BC 07 43 00 40 DA |..r.....m...C.@.|
0x0003DC70  47 C0 03 05 0A 2D 4C FF 07 02 00 00 02 02 C0 09 |G....-L.........| \ 41 C0 03 05 0A 28 4C 06 07 03 01 00 02 03 C0 09 |A....(L.........|
0x0003DC80  00 00 00 84 23 00 00 FF 20 00 80 40 C0 45 20 80 |....#..... .@.E..| / 00 00 00 84 24 00 00 FF 20 00 80 40 C0 45 20 80 |....$..... .@.E..|
*
0x0003DDA0  0C 94 15 05 10 82 13 03 00 A0 0A 55 00 A0 0A 55 |...........U...U| \ 1A 71 17 05 1E 72 12 03 00 A0 0A 05 00 A0 0A 05 |.q...r..........|
0x0003DDB0  00 A0 0A 55 00 A0 0A 55 00 A0 0A 55 00 A0 0A 55 |...U...U...U...U| / 00 A0 0A 05 00 A0 0A 05 00 A0 0A 05 00 A0 0A 05 |................|
0x0003DDC0  1A 0C 7D 0C DC 0A DC 0A DC 0A DC 0A DC 0A DC 0A |..}.............| \ 5D 0C 83 0C DC 0A DC 0A DC 0A DC 0A DC 0A DC 0A |].............. |
*
0x0003DDE0  00 00 00 00 81 01 12 00 12 00 00 00 00 00 00 00 |................| / 00 00 00 00 81 01 12 00 11 00 00 00 00 00 00 00 |................|
0x0003DDF0  00 00 00 00 00 00 00 00 1F 01 39 39 39 39 39 39 |..........999999| \ 00 00 00 00 00 00 00 00 20 01 39 39 39 39 39 39 |.......... .999999|
0x0003DE00  39 39 AA A4 AA 00 A4 AA A2 9C A2 00 9C A2 21 14 |99..........!.| / 39 39 93 8C 93 00 8C 93 91 85 91 00 85 91 21 14 |99..........!.|
*
0x0003DE30  3C 3C 00 00 FD FF FF FF CF FE FF FF 5B 01 3F 00 |<<..........[.?.| \ 3C 3C 00 00 FB FF FF FF FE FE FF FF 5B 01 3F 00 |<<..........[.?.|
*
0x0003DE60  00 00 00 00 A8 08 58 13 88 09 88 09 88 09 88 09 |......X.........| / 00 00 00 00 23 13 00 19 88 09 88 09 88 09 88 09 |....#...........|
*
0x0003DE90  00 00 00 00 47 53 48 54 60 60 60 60 60 60 7E 7B |....GSHT``````~{| \ 00 00 00 00 47 53 48 54 60 60 60 60 60 60 7A 79 |....GSHT``````zy|
*
0x0003F000  52 4F 59 4C 04 00 1E 00 0D 00 01 00 6F C0 91 80 |ROYL........o...| / 52 4F 59 4C 04 00 1E 00 0D 00 01 00 A7 C3 E6 D4 |ROYL............|
*
0x0003F030  4E E2 07 BC 77 DB 00 01 FE FF 00 00 00 00 20 01 |N...w.......... | \ 4E E2 B2 67 3F D8 00 01 FE FF 00 00 00 00 20 01 |N..g?.......... |
*
```

(d) Detailed view of Figure 7a showing the differences in more detail

Figure 7: Cross-comparisons via `hexcompare` of the EEPROM contents of different WD3200AAKXs (Blue and/or green output is identical in both, while red marks differences)

checksums for the header sections to also change. Hence, the EEPROM contents almost completely changed, compared to the contents before the infection.

However, because a direct comparison of the EEPROM contents before and after infection is impracticable, we compared the infected EEPROM contents with clean EEPROM contents with the same firmware revision. The results can be seen in Figure 8b on the following page. We see the same changes as we saw in the case of comparing the same EEPROM contents with itself before and after infection. However, as demonstrated earlier in Figure 7a, the ROYL



(a) Firmware before and after bootkit infection as per sprite_tm's proof of concept rootkit code [3]

(b) Same firmware revision but one with bootkit infection as per sprite_tm [3]

Figure 8: Cross-comparisons via `hexcompare` of the EEPROM contents of a firmware bootkit infected WD3200AAKXs (Blue output is identical in both, while red marks differences)

modules located within the EEPROM after the firmware differ for different HDDs, even when containing the same firmware revision. Hence, we also see some changed byte sequences towards the end of the EEPROM.

We can now, with the above knowledge, formulate a procedure to verify EEPROM contents x against a set of known-good EEPROM contents $G = \{g_1, g_2, \ldots, g_n\}$:

- If $x \in G$ the EEPROM x is good. In fact, it is one of the samples.
