ST. PÖLTEN UNIVERSITY
OF APPLIED SCIENCES

Evil Maids Hate This Trick

Protecting Unattended Computers From Physical Access Attacks

Diploma thesis

for attainment of the academic degree of

Diplom-Ingenieur

submitted by

Hassan Mohamad

is202801

in the

University Course Information Security at St. Pölten University of Applied Sciences

Supervision

Advisor: FH-Prof. Dipl.-Ing. Dr. Robert Luh, BSc

Declaration

Title: **Protecting Unattended Computers From Physical Access Attacks**

Type of thesis: **Diploma thesis**

Author: **Hassan Mohamad**

Student number: **is202801**

I hereby affirm that

- I have written this thesis independently, have not used any sources or aids other than those indicated, and have not made use of any unauthorized assistance.
- I have not previously submitted this thesis topic to an assessor for evaluation or in any form as an examination paper, either in Austria or abroad.
- this thesis corresponds with the thesis assessed by the assessor.

I hereby declare that

- ☒ I have used a Large Language Model (LLM) to proofread parts of the thesis.
- ☐ I have used an LLM to generate portions of the content of the thesis. I affirm that I have cited each generated sentence/paragraph with the original source. The LLM used is indicated by a footnote at the appropriate place.
- ☐ no LLM has been used for this work.

Date *Signature*

Abstract

Full Disk Encryption (FDE) is the predominant solution for protecting the sensitive data on our computers from unauthorized physical access. However, while strong encryption can prevent an attacker from directly accessing the data, it cannot prevent more sophisticated physical access attacks. Techniques such as cold boot attacks, which extract encryption keys from memory, and evil maid attacks, which manipulate devices to hijack user passwords, highlight the limitations of FDE. Designing effective countermeasures against these vulnerabilities requires a thorough understanding of the underlying attack vectors.

This thesis presents a comprehensive review of the available literature on physical access attacks and possible countermeasures. The complex interactions between attacks and countermeasures are then examined to evaluate the effectiveness of different solutions. Based on these findings, a guideline for hardening Linux computers against physical access attacks is developed. The process of implementing this guideline, from hardware and software choices to setting up physical tamper detection, is then documented to provide a blueprint for others seeking to enhance their system's security.

Following this guideline resulted in a Linux computer hardened against most of the physical access threats identified. However, this process revealed vulnerabilities that cannot be addressed with current solutions, which prompted proposals for future research to improve defenses against physical access attacks.

Kurzfassung

Festplattenverschlüsselung ist die vorherrschende Lösung, um die sensiblen Daten auf unseren Computern vor unbefugtem physischen Zugriff zu schützen. Starke Verschlüsselung kann zwar einen Angreifer daran hindern, direkt auf die Daten zuzugreifen, sie kann jedoch keine anspruchsvolleren physischen Angriffe verhindern. Techniken wie Cold-Boot-Angriffe, bei denen Schlüssel aus dem Speicher extrahiert werden, und Evil-Maid-Angriffe, bei denen Geräte manipuliert werden, um Passwörter zu stehlen, zeigen die Grenzen von Festplattenverschlüsselung auf. Die Entwicklung wirksamer Gegenmaßnahmen gegen diese Schwachstellen erfordert ein umfassendes Verständnis der zugrundeliegenden Angriffsvektoren.

Diese Arbeit bietet einen umfassenden Überblick über die verfügbare Literatur zu physischen Angriffen und möglichen Gegenmaßnahmen. Anschließend werden die komplexen Wechselwirkungen zwischen Angriffen und Gegenmaßnahmen untersucht, um die Wirksamkeit der verschiedenen Lösungen zu bewerten. Basierend auf diesen Erkenntnissen wurde ein Leitfaden zur Absicherung von Linux-Rechnern gegen physische Angriffe entwickelt. Der Prozess der Implementierung dieses Leitfadens, von der Auswahl der Hard- und Software bis hin zur Einrichtung physischer Manipulationserkennung, wurde anschließend dokumentiert, um anderen, die die Sicherheit ihrer Systeme verbessern wollen, eine Vorlage zu bieten.

Die Umsetzung dieses Leitfadens resultierte in einem Linux-Computer, der gegen die meisten der identifizierten Bedrohungen durch physischen Zugriff geschützt ist. Es wurden jedoch auch Schwachstellen identifiziert, die mit den derzeitigen Lösungen nicht behoben werden können. Dies veranlasste Vorschläge für zukünftige Forschungsarbeiten zur Verbesserung der Abwehr von Angriffen durch physischen Zugriff.

Contents

1 Introduction

In today's digital age, the security of our computer systems is paramount. Much of our attention is focused on defending against remote threats—from email-delivered malware to network-based intrusions—because these account for the majority of incidents [1]. But there is a physical dimension to security that should not be overlooked: the vulnerability of unattended computers to direct, hands-on attacks. One of the primary technologies used to protect the valuable data on our computers from such attacks is Full Disk Encryption (FDE). Strong encryption can certainly prevent an adversary from directly accessing the data on the disk, but many attacks have emerged that circumvent this security measure. These attacks range from cold boot attacks [2], where the encryption keys are extracted from memory, to evil maid attacks [3], [4], where the device is manipulated to steal the user's password when they enter it in the future.

To defend against these physical access attacks, we must first understand their underlying mechanisms by studying the available offensive security literature. We then turn our attention to appropriate countermeasures and evaluate their effectiveness against these attacks. Armed with this knowledge, we present a comprehensive hardening guide for Linux-based systems, highlighting not only the strengths of available security solutions, but also their weaknesses. Finally, we outline possible directions for further study, setting the stage for enhanced security measures against physical access threats.

1.1 Threat Model and Assumptions

A clear threat model is essential because it provides the framework for our discussions in this thesis. It also serves as a foundation for readers to understand the context, scope, and assumptions at the core of this thesis. The target in our threat model is a desktop or laptop computer, that uses FDE to secure user data when at rest. We assume that the user is reasonably security-conscious and uses strong passwords for FDE and local Operating System (OS) user accounts. We also assume that the user always activates the OS lock screen or turns off the device before

leaving it unattended.

The attacker is an individual or entity that wants to access the encrypted data and can gain physical access to the device. Modes of access can range from *covert*, such as unauthorized entry into a user's hotel room to access devices, to *overt*, such as seizure by law enforcement or during border control in hostile territories. Some modes may grant the attacker only brief, one-time physical access, while others may allow multiple, prolonged interactions with the device. Both the duration and frequency of these physical interactions, combined with the configuration of the device, affect the range of attack vectors available to the adversary.

