Forensic Issues in IoT Devices Using NAND Storage

Project report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

Submitted by

Suchit Reddi (2010110507)

Under supervision

of

Dr. Sonal Singhal
Department of Electrical Engineering

DEPARTMENT OF ELECTRICAL ENGINEERING

SCHOOL OF ENGINEERING

SHIV NADAR INSTITUTION OF EMINENCE

(December 2023)

Candidate Declaration

I/We hereby declare that the thesis entitled "Forensic Issues in IoT Devices Using NAND Storage" submitted for the B. Tech. degree program. This thesis has been written in my/our own words. I/We have adequately cited and referenced the original sources.

(Signature)

Suchit Reddi

(2010110507)

Date: 27/11/2023

CERTIFICATE

It is certified that the work contained in the project report titled "Forensic Issues in IoT Devices using NAND Storage" by "Suchit Reddi" has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree.

(Signature)

Dr. Sonal Singhal

Dept. of Electrical Engineering

School of Engineering

Shiv Nadar Institution of Eminence

Date: 27/11/2023

Abstract

This project focuses on rendering personal data unrecoverable on IoT devices utilizing NAND flash memory. Cloud providers use NAND flash storage to reduce latency. Due to high write and erase speeds, it is highly suitable for sensitive applications like DBMS and banking functions. Point-of-sale devices are an example of the wide usage of IoT devices to store critical and sensitive data.

It is logical to assume that many users are unaware of proper storage sanitization techniques. Personal information like medical or financial records must be sanitized before discarding storage devices. The recoverability of any erased file depends on the sanitization technique used and the determination of the entity attempting file recovery.

The working of modern storage devices using NAND flash memory differs from legacy magnetic drives. The old sanitization/recovery methods might not be effective. The effectiveness of a data sanitization technique on a specific device depends on various factors like type, compatibility, and level of access. Some will reduce the device's lifespan more than others. We will look into multiple data sanitization techniques at the hardware and software level.

Various tools are available to sanitize storage devices on laptops and computers with a display. However, IoT devices have limited access and resources. A tool that can be used on low-end IoT devices with just shell access is much needed. We developed a command-line utility to fulfill all these requirements. This will include the most effective and compatible methods for IoT devices.

Table of Contents

List of Figures

List of Command

WARNING: Don't execute commands before learning what they do. Most of these commands result in data sanitization.

Chapter 1

Introduction

Advantages such as higher storage density, faster write and erase speeds, no mechanical latency from moving parts, and prices dropping every year in line with Moore's Law resulted in higher usage of NAND flash storage devices. They are used as cost-effective storage for IoT devices [1]. Solid State Drive (SSD) is a perfect testing device for this project that uses 100% NAND flash memory as its storage element [2][3].

IoT devices like Point-of-Sale devices store critical information like credit card data. The Target Breach, 2013, resulted from hackers scraping card data from the RAM and disk storage of PoS devices. About 40 million credit cards were stolen in a month [4]. Encrypting the storage from the beginning is recommended to avoid incidents like this and ensure data security and privacy, but it is not always possible. So, sanitization of data inside the disk storage of IoT devices is crucial.

In NAND flash devices, erase must be performed block-wise. If we flip and update a single bit in a block, the whole data block will be read and rewritten into a new block along with the altered bit. This creates a copy that is marked inaccessible but only deleted once wear leveling occurs. All these copies must be deleted for proper sanitization.

Data recovery and sanitization contradict each other. But progress in one drives the other. Sanitization tries to eliminate the possibility of recovery, while recovery tries to get back data even after sanitization is performed properly or improperly. Manufacturer-specific tools for sanitization exist but are not always compatible and are not implemented as expected [5].

We will discuss sanitization procedures for IoT devices with shell access, without any display or graphic user interface. We will look into possible hardware and software sanitization methods. We will test these methods using available equipment and observe the outcomes. All these methods are combined into a useful command-line utility for everyday users to sanitize and protect their sensitive information from prying eyes.