- For each g in G check
  - whether $x \approx g$ with only differences in the ROYL modules after the firmware sections. If this is the case then the EEPROM x is good.
  - whether the section header in x contains more than one section with block code number 0x5a, i.e., main loader section. If this is the case the EEPROM x is definitively infected with a bootkit.
- Otherwise, the EEPROM x can not be verified as good.

Here it is important to understand that it is not enough to verify that the section header at the start of the EEPROM does not contain a second main loader section, i.e., a second section with block code number 0x5a. While this is enough to detect the prototype bootkit by sprite_tm, it is not enough to detect a more elaborate bootkit. In theory, a bootkit could hide in one of the compressed sections. Hence, even an EEPROM comparison looking like the top part of Figure 7c on the preceding page with the bottom part looking like Figure 7b on the previous page, does not indicate a clean EEPROM. The clean state can only be assessed via the above procedure.

Because the firmware is not digitally signed it is hard to be 100 % certain a firmware is not compromised, making it hard to build a set of known good EEPROM firmware samples. There are several ways to verify an EEPROM as good:

- Extract the EEPROM from vendor updates.
  - This seems like the most convenient way. It can also be secure in case the vendor update is digital signed. However, HDD firmware is usually not regularly updated via vendor updates.
- Reverse engineering the EEPROM contents and verify it does not contain any unintentional functionality.
  - This is the safest. This ensures even against compromise via vendor-supplied firmware. However, it is impracticable. Goodspeed acknowledged that it took the authors behind *Implementation and implications of a stealth hard-drive backdoor* [22] "ten man months" [6, `https://youtu.be/8Zpb34Qf0NY?t=1396`] to reverse engineer the firmware of a Seagate Barracuda HDD. He further stated that they "killed 15 hard disks" [6, `https://youtu.be/8Zpb34Qf0NY?t=1442`] during their research. And this was to develop a hook-based bootkit for the HDD. Verifying any and every functionality within the firmware could potentially take much longer. This makes it impracticable for a productive forensic context. It may be applicable within an audit for highest security environments.

- Determining good EEPROMs via clustering.
  - While this approach can not provide 100 % assurance, it provides a reasonable trade-off between practicability and certainty. The basic idea to build a set of good EEPROMs for the WD3200AAKX is as follows:
    1. Collect EEPROM contents of as many WD3200AAKXs as possible into your test set T.
    2. Add each EEPROM sample to its own cluster.
    3. Compare the EEPROM contents pairwise via `hexcompare`.
    4. If the comparison indicates the same firmware revision, i.e., the result looks similar to Figure 7a on page 9, with only differences in the ROYL modules 0x0a, 0x0d, 0x30, and 0x47 join the clusters of the two compared EEPROM samples.
    5. Define a trust-threshold t.
    6. Add one EEPROM sample of each remaining cluster to your good WD3200AAKX EEPROM sample set G if the number of EEPROM samples within the cluster is above your trust-threshold t.

    This scheme only works if no more than trust-threshold t many EEPROMs within the test set T are compromised. Hence, it is advised to not included HDDs that are suspected to be compromised in test set T. Ideally, test set T should only contain samples from a good source, e.g., bought directly from the vendor. Obviously, this does not protect against a bad vendor, unlike the reverse engineering method.

2.4. Subverting

A bootkit compromise as outlined previously can be subverted by replacing the compromised EEPROM contents with legitimate EEPROM contents. This can be done by writing the EEPROM via SPI. This is the inverse process of the EEPROM reading as outlined in Section 2.3.2 on page 6. We verified the writing process for the applicable test drives and the same rules regarding in-circuit programming as discussed for reading also apply to writing and can be found summarized in Table 1 on page 16. Another possibility is to replace the EEPROM by desoldering it, or replacing the entire PCB of the HDD. This process is often done by data recovery specialist in case of a bad EEPROM or bad PCB. Here it is important to leave the drive specific data from the ROYL modules 0x0a, 0x0d, 0x30, and 0x47 from the compromised EEPROM intact, because otherwise the correct functionality of the drive can be impacted. In case the EEPROM or PCB is swapped, this data must be copied to the replacement EEPROM.

2.5. Investigating

To investigate the actual firmware rootkit is a complex task. As noted earlier, Goodspeed acknowledged that it took ten man months to reverse engineer the firmware of a Seagate Barracuda HDD. This firmware is not obfuscated in any way. So if the task is to investigate a sophisticated firmware rootkit this time could potentially be increased dramatically.