We will consider attacks of varying degrees of sophistication. While the simplest attacks can be performed by anyone with basic computer skills, the most advanced attacks require expensive equipment and special expertise. These attacks can be further categorized based on the user interaction required:

- *Drive-by or Single-stage Attacks*: These can be performed in a single visit and are therefore applicable to stolen devices. The adversary requires no further user interaction and, if successful, gains direct access to the encrypted data. This can be accomplished by bypassing the lock screen on an active device, or by extracting the encryption key from Random-Access Memory (RAM) or the Trusted Platform Module (TPM).
- *Multi-stage Attacks*: These are performed in multiple stages and require unsuspecting users to inadvertently provide the secret keys or passwords needed to decrypt the data. In the first stage, the attacker needs physical access to covertly tamper with the device, either through software or hardware, so that the encryption key or passphrase is captured when the user unlocks their device in the future. Once the user enters the required secret to unlock the device, it is either transmitted remotely to the attacker or retrieved when the attacker gains physical access for a second time.

In general, the attacker will prefer the former category of attacks because it does not rely on the user's failure to detect that their device was manipulated. To keep the focus of our discussion concise, our threat model has some clear limitations:

- Breaking the cryptographic primitives used for FDE, such as Advanced Encryption Standard (AES) and Secure Hash Algorithm (SHA), is considered beyond the attacker's capabilities.
- Brute-forcing the disk encryption key or passphrase will not be considered, as this is a general limitation of low-entropy passwords and not specific to physical access attacks.

- The possibility of factory pre-installed backdoors is not considered.
- We exclude "rubber-hose cryptanalysis" scenarios where users are forced to unlock their device due to physical or legal threats.
- Non-physical attacks, such as malware delivered via email or browser exploits, are not considered.

1.2 Contributions

This thesis assesses the susceptibility of unattended computers to physical access attacks by analyzing publicized research that aligns with our threat model. Additionally, it examines both purely academic and readily available countermeasures. It then draws on these insights to build a hardened Linux-based system and identifies areas where solutions are lacking. The following are the main contributions:

- *Survey of Known Attacks and Countermeasures*: Publications in the areas of Direct Memory Access (DMA), cold boot, Self-Encrypting Drives (SEDs), TPM, and evil maid attacks, along with their countermeasures, were reviewed to build a comprehensive understanding of the security landscape.
- *Evaluation of Countermeasure Effectiveness*: The complex interactions between attacks and countermeasures can be obscure. This thesis evaluates the effectiveness of readily available countermeasures against the most important attacks to shed light on this situation.
- *Hardening Guideline*: A guideline for choosing and configuring the right hardware and software to build a hardened Linux-based setup is presented.
- *Highlighting Remaining Weaknesses*: The process of implementing the hardened Linux setup revealed several weaknesses. This prompted proposals for future research to improve defenses against physical access attacks.

1.3 Related Work

This thesis builds on the author's previous work from 2021 that focused on the offensive aspect of physical access threats [5]. The previous study laid the groundwork by creating a framework for determining the most viable physical access attacks given certain conditions. By drawing on the insights of that study and incorporating the developments of the subsequent two years, this thesis aims to create a guideline for *hardening* devices against these threats.

In 2015, Müller and Freiling published a paper with an objective related to chapter 3 of this thesis. In their study, they systematically assessed the attack vectors of FDE solutions for computers and smartphones. Their research operated under a threat model similar to ours, where an attacker gains physical access to an unattended device with FDE and seeks to extract the encrypted data. Since then, offensive security research has emerged that challenges or refutes the effectiveness of their recommendations.

The 2017 dissertation by Götzfried explored strategies for creating trusted systems in untrusted environments [7]. While his threat model differs from ours, he also considered physical access attacks. Götzfried's work relied heavily on solutions that are academic prototypes. In contrast, this thesis focuses on countermeasures that are readily available for ordinary computers.

Heads is a project focused on hardening Linux systems against a wide range of physical access attacks [8]. However, it is only available for a small selection of notebooks because it requires firmware modifications. Other work in this area has focused on smaller aspects of system security, such as securing the boot process. The *safeboot* project aims to reduce the attack surface of the Linux boot process using Secure Boot, TPM-based attestation, and FDE [9].

1.4 Thesis Outline

This thesis is divided into three main chapters:

- *Background*: Chapter 2 covers some essential technological prerequisites for understanding the rest of the thesis.
- *Physical Access Attacks and Countermeasures*: In chapter 3, we take an in-depth look at physical access attack vectors, drawing on the existing literature and discussing mitigation strategies.
- *Hardening Guideline*: Equipped with this knowledge, chapter 4 walks through the process of building a hardened Linux-based setup that will resist most of the discussed attack vectors.

2 Background

To effectively navigate the upcoming chapters, a fundamental understanding of technologies related to FDE and broader platform security is necessary. This chapter provides a brief overview of these basics, supplemented by references for readers seeking further information.

2.1 Secure Boot

Secure Boot is a security feature that was introduced in version 2.3.1 of the Unified Extensible Firmware Interface (UEFI) specification [10], [11]. It uses cryptographic hashes and signatures to verify Extensible Firmware Interface (EFI) binaries, such as drivers and bootloaders, before they are executed. If a binary fails this verification, Secure Boot blocks its execution. The specification defines a set of certificates and databases that are stored in EFI variables. These are used to determine if a binary is allowed to start:

- Platform Key (PK): Acts as the root of trust, used for signing updates to the Key Exchange Key (KEK*), Signature Database (*db*), and Forbidden Signature Database (*dbx*), not for signing executables.
- Key Exchange Key (KEK*): Are used for signing updates to the *db* and *dbx*, but can also sign executables.
- Signature Database (*db*): Acts as the allow list database, containing certificates and hashes that are allowed to boot.
- Forbidden Signature Database (*dbx*): Acts as the deny list database, containing certificates and hashes that are blocked. The *dbx* has ultimate veto power.

These databases usually come pre-populated with a Platform Key (PK) and KEK* from the hardware vendor, as well as a KEK* and two *db* certificates from Microsoft. One of the *db* certificates is from Microsoft's Windows production Certificate Authority (CA), which is used to sign the Windows bootloader, kernel, and drivers. The second is from Microsoft's UEFI third-party marketplace CA, which is used to sign third-party UEFI drivers and Linux bootloaders. Additionally,

the *db* may come with hashes of onboard controller firmware that are explicitly allowed. The *dbx* typically comes pre-populated with hashes of known vulnerable EFI executables, malware, and revoked certificates [12, p. 8].