Chapter 2
Literature Survey

Erasing in HDD vs SSD:

Magnetic drives work differently from modern flash storage [6]. Overwriting mechanical disks with random bits once or multiple times is often enough. Degaussing randomizes the magnetization of grains on the magnetic medium of each disk, rendering it unusable. Physical destruction is a final step, and when done right, it ensures no possibility of data recovery. If any techniques are performed incorrectly, it is possible to recover data using advanced techniques such as Magnetic Force Microscopy. This process is discussed in detail by Vasu Kanekal in [7].

In SSDs, data is programmed electrically. So, most of the old methods will be ineffective. Simply overwriting the disk does not work because of wear leveling and TRIM performed by the NAND flash controller [8][9]. It fragments and stores a single file in various blocks. Flash-specific methods will be discussed in this paper.

Hardware Sanitization Techniques:

Sanitization can be done through software [10] or hardware. Performing hardware sanitization is complex and requires an advanced understanding of the intricate details of NAND chips and their interfacing. Experience handling specialized equipment like flash readers (PC-3000), programmers (Xeltek SuperPro), socket adapters (TSOP48 DIP48), and development boards (FTDI FT2232H) is needed.

This domain of research has very few resources online. Only a few researchers and enthusiasts performed operations directly on individual NAND flash chips and documented them [11][12][13][14]. This approach should provide a higher level of control over the sanitization process but is not very user-friendly. This hardware-oriented research might be useful for a specific device category called “mtd devices”. Low-level codes for specific chips are written for mtd devices to write and erase data in block and bit levels. Our project does not cover this in-depth, but it is a good future prospect.

Chapter 3

Work Done

Sanitization on an SSD must be applied to the whole drive and not to smaller partitions or individual files because of file fragmentation by the NAND flash controller. After looking into many sanitization methods, we added the most effective and compatible ones to this section. For the source code, refer to the tool[1] we developed. This tool was made to be user-friendly to help the general public sanitize their storage drives.

I. Sanitization techniques researched:

Firmware-based:

This process uses firmware sanitization commands provided by the manufacturer. These are executed via different tools based on the device interface. ATA devices can use hdparm [18], SATA can use sg3-utils, and NVMe uses nvme-cli for sanitization. These tools only define functions like "secure erase" and "enhanced secure erase". The underlying operations are programmed by proprietary firmware complying with these standards. Thus, verifying if the command does what it claims is hard without source code.

It is a NIST-recommended method [15][16]. These commands run with higher access to the storage device at the firmware level. The "enhanced secure erase" claims to remove data even from the spare blocks with the help of the flash controller.

This method will be incompatible with most IoT devices, where cost-effective storage manufacturers do not prioritize security compliance. It depends on the device interface and manufacturer [17]. SCSI and UAS-interfaced devices are not supported.

Variables: ($partition - device partition dev/sda*, $strongp - password, $name - device name)

` *sudo hdparm --sanitize-status $partition* `

Cmd. 1: Compatibility check for hdparm

` *sudo hdparm --user-master u --security-erase-enhanced $strongp $partition* `

Cmd. 2: ATA enhanced secure erase

[1][Online]. Available: https://github.com/SuchitReddi/memorywipe

Cryptographic Wipe:

We came up with this procedure to compensate for the incompatibility of firmware-based sanitization. This method is compatible with all devices, irrespective of the manufacturer. The storage device is encrypted with a strong password. We used VeraCrypt [19] for encryption because it supports all major operating systems, including Raspberry Pi. It can be useful for IoT devices as it might support other devices running on ARM architecture like Pi.

```
sudo veracrypt -t -c --volume-type=normal $partition --encryption=aes --hash=sha-512 --filesystem=ntfs -p $strongp --pim=0 -k "" --random-source=/dev/random
```

Cmd. 3: Encryption using VeraCrypt

The encrypted device is then overwritten with one pass each of pseudorandom values and zeroes, using the `dd` command (Data Definition). Multiple passes will decrease the probability of data recovery but will have a negative impact on the drive's lifespan. The device is finally formatted to a usable filesystem.

```
sudo dd if=/dev/random of=$partition bs=1M status=progress && sudo dd if=/dev/zero of=$partition bs=1M status=progress && sudo mkfs.ntfs -L $name $partition
```

Cmd. 4: Wiping process

When the whole drive is encrypted, it will appear as a single large file of garbage data. While overwriting the entire drive with random values, the NAND flash controller might mark some blocks as unmanaged. This might leave some original/encrypted data in the spare area, so the whole disk is overwritten again with a pass of zeroes. This drastically reduces the chance of recovering a complete unencrypted file in a readable format.