3. Overlay/module-based hard disk firmware rootkit

In the previous section, we outlined how the boot loader code stored in the EEPROM can be verified to detect and combat firmware bootkits. However, as already suggested in the previous section, the EEPROM is not the only place where the HDD stores its firmware and hence malware can hide. After the boot loader code loaded from the EEPROM initialized the HDD enough to read from the spinning platter, further firmware code is loaded from the Service Area. Western Digital calls these additional firmware blocks modules, while Seagate calls them overlays. A firmware rootkit hiding in the Service Area is harder to detect than the previously discussed bootkit inside the EEPROM.

3.1. Verifying overlays/modules

In order to detect an infected overlay and/or module, the overlays and modules must be verified. The general ideas of Section 2.3.3 on page 8 can be reused to verify the overlays and modules. The challenging part is acquiring the overlays or modules without the risk of a rootkit interfering in the acquisition process. As outlined before a compromised firmware could simply return the original firmware when it is read, and keep it in a shadow copy for write requests to it — all while continuing to run the compromised firmware. We now outline the possibilities to read the Service Areas on our WD3200AAKX example HDD. We also provide an alternative memory analysis procedure that can be used to analyze a hard disk.

3.1.1. Reading Service Area

To read the Service Area of the WD3200AAKX there exist two possibilities. While the first relies on the potentially compromised firmware, the second can deliver 100 % trust, but requires a large amount of engineering.

Vendor Specific Commands (VSC). The Service Area can be read via SATA by issuing vendor specific commands (VSCs). *Hiding Data in Hard-Drive's Service Areas* by Berkman [2] outlines how the Service Area can be read from a Western Digital WD2500KS HDD. They also published their source code[9], which can be adapted to be used with the WD3200AAKX. The Service Area modules can further be read with the commercial PC-3000 tool suit.

[9] `http://www.recover.co.il/SA-cover/SA-cover-poc.c`

The problem with this method of reading the Service Area is that the vendor specific commands send via SATA to the HDD, are interpreted by the very firmware running the HDD. If that firmware is compromised, it can simply deny the read requests or return a clean copy of the original firmware modules, etc. Hence, this method is not anti-forensic resistant enough to withstand very deep and elaborated rootkit compromises.
The problem is complicated by the fact that the parts of the firmware responsible for providing the SATA interface are located in modules stored in the Service Area. Meaning that this firmware code can not be verified within the EEPROM. Hence a way to access the Service Area without using the firmware located within it must be devised.

Custom Boot Loader. The only way to read the Service Area without relying on the firmware located within it is to use a custom boot loader. To this end, the EEPROM can be programmed via physical access, again programming via software will invoke potentially compromised firmware on the HDD. This is not a trivial task. However, because the HDD runs any code without any signature checks whatsoever, it boils down to reverse engineering how the firmware accesses the Service Area and reimplementing this functionality into a minimal boot loader, which dumps the contents of the Service Area. Many HDDs have a UART interface which can be used to load additional code in case the EEPROM provides insufficient space, and to possibly dump the Service Area contents. Table 1 on page 16 provides an overview of which HDDs contain such a UART interface. Many HDDs also contain the testing and debugging interface JTAG, which can aid in developing such a custom boot loader. Whether a HDD has JTAG is also documented in Table 1 on page 16. The documentation lists the IDCODE that should be expected to be found when performing an IDCODE scan on the JTAG interface. Again, if Table 1 on page 16 does not list UART or JTAG on a specific HDD does not mean it does not have JTAG or UART. It simply means that during our investigations we have not found such interface or could not utilize it. We outline JTAG in more detail in Section 3.2.1.

3.2. Memory analysis

Because developing a custom boot loader to dump the Service Area without involving potentially compromised firmware, we propose memory analysis as an alternative. This can be compared with doing a live analysis on a computer system — in this case the HDDs processor.