Most Secure Boot implementations have several modes of operation that can be configured in the UEFI settings. *Standard Mode* is the default configuration for computers that ship with Windows preinstalled. In this mode, signature and hash checks are enforced using the certificate and hash databases discussed earlier. Switching to *Setup Mode* clears the current PK, suspends signature verification, and allows the user to enroll new certificates. Once a new PK is enrolled, Secure Boot switches to *Custom Mode*, which re-enables signature and hash verification.

2.2 Trusted Platform Module

The Trusted Platform Module (TPM) is a specification developed by the Trusted Computing Group (TCG), a computer industry consortium, and later standardized under ISO/IEC 11889 [13]. It defines a secure coprocessor that acts as a passive device within its host platform, providing a set of cryptographic functionalities via a simple command-response protocol. These functionalities include cryptographic primitives, secure key generation/storage, and status registers that can be used to attest platform integrity. The specification does not specify the form in which the TPM is implemented; implementations range from discrete tamper-resistant chips to entirely software-based solutions. Relevant for our use case are discrete TPMs (dTPMs), which the TCG considers to be the most secure type [14], and firmware TPMs (fTPMs). The latter are implemented as software running within the host's Trusted Execution Environment (TEE), which is usually a dedicated secure part of the main processor.

In our context, a primary use case for the TPM is to aid in measuring the integrity of boot components. This is a process where each component in the boot chain computes a cryptographic hash of the next component before executing it. The first component to initiate the measurement chain is called the Core Root of Trust for Measurement (CRTM). Because it forms the foundation of the trust chain, this component must be inherently tamper-resistant and trustworthy. This is typically ensured by immutable code stored in Read-Only Memory (ROM), which acts as the CRTM. During the boot process, the CRTM runs first and measures the integrity of the subsequent component, which is typically the system firmware. The system firmware does the same before loading the next component, and the process continues until all critical components have been measured.

Table 2.1: PCR usage as specified by the TCG [15].

PCR Index	PCR Usage
0	Static Root of Trust for Measurement (SRTM), core system firmware, embedded option ROMs
1	Host platform configuration
2	UEFI driver and application code (option ROMs)
3	UEFI driver and application configuration and data
4	UEFI bootloader code
5	Bootloader configuration and GPT/partition table
6	Host platform manufacturer specific
7	Secure boot policy including PK, KEK*, *db*, and *dbx*
8–15	Defined for use by the static OS

To protect these measurements from manipulation, the TPM has a series of Platform Configuration Registers (PCRs) designed to store hash values. When a system boots up, all PCRs are initialized to their default value of all zeros or all ones. Once a PCR's value has been changed, it can only be reset to its default value by rebooting the host system. A fundamental security property of these registers is that their values cannot be set arbitrarily; they can only be updated with an *extend* operation. When *extending* a measurement into a PCR, the following formula is used: $PCR_{new} := H_{alg}(PCR_{old} \parallel digest)$. The TPM concatenates PCR_{old} with the new measurement's $digest$ and applies the hash algorithm H_{alg} to derive PCR_{new}.

The concept of PCRs underpins several key features of the TPM, such as *remote attestation* and *sealed storage*. The latter is particularly important in the context of this thesis, as it is commonly used for FDE. When data is sealed with the TPM, it is encrypted with one of the TPM's secret keys. During this process, a policy is defined that specifies the conditions under which the data may be unsealed. For example, the policy may specify that password authorization is required or that specific PCR values must be present to unseal the data. This mechanism is invaluable for FDE because it ties access to the disk encryption key to a trusted system state. Table 2.1 lists the intended use for each PCR index as defined by the TCG in a supplementary specification [15].

2.3 Full Disk Encryption

The goal of Full Disk Encryption (FDE) is to secure data at rest against unauthorized access by encrypting every block on the disk or partition. Symmetric block ciphers, such as AES, are

used to encrypt the data with a Data Encryption Key (DEK). The DEK itself is encrypted with a Key Encryption Key (KEK) and stored in a key slot within the FDE header on the disk. A KEK can be derived from several sources, including a password, a smart card, a Universal Serial Bus (USB) key, or a TPM (see section 2.2). There can be multiple key slots, one for each specific KEK, each containing an encrypted version of the same DEK. This general design is used in most disk encryption schemes due to its flexibility. It allows multiple unlocking methods to be used, and allows those methods to be changed without re-encrypting the data. When changing the password, for example, only the encrypted DEK in that specific key slot has to be replaced.

Microsoft's BitLocker is a popular example of an FDE scheme that follows these design principles [16]. In BitLocker terminology, the DEK is called Full Volume Encryption Key (FVEK) and the KEK is called Volume Master Key (VMK). The FVEK is encrypted with the VMK and stored in the BitLocker volume header. Depending on the unlock method in use, multiple instances of the VMK are stored in the volume header, each protected/encrypted in a certain way. Among others, BitLocker offers the following modes for unlocking a disk [16]:

- *TPM-only*: This mode requires no user interaction to unlock the drive. BitLocker protects the VMK by sealing it with the TPM so that it cannot be unsealed unless the measured boot process succeeds (see section 2.2). This mode relies on the operating system's user authentication scheme to protect the data, sacrificing security for convenience.
- *TPM + PIN*: In this mode, BitLocker asks for a PIN code, which is then hashed and used as additional authentication data to unseal the VMK with the TPM. The TPM has anti-hammering protection to prevent brute-force attacks on the PIN. Despite the use of the word PIN, BitLocker allows the use of complex alphanumerical passwords.
- *TPM + Startup Key*: Instead of a PIN, a high entropy key file stored on a USB drive can be used as a second factor with the TPM. The TPM key and startup key are XORed together to form the VMK.
- *Recovery key*: If the measured boot process fails or there are too many incorrect PIN entry attempts, the TPM will no longer unseal the VMK. For this reason, a recovery key is generated during the BitLocker initialization process. An instance of the VMK is encrypted using only this recovery key and is not dependent on the TPM. If all else fails, the recovery key can be used to unlock the drive.