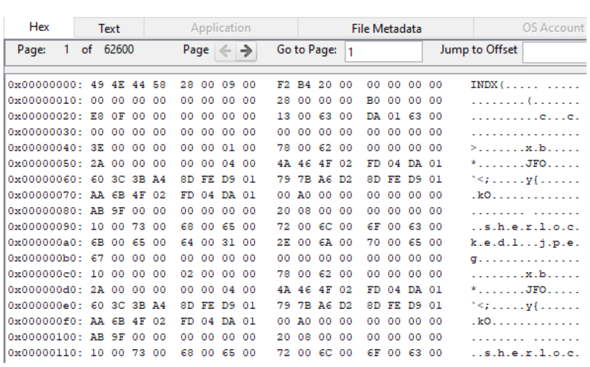

Fig. 1: After OS-level deletion

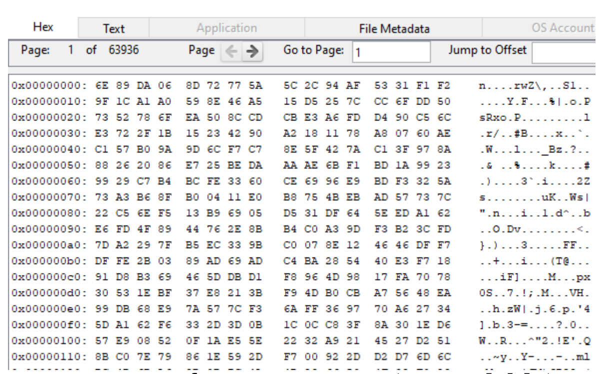

Fig. 2: After Encryption (Unreadable)

II. Verification

Tools used:

We used an 8GB SD card, a 32GB USB thumb drive, a 1TB HDD, and a 1TB SSD for testing various sanitization processes. Limited equipment and testable storage drives slowed the testing process and closed some paths altogether. To extensively test these sanitization functions, the required equipment is internal and external SSDs with different interfaces compatible with the SANITIZE feature set necessary for performing firmware-based sanitization.

Fig 3. Setup

But verification for successful sanitization on a NAND flash storage cannot be 100% certain when overprovisioning area comes into the picture. For absolute certainty that no data can be extracted, a physical chip read may be required to scan the OP blocks. Special equipment and some kind of documentation/tools from the manufacturer are needed.

The next best option is to use some forensic tool used by law enforcement that is capable of extracting incriminating evidence from storage devices. Some of the best out there are Magnet Axiom and Cellebrite UFED, but their subscriptions cost a fortune. So, we used Autopsy [20], a free, open-source tool that can run on Linux, Windows, and MacOS. Using `dd`, we extracted images from the target storage device, which was connected to an IoT device.

A command-line forensic recovery tool can verify successful sanitization on the same device where sanitization takes place. Scalpel or PhotoRec are such tools that can be incorporated into the tool in the future. This tool will be discussed in detail in further chapters.

Process:

1. Take a binary image of the storage device before starting the sanitization process. This command copies the entire drive, including the unallocated empty space.

```
sudo dd if=$partition of=/<output location>/image.bin status=progress
```

Cmd. 5: Imaging a drive using dd

2. Send this image to the device running Autopsy and add the disk image as a data source.
3. Autopsy will show all the files in the binary image, including those recently deleted, using os-level deletion, which just removes the file pointers.
4. At different steps of the process, take images of the device using the dd utility.
5. When the image taken after sanitization is loaded into Autopsy, no file should be recoverable if the sanitization process is successful.

III. Results:

Cryptographic Wipe:

Cryptographic wipe is compatible with most devices irrespective of the manufacturer, unlike firmware-based sanitization. This method was successfully tested on all the devices, i.e., SD cards, USB thumb drives, HDDs, and SSDs. All these devices were successfully sanitized, and the personal data from before the process was unrecoverable.