3.2.1. Dumping memory via JTAG

As mentioned before, many hard disks contain the testing and debugging interface JTAG. We can use the JTAGulator hardware[10] to determine the JTAG pinout of suspected JTAG pins. Figure 9 shows the JTAG pin

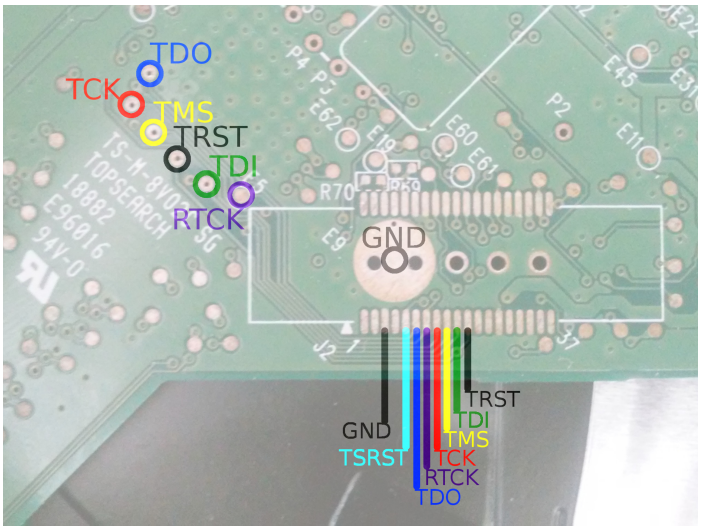


Figure 9: JTAG pin layout of the WD3200AAKX

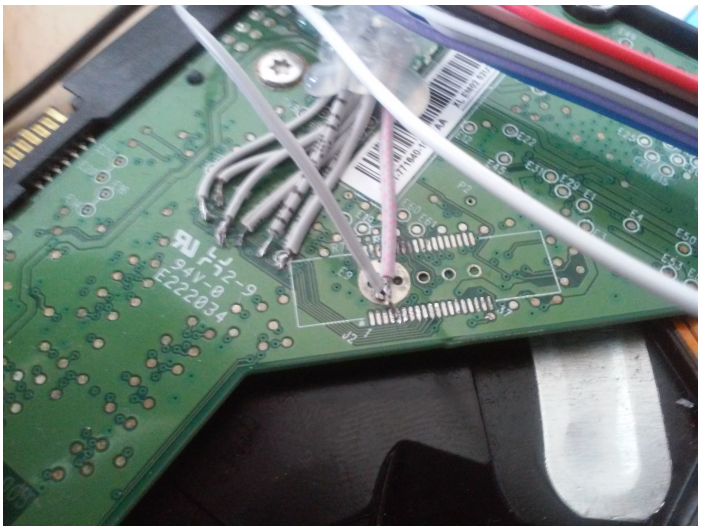


Figure 10: Wires soldered to the JTAG pins of a WD32000AAKX

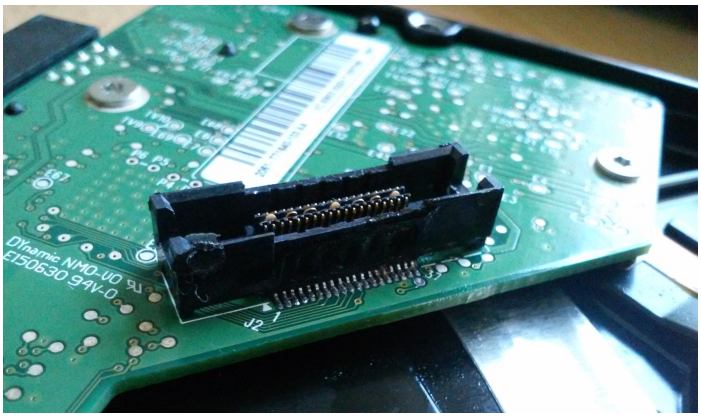


Figure 11: A MICTOR 38 pin connector soldered to the JTAG test pads of a WD32000AAKX

[10] `http://www.grandideastudio.com/jtagulator/`

layout of our example WD3200AAKX HDD. Figure 10 on the previous page shows wires soldered to the JTAG pins, which are then connected to the BusBlaster v4[11] JTAG interface. In Figure 11 on the preceding page, a MICTOR connector was soldered to the WD3200AAKX. This not only provides a more convenient access, but also is easier to solder to the pads than individual wires. This means that the soldering skills only need to be moderate to achieve a JTAG connection.

With OpenOCD we can then dump the relevant firmware files. The used OpenOCD commands can be seen in Listing 1. The mask ROM boot loader is mapped into memory at offset 0xffff0000. It is dumped via the first command. Then the firmware sections of the EEPROM are dumped.

```
dump_image bootrom.bin 0xffff0000 0x10000
dump_image bootsection0.bin 0x1b000 0x1aa0
[...]
```

Listing 1: OpenOCD command to dump memory sections of a WD3200AAKX HDD

The addresses of the firmware sections in memory can be extracted from the header in the EEPROM. This can, e.g., be done with sprite_tm's fwtool [3, `http://spritesmods.com/hddhack/hddhack.tgz`] via `./fwtool -i eeprom.bin`. The "`Block load vaddress`" of each section in the output is the memory address to which that section is loaded by the boot loader. Next, the same procedure can be used to dump the entire address space and verify code integrity by reverse engineering.

The drawback of the memory analysis method is that only firmware code that is currently loaded into memory can be captured. Firmware modules, or module backups, residing in the Service Area are not captured. Because the firmware is capable of loading additional modules from the Service Area at any time, the memory should be dumped periodically and analyzed for changes. Even though this can also not provide 100 % assurance, it is more feasible, to begin with, than developing a boot loader. But ultimately this form of memory analysis should be used to reverse engineer the hard disk under investigation to the point where developing custom boot loader code is possible. This is especially true, since all HDD controller processors we tested via JTAG were all MMU-less, i.e., not using a memory managing unit (MMU) to restrict memory accesses. This means any process running on the controller can modify any and all code and data in RAM. This means the only way to truly assess non-compromise is when each and every instruction executed by the processor is verified before execution.