2.3.1 Software-based FDE

Solutions such as BitLocker (Windows), dm-crypt (Linux), FileVault (macOS), and VeraCrypt use the Central Processing Unit (CPU) to encrypt every block before writing it to disk. To reduce the performance overhead of using FDE, these tools use AES New Instructions (AES-NI), an extension to the x86 instruction set implemented by many modern CPUs. While this addresses the performance concern, software-based FDE solutions face two major security issues. First, the encryption key typically needs to be held in RAM while the device is running, making it vulnerable to cold boot and DMA-based attacks aimed at extracting the key. Second, parts of the disk, such as the boot sector or boot partition, must remain unencrypted to initiate the bootstrapping process that unlocks the rest of the drive. This opens the door for malicious code to be injected into the boot process, a class of malware known as a *bootkit*.

2.3.2 Self-Encrypting Drives

SEDs implement encryption transparently within the drive itself, either on the disk controller or a dedicated coprocessor. This avoids retaining the DEK in system memory and allows encryption of the entire drive, including the boot sector. This was thought to solve the two problems of software-based FDE mentioned so far. Earlier models repurposed the ATA security feature set, which was intended as a password-based access control mechanism that did not necessarily require encryption. Later, the TCG released the Opal SSC, a new standard for SEDs that mandated the use of AES encryption and introduced advanced key management capabilities [17]. For drives that implement the Opal SSC 2.0 and IEEE 1667 standards, BitLocker supports offloading encryption to the dedicated hardware [18]. Microsoft refers to drives that meet this specification as *Encrypted Hard Drives* or *eDrives*.

2.4 ACPI States

Originally released by Intel, Microsoft, and Toshiba in 1996, the Advanced Configuration and Power Interface (ACPI) is a widely adopted open standard for power management and hardware configuration [19]. On ACPI-compliant systems, all power events trigger an interrupt event that is handled by the operating system. These events, such as closing a laptop lid or pressing the power button, prompt the operating system to start the appropriate power management procedure, such as the sleep or shutdown sequence. ACPI defines several power states that indicate

whether a device is running, in a power-saving state, or powered off. ACPI states S0, S3, S4, and S5 are particularly relevant for physical access attacks, as they directly affect the attack surface of a device.

- S0: The system is fully running, but may be locked by the OS lock screen. In this state, software-based FDE keys are stored in RAM and SEDs are unlocked.
- S3: Commonly referred to as *sleep mode* or *suspend-to-RAM*, this state saves power by turning off most components except RAM, therefore fully preserving the system's context. In this state, software-based FDE solutions keep the keys in RAM and while SEDs are powered off, they are automatically unlocked when switching back to S0 [20], [21].
- S4: Known as *hibernate* or *suspend-to-disk*, this mode preserves the system's context by writing the RAM contents to a special partition or file on disk called *swap space*. It then powers down all system components to save power, and requires the FDE password to return to S0. If the swap space is encrypted, this mode is practically equivalent to S5 in terms of attack surface.
- S5: Except for the power supply, the system is completely powered off in this state and requires the FDE password to boot to S0.

It is important to note that most device and OS configurations allow a potential attacker to switch between S0 and S3. This can usually be done by opening or closing the laptop lid, pressing the dedicated buttons, or using the controls found on the lock screens of most operating systems. The same holds true for the more recently introduced power saving state known as *Modern Standby* on Windows or *S2Idle* on Linux. Since switching between these power states does not require a password, S0, S3, and S2Idle are practically equal in terms of attack surface when physical access is involved. An attacker will simply put the device into whichever state is more vulnerable.

3 Physical Access Attacks and Countermeasures

Adversaries seeking to bypass FDE have an array of physical attack vectors at their disposal. Defending against these threats requires a deep understanding of the vulnerabilities and attack mechanisms involved. In this chapter, we will take an in-depth look at the attack vectors available to these adversaries and the potential countermeasures. Our discussion will draw on the existing literature, and assess the fundamental vulnerabilities that enable these attacks. While our primary focus will be on the current landscape of attacks and countermeasures, we will also delve into historical contexts when they provide relevant insights. As we navigate these topics, we will continuously align our discussion with our threat model. By the end of this chapter, readers will have a thorough understanding of the physical access attack landscape, the countermeasures available to mitigate them, and their significance within our threat model.

3.1 Methodology

The following structured methodology was used to systematically evaluate the vast landscape of physical access attacks:

1. *Information Sources and Search*: Scholarly search engines, primarily Google Scholar and Semantic Scholar, were queried for targeted keywords such as *Cold Boot*, *DMA*, *evil maid*, *TPM*, *FDE*, *BitLocker*, *LUKS*, *keylogging*, *hardware implants*, *Secure/Measured/Trusted Boot*, in combination with the words *attack*, *vulnerability*, *countermeasure*, and relevant synonyms.
2. *Inclusion Criteria*: Preliminary search results were filtered based on their relevance to our threat model defined in section 1.1. Attacks and countermeasures were excluded if they were either not applicable to our threat model or became irrelevant due to significant technological advances since their discovery.

3. *Information Collection*: Eligible publications were thoroughly reviewed to extract the relevant information on attacks and countermeasures. Furthermore, the references of the initial publications and the subsequent publications that cited them were examined. This recursive method allowed for a comprehensive understanding of the topic.
4. *Additional Sources*: Recognizing the niche nature of the subject, the search was not limited to peer-reviewed publications. After gaining a solid understanding of the subject through the work in step 3 and reviewing the core parts of relevant technical specifications, the search was extended to the broader web. This included blogs and conference presentations by recognized industry experts, both individuals and companies. These sources were subjected to the same inclusion/exclusion criteria and additional scrutiny, such as verifying their claims in available source code or vendor vulnerability disclosures.

Throughout the research process, new attack vectors relevant to this thesis occasionally emerged that were not initially considered. Once identified, they were included into the search keywords and subjected to the previously described steps. Through this iterative and methodical approach, this chapter aims to provide a comprehensive perspective on physical access attacks, the vulnerabilities they exploit, potential countermeasures, and their effectiveness.

3.2 DMA Attacks

Direct Memory Access (DMA) is what facilitates fast communications with minimal CPU overhead over the PCI Express (PCIe) bus. Modern computers rely heavily on DMA for all types of PCIe connected devices, such as storage and USB controllers, network adapters and graphics cards. Using DMA, these devices can transfer data directly to and from system memory without much CPU involvement, thus drastically improving data rates and latency. This technology is not only available to internal devices through interfaces such as Peripheral Component Interconnect (PCI) and PCIe, but also to externally connected devices through the use of FireWire, PCMCIA and Thunderbolt.