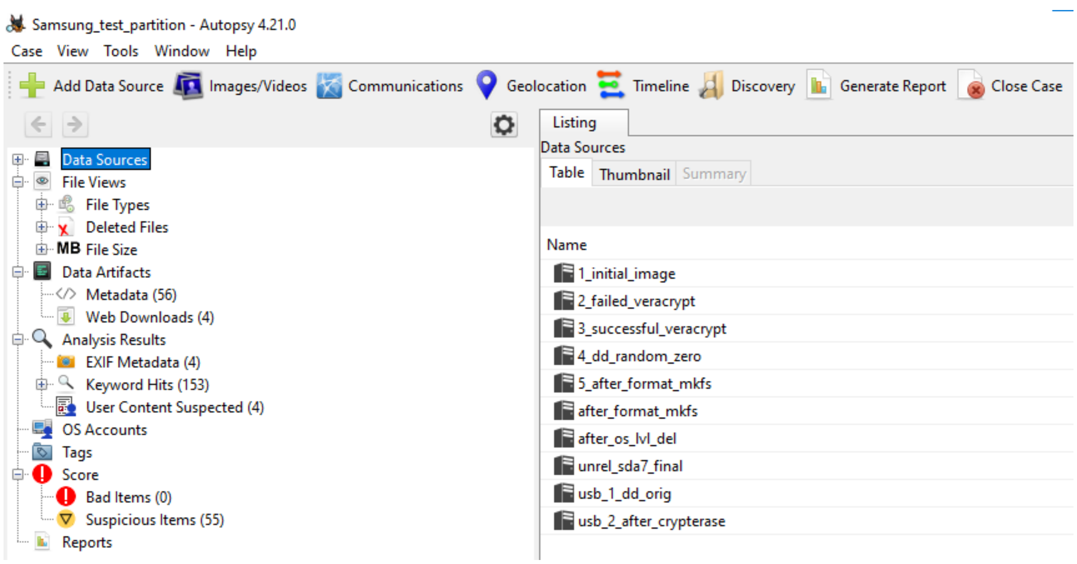

Fig. 4: Images extracted for testing

However, we will focus on the results of the SSD, which uses NAND flash storage. The verification process in detail is given below:

1. The SSD was connected to the laptop with Autopsy on it, and its image was taken through Autopsy first, which was stored as `1_initial_image`. It consisted of various folders, as shown in the figure below.

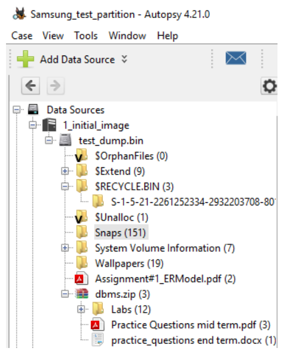

Fig. 5: Files in the initial SSD image

2. The same SSD was connected to the Raspberry Pi, and an image was taken via the Pi.
3. These images were successfully compared to ensure that Autopsy was error-free.
4. To test if a normal operating system level delete operation actually removes files, an image `after_os_lvl_del` was taken after deleting all folders using the `rm` command. All deleted files were recovered successfully, which emphasized the need for sanitization.

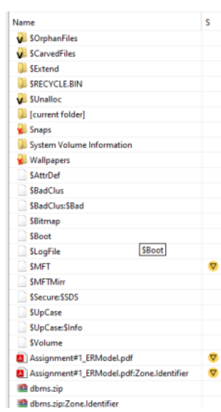

Fig. 6: Recovered folders (red cross at the bottom)

5. VeraCrypt was installed on the Pi, and encryption was attempted while the drive was still mounted, which gave a failure message. This was stored as `2_failed_veracrypt`.
6. The drive was unmounted, and encryption was attempted again. This time, it was successful, and the extracted image was stored as `3_successful_veracrypt`. This image contained a single unallocated file with unreadable values from the first page to the last.