3.2.2. Dumping memory via UART

While not anti-forensic resistant because it involves the firmware of the HDD, is the possibility to leverage the debugging console provided by Seagate HDDs via their UART interface. Figure 12 shows the UART pin layout. Table 1 on page 16 lists the baud rates and stop bit configurations of the tested Seagate HDDs. Booting the HDD should output a message similar to the one listed in Listing 2. Once this message is displayed commands can be entered. Listing 3 on the next page displays an example session. First, Ctrl+Z is pressed to activate the ASCII Diagnostic Serial Port Mode. This is acknowledged by the device by displaying the `F3 T>` prompt. From there further commands can be issued. In the example the Diagnostic Command Level is changed to Level C by entering `/C`. Then the ASCII Command Information is displayed via `Q`.

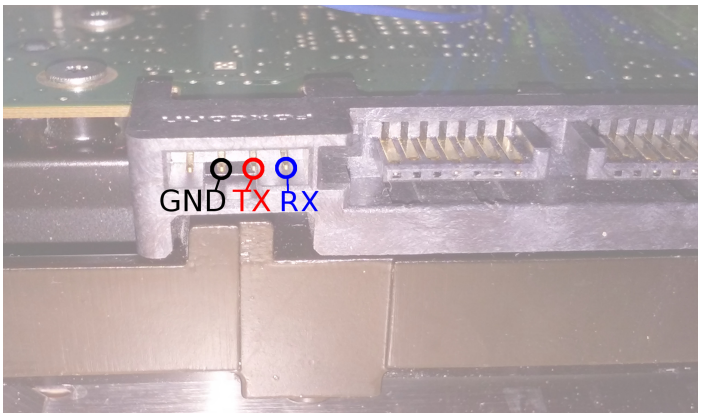


Figure 12: UART pin layout of Seagate HDDs

```
Boot 0x40M
 Spin Up
TCC-001C[0x000065B4][0x00006A20][0x00006E8C]
 Trans.

Rst 0x40M
 MC Internal LPC Process
 Spin Up
TCC-001C
(P) SATA Reset

 MCMainPOR: Start:
 Check MCMT Version: Current
 MCMainPOR: Non-Init Case
Reconstruction: MCMT Reconstruction Start
  Max number of MC segments 0A61
 Nonvolatile MCMT sequence number 000161C9
 [RSRS] 07C8
Reconstruction: Completed 1:
[MCMTWS]
 MCMainPOR: MCTStateFlags 0000002A  MCStateFlags 00005141
 MCMainPOR: Feature Enabled...

[SR] 0000
 PowerState = IDLE1
```

Listing 2: Boot message on UART from Seagate ST2000DM001

The interesting commands to peek and poke, i.e., read and write memory, are `+` to peek a byte and `=` to write a byte. Zaddach et al. [22] were able to use this simple interface to place a GDB stub into the firmware and debug it further.

[11] `http://dangerousprototypes.com/docs/Bus_Blaster`

```
F3 T>/C

F3 C>Q

Online CR: Rev 0011.0000, Flash,   Abort
[...]
Online ^C: Rev 0011.0000, Flash,   Firmware Reset
[...]
Online ^Z: Rev 0011.0000, Flash,   Enable ASCII Diagnostic Serial Port Mode
[...]
All Levels '/': Rev 0001.0000, Flash,   Change Diagnostic Command Level, /[Level]
All Levels '+': Rev 0012.0000, Flash,   Peek Memory Byte, +[AddrHi],[AddrLo],[NotUsed],[NumBytes]
All Levels '-': Rev 0012.0000, Flash,   Peek Memory Word, -[AddrHi],[AddrLo],[NotUsed],[NumBytes]
All Levels '=': Rev 0011.0002, Flash,   Poke Memory Byte, =[AddrHi],[AddrLo],[Data],[Opts]
[...]
Level 1 'S': Rev 0011.0001, Flash,   Edit Processor Memory Byte, S[AddrHi],[AddrLo],[MemValue],[NumBytes],[Opts]
Level 1 'U': Rev 0011.0001, Flash,   Edit Buffer Memory Byte, U[AddrHi],[AddrLo],[MemValue],[NumBytes]
Level 1 'e': Rev 0011.0000, Flash,   Spin Down and Reset Drive, e[MsecDelay],[Opts]
Level 1 'm': Rev 0011.0001, Flash,   Edit Processor Memory Word, m[AddrHi],[AddrLo],[MemValue],[NumBytes],[Opts]
[...]