Direct access to system memory becomes a security concern when considering malicious devices. These could be regular system components with compromised firmware or devices that were designed from the ground up by adversaries to target systems through DMA. Back in 2004, Dornseif was the first to demonstrate the dangers of DMA [22]. An Apple iPod with modified firmware was used to attack a computer connected via FireWire. Using DMA to read arbitrary memory regions of the targeted computer, Dornseif was able to retrieve screen contents, read se-

cret key material, and create full memory dumps. Write access was leveraged to inject code into processes or perform privilege escalation.

Over the years, DMA attacks have been demonstrated via several interfaces, such as FireWire [23]–[25], CardBus [26], [27], PCIe [28], [29] and Thunderbolt [30]–[32]. In 2016, Frisk presented *PCILeech*, a DMA attack platform that supports all major operating systems (Windows, Linux, macOS, and FreeBSD) and various Field Programmable Gate Array (FPGA)-based PCIe development/research boards [31]. Using PCILeech, an attacker can read and write system memory arbitrarily and therefore run malicious code with kernel privileges, bypass the OS lock screen or extract the FDE keys. In our threat model (refer to section 1.1), this makes DMA attacks a serious threat if the computer is either running or in sleep mode. If FDE is configured to unlock without a user-supplied secret (i.e., TPM-only mode), the computer is vulnerable to DMA attacks regardless of its ACPI state.

3.2.1 DMA-capable Interfaces

To properly understand the attack surface of a device, it is important to identify all interfaces that support DMA. The following DMA-capable internal interfaces are/were the most common:

- *PCI slots (deprecated)*: Multiple PCI slots used to be found on desktop and server mainboards. Nowadays, one such slot might be available for backwards compatibility with legacy devices.
- *MiniPCI (deprecated)*: The PCI-equivalent for laptops, rarely found anymore.
- *PCIe slots*: Modern desktop and server mainboards commonly offer multiple PCIe slots, which are used for all kinds of expansion cards.
- *MiniPCIe (deprecated)*: The PCIe-equivalent for laptops, rarely found anymore.
- *PCIe M.2*: Very common in laptops, where they are used for storage (SSDs), Wi-Fi and WWAN. Modern desktop and server mainboards often have M.2 as well, although primarily for storage.

While desktops and servers commonly have unoccupied PCI/PCIe slots into which a malicious device could be inserted, this is rarely the case in laptops. However, this does not mean that occupied slots are safe, as an attacker might be able to hot swap a PCIe device without causing the system to crash. In laptops, this has been successfully demonstrated by replacing an M.2-connected Wi-Fi card with a malicious device during sleep mode [29]. This consideration is not necessary for external DMA capable interfaces, as these are designed to be hot-pluggable:

- *FireWire (deprecated)*: The first interface demonstrated to be vulnerable to DMA attacks, rarely found anymore.
- *CardBus (deprecated)*: Also known as 32-bit PC Card (formerly PCMCIA), was used for connecting various expansion cards to laptop computers.
- *ExpressCard (deprecated)*: The successor of CardBus.
- *Thunderbolt*: Versions 1 and 2 used the Mini DisplayPort connector, while current versions use USB-C.
- *USB4*: Based on the Thunderbolt 3 protocol, also uses USB-C.

Even though all these interfaces *may* be vulnerable due to their support for DMA, that does not have to be the case. Depending on the interface, there are mechanisms such as port authorization and memory segmentation that aim to mitigate such vulnerabilities.

3.2.2 Port Authorization

Starting with version 2, *Thunderbolt Security Levels* have been introduced into the Thunderbolt standard [33]. This is an access control feature that aims to guard against malicious peripherals, by requiring the user's approval to initialize Thunderbolt devices. The standard specifies four security levels, ranging from *no security* (SL0) to *disabling PCIe tunneling* (SL3). Some devices support SL4, which allows PCIe tunneling only for the first device in the chain. The more interesting levels are SL1 and SL2, which are based on the Trust On First Use (TOFU) principle. When a new Thunderbolt device is plugged in, the user is asked to *deny*, *allow once* or *always allow* access by that device. In the case of SL1, the system keeps track of the device's Universally Unique IDentifier (UUID), once it has been permanently authorized or rejected. SL2, on the other hand, uses a cryptographic challenge-response procedure to verify the authenticity of the device, which promises to be more resistant to spoofing attacks.

In 2020, Ruytenberg published a report describing seven vulnerabilities that completely break the security claims of the Thunderbolt specification [34]. Among other weaknesses, Ruytenberg demonstrated that the UUID of a Thunderbolt device can be spoofed to emulate an already trusted device and gain DMA to the target system. Even worse, the SPI flash, where the Thunderbolt controller stores its firmware and configuration, was found to be insufficiently protected and re-programmable by an attacker with physical access. This allowed the author to modify the Thunderbolt firmware or disable security levels altogether. Furthermore, to allow backward compatibility with Thunderbolt 2 devices, the Thunderbolt 3 implementations tested were found

to completely disregard the configured security level, automatically allowing DMA to legacy devices. Ruytenberg argues that, since most of the vulnerabilities discovered are part of the Thunderbolt 3 specification, virtually all devices are vulnerable. Intel responded to this publication by recommending Input-Output Memory Management Unit (IOMMU)-based protection against DMA attacks and made it part of the Thunderbolt 4 certification program [35]. Similarly, the Windows certification program for USB4 requires that the device ships with IOMMU-based protection [36].

3.2.3 IOMMU-based Protection

The Input-Output Memory Management Unit (IOMMU) plays a big role in modern operating systems' defense strategies against DMA attacks. It is the part of the memory controller that translates physical memory addresses to I/O Virtual Addresses (IOVAs) in a process called DMA remapping (DMAr). This allows the OS to restrict a device's DMA capability to a confined memory region, similarly to how the Memory Management Unit (MMU) is used to restrict memory access by processes. To accelerate the translation process, the IOMMU features an Input-Output Translation Lookaside Buffer (IOTLB) that caches the most recently used translations. Except for the most low-end configurations, most modern computers have an IOMMU, although it is usually marketed using a different term. In the case of Intel, it is part of their Virtualization Technology for Directed I/O (VT-d) [37] feature set, whereas AMD offers it as part of AMD-Vi [38]. Apple's own processors feature an IOMMU as well, as described in their platform security document [39, p. 63]. On its own, the IOMMU does not provide any protection against DMA attacks. It is the responsibility of the operating system and its drivers to utilize DMAr to protect against DMA attacks. The availability and effectiveness of IOMMU-based DMA protection therefore highly depends on the operating system and platform in general.