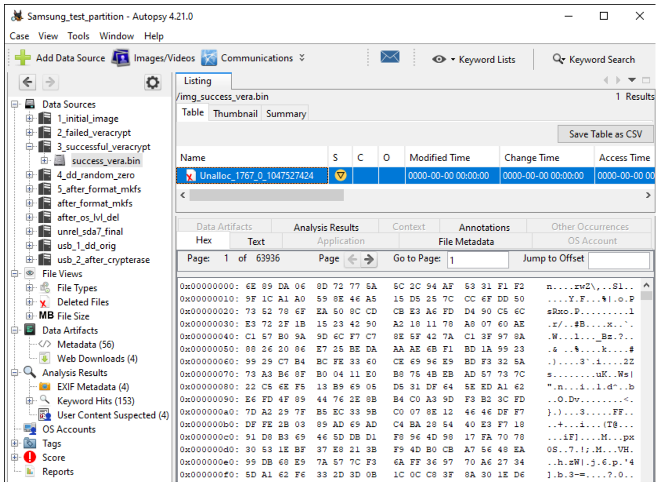

Fig. 7: After encryption

7. The encrypted drive was overwritten with one pass of random values from the source /dev/random. This should result in all blocks being marked invalid, erased, and overwritten. There is a chance for OP spare blocks to be swapped with encrypted blocks.
8. So, we write a pass of zeros over the random data again, forcing the controller to either swap OP spare blocks with random data blocks or erase the existing blocks.
9. A drive will only have a small percentage of spare blocks. So, the flash controller must clear most of the drive even after swapping spare blocks with random or encrypted blocks. The extracted image at this step was stored as `4_dd_random_zero`.

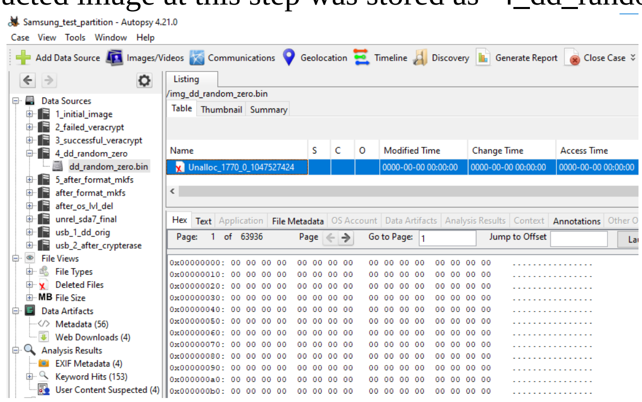

Fig. 8: After one pass, each of random data and zeros

10. After the passing of zeros, the drive will be unreadable without a file system. So, it is formatted. Recovering personal data from unmanaged blocks among all the encrypted, random, and zeroed blocks in the overprovisioning area will be practically impossible.
11. With this step, the cryptographic wipe process is complete. So, the final image was saved as `5_after_format_mkfs`. This final image only contained the filesystem but no recovered files before the cryptographic wipe.

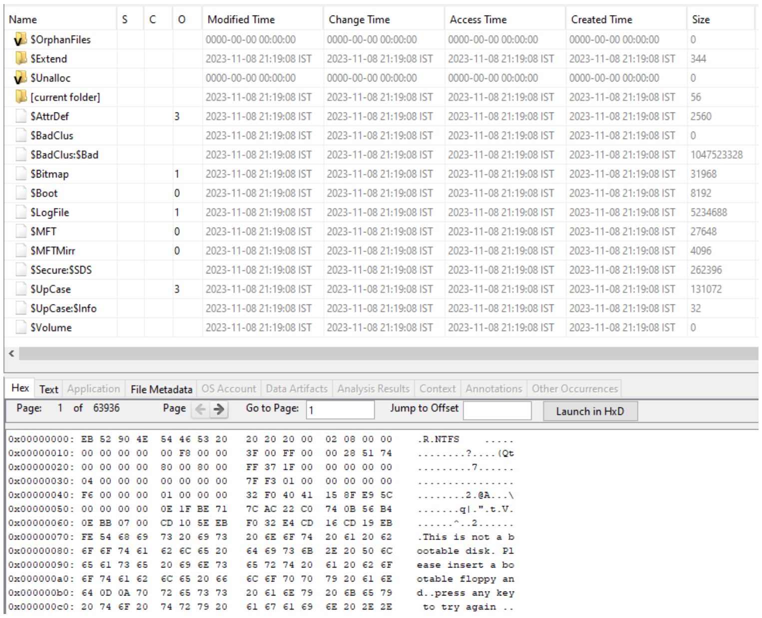

Fig. 9: Final image of SSD

12. This step verifies the successful deletion of personal information from the SSD.

Firmware-based:

This process works only if the SANITIZE feature set is enabled for a storage device. We tested these commands on two notable storage devices: a Samsung T7 1TB SSD interfaced via SCSI (external) and a Toshiba MQ04ABF100 1TB HDD via SATA (internal). The SATA-interfaced device was compatible, while the SCSI-interfaced device was not, which was expected. As the compatible device is a personal drive still in use and was tested by mistake in the first place, further tests were not conducted on it. However, the ATA_hdparm() function we developed for our tool has been tested by others and was reported to work without errors.