Level C 'Q': Rev 0001.0000, Overlay, Display ASCII Command Information, Q[CmdLevel],[Cmd]
[...]
Level T '[': Rev 0011.0000, Overlay, ASCII Log Control, [[LogFunction],[Log]
```

Listing 3: Displaying the available commands over the ST2000DM001's UART

Seaget[12] is a project that uses this UART memory peeking interface to dump the memory and buffers of a Seagate HDD, hence also dumping the firmware as it resides in memory. Seaget simply automates the above manual commands.
As hardware interface we used the Bus Pirate, but also a common off-the-shelf 3.3v USB-to-serial works. as software we used GNU screen.

4. Discussion

Before concluding this paper we would like to discuss the various compromises and how either our work or already existing research can be used to detect the compromise.

4.1. Compromise in EEPROM

A compromise located within the EEPROM, such as firmware bootkits, can be easily detected via the methods outlined in Section 2.3 on page 5. Because currently, all publicly available source code with regard to HDD rootkits use a bootkit within the EEPROM, verifying at least EEPROM integrity during forensic investigations seems like a reasonable addition to HDD acquisition.

4.2. Compromise in Service Area

As we have also outlined in this paper more elaborate malware residing in the firmware overlays or modules waiting to eventually be triggered is also possible. We propose the development of custom boot loaders to dump potentially infected overlays and/or modules from the Service Area. However, due to the high complexity and unavailable documentation of the HDDs internal structures, such a task is not trivial. Our proposed memory analysis method can, however, be used to search for malware during firmware execution. But this is no automated task and requires a large amount of reverse engineering.

4.3. Compromised controller hardware

What has not been discussed in this paper is a hardware compromise. This means that the controller chip itself is compromised. Either the chips mask ROM has been compromised during manufacturing or the chip itself has been replaced with an optical identical but functional different chip. One example of such component replacement is the NSA's FLUXBABBIT project[13]. Compromised hardware can either be detected via differential power analysis and related techniques [17] or depending on the type of compromise via X-Ray [5]. A destructive attestation method would be to "decap" the chip and successively remove each of the chip's layers and use a scanning electron microscope to verify the chip's individual gates [12, 13, 15, 19].

However, hardware compromise is very unlikely, as this would mean redeveloping the entire chip with compromising enhancements. Hence a more likely scenario is the compromise of the boot ROM stored within the controller chip. We were not able to verify the boot ROM directly. However, it can be verified via JTAG by reading it as mapped into memory at offset 0xffff0000. But this assumes the JTAG hardware itself isn't compromised. To rule out any hardware compromise in the controller the entire controller chip can be replaced with a controller chip of a known non-compromised hard-disk. In fact, for simplicity, the entire controller board can be replaced with a known non-compromised controller board. In such a case, the HDD specific calibration data and references to the firmware overlays in the Service Area must, however, be copied from the old EEPROM onto the new EEPROM, as already outlined in Section 2.4 on page 11.

4.4. SSDs

While this paper was demonstrated with a WD3200AAKX HDD, parts of this paper are also applicable to SSDs. To demonstrate this we also tested SSDs. These are listed in Table 1 on page 16. While we were not able to find an SSD that contained firmware on an EEPROM, we were able to find JTAG interfaces on most of them.

[12] `https://github.com/BlackLotus/seaget`

[13] `https://cryptome.org/2014/01/nsa-codenames.htm`

5. Conclusion and future work

After presenting the state of the art in hard drive firmware rootkit compromise, outlining detection and ramification measures for bootkits residing in the EEPROM and also a presentation of possible detection methods for more sophisticated rootkits hiding within overlays or modules in the Service Area, we would like to conclude this paper. While we do not completely solve the problem of hard drive rootkits, it is the first work to investigate such hard drive compromises from a forensic perspective. It thereby offers the forensic community a first starting point for immediate measures, such as saving and potentially verifying the EEPROM contents when acquiring hard drives. This way EEPROM residing bootkits can be ruled out during investigations. This is important because, as has been shown in this paper, the knowledge on how to implement these bootkits is publicly available. Hence, this form of hard drive compromise may no longer be the reserved domain of government attackers, but may eventually end up being used by criminals.

Future work in this area is the establishment of a known good firmware database. Maybe even support by VirusTotal[14] (an online malware scanning service) to scan hard drive firmware samples. They already provide a possibility to scan BIOS firmware binaries.

Another important future step is verifying the overlays and modules stored in the Service Area without relying on the firmware of the drive itself and also, for transparency sake, without relying on proprietary software. Vendors are also responsible for ensuring their firmware can not be trivially manipulated anymore. To this end, digital signing, as already proposed by Seagate in a technology paper [16], must become common practice.