Windows 10 introduced support for DMAr for Thunderbolt devices in version 1803 and extended it to all PCIe devices in version 1903 [40]. Microsoft refers to this feature as *Kernel DMA Protection* and states that it does not protect against DMA attacks via FireWire, CardBus and ExpressCard [41]. Further, Microsoft states that only certain drivers support DMAr, allowing the corresponding devices to utilize DMA automatically. The behavior for handling devices with DMAr-incompatible drivers can be configured using a group policy. By default, these devices will blocked from performing DMA until an authorized user signs in to the system. While *Kernel DMA Protection* started as an opt-in feature, newer devices are now beginning to ship with it enabled,

not least because of the requirements posed by the Thunderbolt 4 and USB4 certifications [35], [36].

In other operating systems, the state of DMAr support is similarly complicated. Linux kernel support for DMAr has been available since 2006 for Intel and 2007 for AMD CPUs [42] and can be configured via a set of kernel parameters [43]. In systems with an Intel CPU, it is opt-in, unless the kernel was compiled with a certain configuration option [44]. Because of bug reports[1], most Linux distributions chose not to enable it by default on Intel systems [32]. Systems with AMD-Vi, on the other hand, use the IOMMU by default on Linux if AMD-Vi is enabled in the Basic Input/Output System (BIOS) [43]. Apple's macOS is the only OS that has been shipping with DMAr enabled since 2012 [42, p. 56], [32].

3.2.4 IOMMU-resistant Attacks

The erratic support for DMAr via the IOMMU in operating systems and drivers is not the only challenge when it comes to protecting against DMA attacks. As far back as 2009, research started showing that implementation issues can allow attackers to bypass the IOMMU [45], [46]. These early findings do not apply to the threat model of this thesis because they describe an attack that requires full access to a Virtual Machine (VM) running on the target computer. However, other ways to bypass the IOMMU were discovered in the following years that do fit our threat model. In 2016, Morgan *et al.* demonstrated an attack that allowed them to add an entry to the DMAr table during the early Linux boot stages, before DMA is governed by the IOMMU [47]. This effectively bypassed the IOMMU and granted full memory access to the malicious device. Since then, Apple and Microsoft have introduced measures to block such vulnerabilities [32]. In 2017, Rothwell described a new attack that bypasses the IOMMU by using PCIe's Address Translation Services (ATS) feature [42, p. 97]. ATS is a performance optimization that allows PCIe devices to handle IOMMU translations themselves instead of relying on the system's IOMMU. Rothwell demonstrated how a malicious device can advertise ATS support to the operating system and can then perform DMA without any restrictions. In response to this vulnerability, Intel's introduced a patch in Linux version 4.21 to disable ATS for Thunderbolt devices [32]. On Linux, ATS can also be disabled altogether through a kernel parameter [43].

There are several weaknesses beyond the IOMMU bypass vulnerabilities described so far. In 2016, Markuze *et al.* theorized about two fundamental vulnerabilities that exist because of how

[1] `https://bugs.launchpad.net/ubuntu/+source/linux/+bug/1971699`

the IOMMU is commonly used [48]. First, the IOMMU works at page granularity (4 KiB) only and cannot offer the byte-level protection that the DMA API specifies, creating a *spatial*, also called *sub-page*, vulnerability. This means that a device can access any memory that is located on the same page that one of its DMA buffers begins or ends on, potentially exposing sensitive kernel memory. Second, most operating systems invalidate the cached IOMMU translations asynchronously, which is important for performance reasons, but leads to potential *temporal* vulnerabilities. Deferred invalidation creates a small window of time during which a device still has access to the DMA buffer after it has been unmapped. Sensitive data could now be written to the buffer and become accessible to the device until invalidation takes place. Temporal vulnerabilities can be avoided by reverting to synchronous invalidation of IOMMU translations, at the aforementioned cost in performance.

In 2017, Rothwell demonstrated that the spatial vulnerability can be exploited to overwrite kernel function pointers and gain arbitrary code execution. Temporal vulnerabilities can be exploited as well, as demonstrated by Kupfer *et al.* in 2018 [49]. A year later, Markettos *et al.* introduced *Thunderclap*, a PCIe attack platform which they used to evaluate ATS, spatial, and temporal vulnerabilities against all major operating systems and across different hardware [32]. Their work showed that all major operating systems were vulnerable to DMA attacks, leading to information leakage and code injection, despite utilizing the IOMMU for DMA remapping.

In 2021, Markuze *et al.* published their extensive research on characterizing, exploiting and detecting IOMMU-resistant DMA attacks in Linux [50]. They identified four types of sub-page vulnerabilities and three vulnerability attributes that are required for code injection attacks. Additionally, they published a static code analysis tool and a runtime analysis tool to systematically identify these vulnerabilities in the Linux kernel and drivers. Using these tools, they found that 72% of device drivers expose callback pointers, allowing a malicious device to perform code injection attacks. Prior work on sub-page vulnerabilities demonstrated code injection attacks that require certain conditions to be met, such as an exposed kernel pointer. Markuze *et al.* introduce *compound attacks*, which are multi-stage DMA attacks that use various strategies to create the necessary conditions for a code injection attack.

3.2.5 Summary and Outlook

DMA attacks have been known for almost two decades and are still at least as relevant of a threat as ever. Virtually all computers have DMA-capable interfaces and, with the growing adoption

of Thunderbolt 4 and USB4, more and more of these interfaces are accessible externally. OS and hardware makers are aware of this threat and have implemented countermeasures based on port authorization schemes and DMA remapping. However, the research presented in this section highlights that these countermeasures are severely lacking and cannot protect against more advanced attacks.

There is no shortage of proposals for preventing these vulnerabilities. Notably, Markuze *et al.* proposed a new way of using the IOMMU in 2016 that uses distinct pages for each device and copies data in and out of these pages upon DMA request [48]. Their solution eliminates the spatial and temporal vulnerabilities discussed in this chapter, but comes at a considerable cost in performance. They later addressed some of these performance issues, although only for a specialized networking use case [51]. Other solutions, both hardware- and software-based, have been proposed [52, pp. 19–21], [42, pp. 105–117], but they are beyond the scope of this thesis. Until these countermeasures evolve from academic proof-of-concepts to mature safeguards, DMA attacks must be considered a serious risk in our threat model—one that cannot be prevented with current technology alone. As a consequence, users who need the utmost level of protection against DMA attacks, need to avoid leaving their devices unattended unless they are powered off (S5) or in hibernate mode (S4).