Fig. 10: Firmware sanitization compatibility (`supported: enhanced erase` should appear)

IV. Software sanitization tool:

As discussed in this paper, we developed a command-line utility to perform sanitization, extraction, and verification. It is named `memorywipe` and can be found here [21]. It has options for manual and automatic execution, with minimal user interaction. The sanitization techniques discussed in this paper were executed in this tool.

```
┌──(sherl0ck㉿pi 14:35)-[~]
└─$ bash memorywipe.sh

Available actions:
[1] Sanitization (Wiping)
[2] Extraction (Imaging)
[3] Verification (External process)
Select your purpose: 1

You selected Wiping
Available Methods: [Default: 5]
[1] Cryptographic Wipe
[2] ATA Secure Erase (hdparm)
[3] SATA Secure Erase (sg-utils)
[4] NVMe Secure Erase (nvme-cli)
[5] Automatic wipe (Executes the best compatible method)
Select your wiping method:
```

Fig 11. Memorywipe - sanitization methods

Once the user selects a method, the tool checks for the required programs and installs them if they are absent. If a known failure occurs at any point, it displays the reason and necessary tips to solve it. It also lets the user know the best way to apply a sanitization technique.

```
Select your wiping method: 1

You selected Cryptographic Wipe
Starting Cryptographic Wipe using VeraCrypt...

Checking for existing veracrypt installation...
veracrypt is installed.
Check complete...

WARNING: This process is irreversible. Create necessary backups if required (You may use the extraction module for this).

NOTE: Cryptographic Wipe on Flash Storage partitions may not be effective. Perform it on the whole disk.
If there are multiple partitions in the device, format them into one single partition, and then wipe it.

Do you want to see all the partitions?(y/n)[n]:
```

Fig. 12: Memorywipe - existing installation checking

The tool can even take users with little technical knowledge along the process by abstracting the commands executed and automating wherever required.

```
Do you want to see all the partitions?(y/n)[n]: y
NAME        FSTYPE FSVER LABEL   UUID                                 FSAVAIL FSUSE% MOUNTPOINTS
sda
└─sda1      ntfs         testusb 55C9623B7C5C3D61                       28.6G     0% /media/sherl0ck/testusb
mmcblk0
├─mmcblk0p1 vfat   FAT32 bootfs  B888-51D8                             397.3M    22% /boot/firmware
└─mmcblk0p2 ext4   1.0   rootfs  3c215345-c59c-4a79-941a-ab121e090a42    8.1G    37% /

Select /dev/sda, if you want to wipe the whole drive, partitioned as sda1, sda2, ..., sdaN
Enter your device's partition (/dev/sda1): /dev/sda1

Unmounting the partition...
/dev/sda1 has been successfully unmounted.

Do you want to encrypt manually or automatically? (m/a)[a]: a
Uses AES, with SHA-512 and makes an NTFS filesystem

Enter strong password for encrypting. (You don't have to remember it)
So set a random password with special characters: 108dklancdi*sdt^*(5#ej*JgF69(746
```

Fig. 13: Memorywipe - sanitization (Cryptographic Wipe)

```
Enter strong password for encrypting. (You don't have to remember it)
So set a random password with special characters: 1923jhaHdYT6*(f4=%&hf$8kr54^0ld
Finished encrypting /dev/sda1

Wiping /dev/sda1.
Unmounting the partition...
/dev/sda1 already unmounted
/dev/sda1 overwritten with one write each of random data and zeroes
Set device name: testdevice
Wiped /dev/sda1

Mounting /dev/sda1 at /media/testdevice
mount: (hint) your fstab has been modified, but systemd still uses
       the old version; use 'systemctl daemon-reload' to reload.
Finished mounting!

Cryptographic Wipe procedure completed successfully!
```