However, to not be at the mercy of trusting the vendor, the outlined firmware verification, especially of the Service Area must still be pursued.

[14] `https://virustotal.con`

Type	Vendor	Model	EEPROM	V_{dd}	IC	Reader	JTAG	UART
HDD	HGST	HTS541010A9E680	MX25L4006E	3.3	no			
		HUA722020ALA330	LE25FU206	3.3	no			
		HUA722020ALA330	PM25LD020	3.3	no			
		HUA723020ALA640	LE25FS406	1.8	no			
	Samsung	SP2504C	-					57,600-8N1[dbg]
	Seagate	ST10000DM003	LE25S81QE	1.8	no			38,400-8N1[dbg]
		ST10000DM003	W25Q80.W	3.3	yes	Bus Pirate		38,400-8N1[dbg]
		ST2000DM001	EN25S40	1.8	yes	Bus Pirate		38,400-8N1[dbg]
		ST31000340NS	W25X40L	3.3	yes[rst]	MiniPRO		38,400-8N1[dbg]
		ST3320620AS	AT25F512AN	3.3	yes[rep]	MiniPRO		9,600-8N1[dbg]
	Toshiba	DT01ACA100	PM25WD040	3.3	no			
		DT01ACA300	PM25WD040	3.3	no			
	WD	WD3200AAKX	W25X40	3.3	yes	MiniPRO	`0x121003d3`	115,200-8N1[xmod]
		WD3200AAKX	PM25LD020	3.3	yes	MiniPRO | Bus Pirate	`0x121003d3`	115,200-8N1[xmod]
		WD5000AAKX	W25X20BL	3.3	yes	MiniPRO	`0x121003d3`	115,200-8N1[xmod]
		WD800JD	SST25VF010	3.3	yes	MiniPRO | Bus Pirate	`0x25966ed3`	
SSD	Intel	SSDSA2CT040G3	W25X40C[0x0]	3.3	yes	MiniPro		
	Micron	MTFDDAC128MAM-1J1	-				?[hdr]	
	Samsung	MZ-75E500	-				`0x4ba00477`	
	Super Talent	STT_FTM32GX25H	-				`0x4f1f0f0f`	
	Zheino	SM2246HX-MS-4B1X2-V1.2	-				`0x100434b1`	
		SM2246MX-MS30-1B1X2-V1.0	-				`0x100434b1`	
		SM2246HX-15-4B1X2-V1.3	-				`0x100434b1`	

[rst] Controller must be held in reset.
[rep] Multiple quick read attempts must be issued. Or an external 3v3 supply may be used to overcome the power issues.
[dbg] Fully fledged debugging console. Note: The baudrate can be changed.
[xmod] Xmodem protocol for downloading code. Must be enabled via shorting pins.
[0x0] Did not contain any firmware. EEPROM was zeroed.
[hdr] SSD has labeled JTAG header, but JTAGulator's IDCODE scan fails.

Table 1: Analyzed HDDs: listing their EEPROMs with the corresponding voltages (V_{dd}); in-circuit programmability (IC); SPI reader used for in-circuit reading (if possible); IDCODE found via JTAG; baud rate and stop bit configuration for UART interfacing; (where applicable)

References

[1] Harald Baier and Julian Knauer. AFAUC - anti-forensics of storage devices by alternative use of communication channels. In *Eighth International Conference on IT Security Incident Management & IT Forensics, IMF 2014, Münster, Germany, May 12-14, 2014*, pages 14–26, 2014. doi: 10.1109/IMF.2014.11. URL http://dx.doi.org/10.1109/IMF.2014.11.

[2] Ariel Berkman. Hiding data in hard-drive's service areas. Paper published via fulldisclosure@seclists.org (paper: http://www.recover.co.il/SA-cover/SA-cover.pdf), 2013.

[3] Jeroen "sprite_tm" Domburg. Sprites mods – hard disk hacking. Website: https://spritesmods.com/?art=hddhack, 2013.

[4] Jeroen "sprite_tm" Domburg. Hard Disks: More than just block devices. Talk at OHM (video: https://www.youtube.com/watch?v=0Da6OARhgXk), 2013.

[5] Houda Ferradi, Rémi Géraud, David Naccache, and Assia Tria. When organized crime applies academic results: a forensic analysis of an in-card listening device. *J. Cryptographic Engineering*, 6(1):49–59, 2016. doi: 10.1007/s13389-015-0112-3. URL http://dx.doi.org/10.1007/s13389-015-0112-3.

[6] Travis Goodspeed. Active disk antiforensics and hard disk backdoors. Talk at 0x07 Sec-T Conference (video: https://www.youtube.com/watch?v=8Zpb34Qf0NY), 2014.