3.3 Cold Boot Attacks

While it is generally assumed that RAM contents vanish once the system is powered off, the *memory remanence* effect has been known for decades [53]–[55]. Instead of immediately losing their charge, memory cells will slowly fade towards their ground state if they are not refreshed. This effect can be prolonged to several minutes by sufficiently cooling the memory modules [2], [55].

Security implications of the memory remanence effect have been theorized early on, but it took until 2008 for practical attacks to be demonstrated by Halderman *et al.* [2]. Their renowned paper demonstrates practical cold boot attacks targeting numerous popular disk encryption solutions (BitLocker, TrueCrypt, dm-crypt, and FileVault). All these software-based FDE solutions store the encryption key in main memory while the computer is running or in sleep mode. The cold boot attacks they describe exploit this property to extract the disk encryption key in three steps:

1. Gain physical access to the target computer while it is running or in S3 sleep.
2. Acquire an image of the system memory by

a) either rebooting the machine from a USB drive with a specialized memory imaging tool
b) or transplanting the memory modules to a machine controlled by the attacker.

3. Analyze the memory image, identify encryption keys and reconstruct keys in case of bit errors.

Halderman *et al.* documented their results on six laptops with different hardware configurations (DDR1 and DDR2 memory). They found that even without cooling, the memory modules would retain most of their data for several seconds before the error rate became too high to recover usable data. This could be prolonged to a few minutes by cooling the memory with the refrigerant found in readily available compressed air cans. When these cans are used in an upside-down position, the contained refrigerant leaves the can in liquid form, evaporates rapidly and cools the surface of the memory chips to temperatures as low as $-50\,°C$.

They created a bootable tool with a minimal memory footprint for creating a memory image. If the BIOS configuration allows it, this tool can be booted directly on the target machine via PXE network boot or USB. Otherwise, the RAM modules had to be quickly transplanted to an attacker-controlled machine, while providing sufficient cooling. Whenever possible, the former method is preferred because it reduces the memory's chance to decay. In the case of a soft reboot, the memory is continuously receiving power, which completely avoids decay.

Arguably the most important part of their research was the development of *keyfind* and *keyfix*, a set of tools that identify encryption keys in memory images and correct potential bit errors that occurred because of memory decay. Instead of looking for the DEK itself, they search for the pre-computed key schedule in memory, which is a set of permutations of the DEK that are part of the symmetric encryption algorithm. Depending on the encryption algorithm, 10 to 16 round keys exist in the key schedule and can be used as a form of error correction code. With *keyfind* and *keyfix*, they were able to defeat BitLocker, TrueCrypt, dm-crypt, and FileVault FDE.

3.3.1 Modern Cold Boot Attacks

In an effort to reproduce the findings described so far, Gruhn and Müller conducted a trial with 17 laptops, including systems with more modern DDR3 based memory [56]. For systems with DDR1 and DDR2 memory, their results were largely consistent with those from Halderman *et al.*, but they were unable to recover any data from DDR3 based systems in case of a hard reboot or memory transplantation. It was later determined, that *memory scrambling* is the reason for this

complication with DDR3 memory [57]. The memory controller on DDR3 and DDR4 systems does not store the data in plain text but rather XORs it with a pseudo-random keystream [58]. This was never intended as a security feature, as it does not use a cryptographically secure source of random data. Instead, the memory scrambler aims to create a more uniform distribution of ones and zeros, reducing current fluctuations and improving signal integrity on high-speed memory buses [59]. Further research has shown that these scramblers offer no meaningful protection against cold boot attacks on DDR3 [57] and DDR4 [58] based systems.

In 2018, Segerdahl and Saarinen have demonstrated some improvements to the practicability of cold boot attacks, that avoid transplanting memory and thus the complications of memory scrambling [60]. To accomplish this, they had to circumvent a security feature that was specifically designed to mitigate reboot-based cold boot attacks. Modern UEFI firmware implements the *TCG PC Client Platform Reset Attack Mitigation Specification* [61]. It specifies the Memory Overwrite Request (MOR) bit, which can be set by the OS to inform the firmware about the presence of secrets in memory. Per default, the MOR bit is set to induce a memory-overwrite by the firmware. If the OS performs a clean shutdown procedure that includes overwriting secret keys in memory, it sets the value of the MOR bit to indicate that no secrets need to be overwritten by the firmware. However, the MOR bit was found to be improperly secured because it is stored in an unprotected region of the surface-mounted SPI flash chip on the mainboard. The exposed pins of this UEFI chip allow easy reading and reprogramming of its contents in a non-destructive way by using a test clip. By doing so, an attacker can disable the memory-overwrite functionality and change any firmware settings that would prevent booting a memory imaging tool to conduct a cold boot attack. With this technique, they were able to avoid memory transplantation and the effects of memory scrambling on DDR4 systems. The TCG has since released version 1.10 of their *PC Client Platform Reset Attack Mitigation Specification*, which includes a locking mechanism for the MOR bit [62]. This mechanism aims to protect against unauthorized modification of the MOR bit by the OS kernel [63]. It is unclear if this revision of the standard protects against physical attacks through the SPI interface.

3.3.2 CPU-bound Encryption

Early strategies for mitigating some of the risks of cold boot attacks involve the utilization of *CPU-bound encryption*. The general idea is to avoid storing any cryptographic key material over long periods of time in system memory, and instead utilize CPU registers for key storage. In

2011, Müller *et al.* introduced TRESOR, an implementation of this concept in the form of a Linux kernel module [64]. It is an extension to the kernel's crypto API that implements AES and stores the encryption key in the debug registers of the CPU. By utilizing AES-NI, TRESOR can perform encryption without ever storing cryptographic key material in RAM.

Unfortunately, TRESOR requires two patches to the Linux kernel that hinder its mainstream potential. The first patch ensures that the encryption of a block of data is an atomic operation. This is crucial because if the CPU scheduler were to perform a context switch during encryption, the CPU registers containing key material would be swapped to system memory. The second one blocks user-space access to the CPU's debug registers, as those are used to store encryption keys. This has the consequence of blocking the use of hardware breakpoints or watchpoints, features often utilized by debuggers. The authors of TRESOR argue that this is not an issue for regular users and developers, because debuggers can emulate them in software.