Fig. 14: Memorywipe - sanitization successful

The whole program is a shell script where operations are performed using a combination of different functions.

```
memorywipe.sh

crypt_wipe() {
    echo "Starting Cryptographic Wipe using VeraCrypt..."
    echo
    ins_veracrypt #Script should not proceed if this step fails.
    echo

    echo "WARNING: This process is irreversible. Create necessary ba
    echo
    echo "NOTE: Cryptographic Wipe on Flash Storage partitions may r
    echo "If there are multiple partitions in the device, format the
    echo
    list_partitions

    # Setting a value for partition (/dev/sdb1) to wipe
    echo
    echo "Select /dev/sda, if you want to wipe the whole drive, part
    read -p "Enter your device's partition (/dev/sda1): " partition
    echo

    # Making sure disk is unmounted
    if chk_unmount; then
      echo
    else
      echo "Unmounting failed!! Please unmount manually before proce
      echo
      return 1
    fi

    # Start the process
    veracrypt_encrypt
    echo
    wipe_disk
    echo
    mount_disk
    echo
    echo "Cryptographic Wipe procedure completed successfully!"
}
### --- Cryptographic Wipe ends here ---

### ---ATA Secure Erase starts here ---
# Installing hdparm
ins_hdparm() {
```

Fig. 15: Memorywipe - a glimpse of source code

Chapter 4
Conclusion

Our exploration into secure sanitization provided an understanding of its significance and methodologies. We emphasized the requirement of disk storage sanitization for IoT devices. Differences in the working of older mechanical drives and modern drives utilizing NAND flash were highlighted. The need for a sanitization technique that can work on devices like SSDs was acknowledged and addressed.

Different techniques and their working were explored, and the most compatible ones with IoT devices were discussed in this paper. A method for verifying the success or failure of sanitization techniques was discussed and implemented. The detailed results of the sanitization process and verification were shown.

Moreover, our efforts continued beyond theoretical analysis. We took a proactive step by implementing the discussed techniques as a command-line utility developed using shell scripting. This helped in converting our research into a real-world solution that the general public can use to sanitize their storage devices, giving them the power to protect themselves.

Research into a more certain sanitization and verification by utilizing hardware techniques is very much needed. With the disturbing increase in cybercrime in recent years, there is a strong need to stay ahead in the relentless pursuit by maintaining data security and thorough sanitization. Through our project, we have not only highlighted the challenges faced but also presented a viable solution to improve personal data security.

Chapter 5

Future Prospects

1. The memory wipe tool in development can be further improved by adding better sanitization, extraction, and verification methods, which induces low wear to chips.
2. Incorporating terminal-based verification methods (PhotoRec, SleuthKit's Scalpel).
3. Preparing tools specific to widely used OS, device types, interfaces, etc.
4. Automating the tools to remove user interaction for IoT devices with limited access.
5. Exploring sanitization possibility for IoT devices without shell access.
6. Possibility of sanitizing storage on mobiles using ADB command interface.
7. Performing hardware-based bit and block-wise operations on NAND chips to improve the effectiveness of the sanitization process.
8. Factory Access Mode is similar to rooting a phone and allows low-level command execution but is limited to the manufacturers and service centers [22].
9. The Flash transition layer might allow low-level access without special hardware equipment. If it can, researching this can improve the certainty of sanitization.
10. `mtd` devices seem to support these low-level operations, where an image including unmanaged blocks can be extracted using the below command.

` nanddump -s $partition --bb='dumpbad' -p -n `

Cmd. 6: Unmanaged block image using mtd device

(a) (b)

Fig. 16: Hardware-level sanitization (a) Connections (b) Hack board for PoS

References

[1] S. Bennett and J. Sullivan, "NAND Flash Memory and Its Place in IoT," 2021 32nd Irish Signals and Systems Conference (ISSC), Athlone, Ireland, 2021, pp. 1-6, doi: 10.1109/ISSC52156.2021.9467859.

[2] Sanvido, Marco & Chu, Frank & Kulkarni, Anand & Selinger, Robert. (2008). NAND Flash Memory and Its Role in Storage Architectures. Proceedings of the IEEE. 96. 1864 - 1874. 10.1109/JPROC.2008.2004319. M. N. DeMers, Fundamentals of Geographic Information Systems, 3rd ed. *New York: John Wiley*, 2005.