[7] Mayank R. Gupta, Michael D. Hoeschele, and Marcus K. Rogers. Hidden disk areas: HPA and DCO. *IJDE*, 5(1), 2006. URL http://www.utica.edu/academic/institutes/ecii/publications/articles/EFE36584-D13F-2962-67BEB146864A2671.pdf.

[8] Dan Kaminsky. Secure random by default. Talk at DEF CON 22 (video: https://youtu.be/xneBjc8z0DE?t=203), 2014.

[9] Kaspersky Labs GReAT (Global Research & Analysis Team). Equation group: Questions and answers. Version 1.5, Februar 2015 (paper: https://securelist.com/files/2015/02/Equation_group_questions_and_answers.pdf), 2015.

[10] Julian Knauer and Harald Baier. Zur Sicherheit von ATA-Festplattenpasswörtern. *Proceedings of D-A-CH Security*, pages 26–37, 2012.

[11] J. Laan and N. Dijkhuizen. Firmwall: Protecting hard disk firmware. Published online (paper: https://www.os3.nl/_media/2013-2014/courses/ot/jan_niels.pdf tool: https://github.com/janlaan/firmwall), 2014.

[12] Karsten Nohl and Jan "Starbug" Krissler. Deep silicon analysis. Talk at HAR (video: https://www.youtube.com/watch?v=UoLyYZzXjQE), 2009.

[13] Karsten Nohl and Jan "Starbug" Krissler. Silicon chips: No more secrets. Talk at PacSec (slides: http://www.degate.org/Pacsec2009/091001.Pacsec.Silicon.pdf), 2009.

[14] Huw Read, Konstantinos Xynos, Iain Sutherland, Gareth Davies, Tom Houiellebecq, Frode Roarson, and Andrew Blyth. Manipulation of hard drive firmware to conceal entire partitions. *Digital Investigation*, 10(4):281 – 286, 2013. ISSN 1742-2876. doi: http://dx.doi.org/10.1016/j.diin.2013.10.001. URL http://www.sciencedirect.com/science/article/pii/S1742287613001072.

[15] Martin Schobert. Semiautomatisches Reverse-Engineering von Logikgattern in integrierten Schaltkreisen (speziell zur Aufklrung geheimgehaltenener Verschlsselungsverfahren). Talk at 0sec (slides: http://www.degate.org/documentation/0sec_talk_degate.pdf), 2009.

[16] Seagate. Maximize security, lock down hard drive firmware with Seagate Secure Download & Diagnostics. Technology Paper, 2015. URL http://www.seagate.com/files/www-content/solutions-content/security-and-encryption/en-us/docs/seagate-secure-download-diagnostics-with-maximize-sec-lock-down-hard-drive-firmware-tp684-1-1508us.pdf.

[17] Sergei Skorobogatov and Christopher Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, pages 23–40, 2012. doi: 10.1007/978-3-642-33027-8_2. URL http://dx.doi.org/10.1007/978-3-642-33027-8_2.

[18] Iain Sutherland, Gareth Davies, Nick Pringle, and Andrew Blyth. The impact of hard disk firmware steganography on computer forensics. *JDFSL*, 4(2):73–84, 2009. URL http://ojs.jdfsl.org/index.php/jdfsl/article/view/165.

[19] Christopher Tarnovsky and Karsten Nohl. Reviving smart card analysis. Talk at Chaos Communication Camp (slides: https://events.ccc.de/camp/2011/Fahrplan/attachments/1888_SRLabs-Reviving_Smart_Card_Analysis.pdf, videos: https://www.youtube.com/watch?v=fFx6Rn57DrY), 2011.

[20] Martin Wundram, Felix Freiling, and Christian Moch. Anti-forensics: The next step in digital forensics tool testing. In *Seventh International Conference on IT Security Incident Management and IT Forensics, IMF 2013, Nuremberg, Germany, March 12-14, 2013*, pages 83–97, 2013. doi: 10.1109/IMF.2013.17. URL http://dx.doi.org/10.1109/IMF.2013.17.

[21] Jonas Zaddach. Exploring the impact of a hard drive backdoor. Talk at REcon (slides: https://recon.cx/2014/slides/Recon14_HDD.pdf, video: https://www.youtube.com/watch?v=KjmsLvD76rM), 2014.

[22] Jonas Zaddach, Anil Kurmus, Davide Balzarotti, Erik-Oliver Blass, Aurélien Francillon, Travis Goodspeed, Moitrayee Gupta, and Ioannis Koltsidas. Implementation and implications of a stealth hard-drive backdoor. In *Annual Computer Security Applications Conference, ACSAC '13, New Orleans, LA, USA, December 9-13, 2013*, pages 279–288, 2013. doi: 10.1145/2523649.2523661. URL http://doi.acm.org/10.1145/2523649.2523661.

All online references last accessed 2016-07-17.