[3] Blog post. Single package SSDs reduce size of IoT devices. [Online] Available from: https://www.electronicspecifier.com/products/memory/single-package-ssds-reduce-size-of-iot-devices [Accessed 26th November 2023]

[4] Blog post. Target Breach. [Online] Available from: https://www.securityweek.com/target-confirms-point-sale-malware-was-used-attack/ [Accessed 26th November 2023]

[5] Wei, Michael & Grupp, Laura & Spada, Frederick & Swanson, Steven. (2011). Reliably Erasing Data from Flash-Based Solid-State Drives. 105-117.

[6] Xiaolu Zhang and Kim-Kwang Raymond Choo. 2019. Digital Forensic Education: An Experiential Learning Approach (1st. ed.). Springer Publishing Company, Incorporated.

[7] Kanekal, Vasu. 2013. “Data Reconstruction from a Hard Disk Drive using Magnetic Force Microscopy.”

[8] Ahn, Na Young & University, Dong. (2022). Security of IoT Device: Perspective Forensic Anti-Forensic Issues on Invalid Area of NAND Flash Memory. IEEE Access. 10. 10.1109/ACCESS.2022.3190957.

[9] Blog Post. Solid-State Drive (SSD) File Recovery Challenge. [Online] Available from: https://perez-aids.medium.com/solid-state-drive-ssd-file-recovery-challenge-cbde1935e33a [Accessed 26th November 2023]

[10] Blog Post. (2023). Data Sanitization Methods. [Online] Available from: https://www.lifewire.com/data-sanitization-methods-2626133 [Accessed 26th Nov 2023]

[11] Jeong Wook, Matt Oh. (2020). Reverse Engineering Flash Memory for Fun and Benefit.

Available from: https://www.blackhat.com/docs/us-14/materials/us-14-Oh-Reverse-Engineering-Flash-Memory-For-Fun-And-Benefit.pdf [Accessed 26th Nov 2023]

[12] FT2232H NAND flash reader. [Online] Available from: https://spritesmods.com/?art=ftdinand&page=1 [Accessed 26th November 2023]

[13] Blog Post. Dumping a SLC NAND Flash with Atmel PMECC. [Online] Available from: https://www.mickaelwalter.fr/dumping-a-slc-nand-flash-with-atmel-pmecc/ [Accessed 26th November 2023]

[14] Murza, Yoshi. (2018). Reading Nand Flash with Raspberry Pi 3 b. [Online] Available from: https://www.youtube.com/watch?v=hHkBseIblBM [Accessed 26th November 2023]

[15] Regenscheid, A., Feldman, L., & Witte, G. (2015). Pg-44, NIST Special Publication 800-88 Revision 1, Guidelines for Media Sanitization (No. ITL Bulletin February 2015). National Institute of Standards and Technology.

[16] Computer Security Resource Center (CSRC). SANITIZE Command. Available from: https://csrc.nist.gov/glossary/term/sanitize_command [Accessed 26th November 2023]

[17] Jon Tanguy. Data Sanitation: Securely Erasing Micron SATA SSDs. [Online] Available from: https://www.micron.com/-/media/client/global/documents/products/technical-marketing-brief/brief_ssd_secure_erase.pdf?la=en [Accessed 26th November 2023]

[18] Blog Post. Securely wipe an SSD with its built-in commands. [Online] Available from: https://code.mendhak.com/securely-wipe-ssd/ [Accessed 26th November 2023]

[19] Veracrypt. [Online] Available here: https://veracrypt.fr [Accessed 26th November 2023]

[20] Autopsy. [Online] Available here: https://www.autopsy.com [Accessed 26th Nov 2023]

[21] Memorywipe. [Online] Available here: https://github.com/SuchitReddi/memorywipe [Accessed 26th November 2023]

[22] Blog Post. Life after Trim: Using Factory Access Mode for Imaging SSD Drives. [Online] Available from: https://blog.elcomsoft.com/2019/01/life-after-trim-using-factory-access-mode-for-imaging-ssd-drives/ [Accessed 26th Nov 2023]