In 2012, *TreVisor* was introduced in an effort to improve the security and compatibility of CPU-bound encryption [65]. It is based on *BitVisor*, a thin hypervisor that enables transparent disk encryption and DMA protection for its single guest operating systems [66]. By implementing these security controls transparently on a hypervisor level, the need for OS-specific patches is eliminated. TreVisor effectively combines the security benefits of TRESOR and BitVisor, enabling an FDE scheme that is resilient against cold boot attacks as well as DMA based attacks.

Even with OS-agnostic solutions such as TreVisor, CPU-bound encryption still has a major flaw. Cold boot attacks can not only target disk encryption keys, but also other sensitive information that resides in RAM. This may include application-specific secret keys or browser data such as user credentials and payment information. Memory encryption solutions can mitigate these attacks by making use of CPU-bound encryption schemes to encrypt system memory. These solutions can be software- [67]–[70] or hardware-based [71]–[74].

3.3.3 Software-based Memory Encryption

In 2016, Götzfried *et al.* introduced *RamCrypt*, a Linux kernel patch that transparently encrypts the address space of user-mode processes [67]. It is based on TRESOR and therefore operates without relying on system memory for encryption and secret key storage. While it protects the memory of all user-mode programs transparently, RamCrypt cannot be used to encrypt kernel memory. As an effort to eliminate this restriction and avoid patching the OS, *HyperCrypt* was created [68]. Similar to TreVisor, it is based on BitVisor but instead implements RAM encryption

transparently for the entire guest OS. While HyperCrypt simplifies OS compatibility and allows encryption of kernel memory, performance remains a concern.

A different approach to memory encryption was taken by Zhao and Mannan when designing *Hypnoguard* [69]. Instead of aiming to encrypt RAM while the system is running, Hypnoguard only encrypts memory when switching to sleep mode. By using AES-NI, the authors claim to be able to perform full memory encryption in under one second for a computer with 8 GB of RAM. The encryption key is sealed in the system's TPM, where it cannot be read by a potential attacker. Upon returning from sleep mode, Hypnoguard requires a user password as well as a valid integrity measurement by the TPM to release the key. A similar approach to memory encryption was taken by Franzen *et al.* in 2020 with their solution called *FridgeLock* [70]. Their goal was to implement everything as a Linux kernel module, which reduces the amount of work necessary to maintain FridgeLock in future kernel versions. The security guarantees of Hypnoguard and FridgeLock depend on the assumption that users will switch into sleep mode whenever leaving their device unattended. This makes them primarily suitable for laptop and desktop computers, whereas servers are incompatible with this approach.

3.3.4 Hardware-based Memory Encryption

A possible option to overcome the limitations of the software-based solutions mentioned so far are hardware-based memory encryption technologies. Intel released Software Guard Extensions (SGX) in 2015, a set of CPU instructions that allow user-mode processes to create an encrypted memory region, called a *secure enclave* [71], [72]. When a program creates a secure enclave, it provides trusted code that runs in the enclave and has exclusive access to the protected memory region. The CPU ensures that even high-privileged kernel or hypervisor code is unable to access protected enclave memory. While the primary use-case for SGX is process isolation, enclave memory is also protected against physical access threats, such as cold boot attacks. Nonetheless, SGX has at least three critical limitations that prevent its use for full RAM encryption: (1) applications have to be rewritten to use SGX, (2) only user-mode processes can utilize SGX and (3) the total size of all enclaves is limited to 128 MB. Götzfried demonstrated that the second limitation can be overcome, allowing SGX to be used for disc encryption, although at a significant performance cost [7].

Thankfully, CPU manufacturers have caught on and introduced hardware-based memory encryption technologies specifically intended to protect against cold boot attacks. Unlike SGX,

AMD's Secure Memory Encryption (SME) works with existing applications and only requires setting a special bit in the x86 page table to enable encryption for that specific memory page [73]. There is also Transparent Secure Memory Encryption (TSME), a feature controlled through a BIOS setting that transparently encrypts all memory with a key randomly generated at boot time and requires no special support by the OS. A study by Mofrad *et al.* shows that the performance impact of full memory encryption with SME is minimal for most workloads [75]. Intel has since introduced a competing technology called Total Memory Encryption (TME) [74]. While the availability of TSME and TME is still limited to higher-end or business-oriented CPUs, they are available widely enough to be a feasible solution for security conscious users.

3.3.5 Summary

The memory remanence effect of dynamic RAM still poses a threat to modern computers by exposing sensitive information such as encryption keys to cold boot attacks. Reboot-based cold boot attacks circumvent the added complexity of memory scrambling introduced with DDR3 and work on systems using soldered memory. The MOR specification designed to mitigate this type of attack has been shown to be ineffective, as it relies on UEFI functionality that can be manipulated by an attacker [60]. However, even transplantation-based cold boot attacks are still viable on newer systems, because memory scrambling does not offer any cryptographic protection and can be reversed [57], [58].

Various software-based memory encryption solutions have been introduced in academia, each with their set of shortcomings in terms of security, compatibility, and performance [64], [65], [67]–[70]. They have not seen any widespread adoption beyond academia, presumably because the demand for cold boot protection was not high enough to bear these burdens. Hardware-based memory encryption solutions, such as AMD's TSME [73] and Intel's TME [74], seem to be the most promising options, as they can work transparently with existing operating systems and have a minimal performance impact.

3.4 Attacks on Self-Encrypting Drives

When Self-Encrypting Drives (SEDs) started to become widely available, they were believed to offer better security than software-based FDE solutions [76]. This was because traditional DMA and cold boot attacks, targeting the encryption key in memory, were mitigated by implementing

disk encryption separately from the main memory and CPU. SEDs also promise to offer protection against malware targeting the boot sector or bootloader because they encrypt the entirety of the disk, whereas software-based FDE requires an unencrypted bootloader. However, since their emergence, new attacks and flaws have been discovered, which challenge the supposed security benefits of SEDs.

3.4.1 Hot Plug Attacks

In 2012, Müller *et al.* introduced a novel attack that allows an attacker to defeat the encryption provided by SEDs if the target system is running or in sleep mode [20]. When a system with a SATA-based SED is powered on, the user is asked for a password to unlock the drive. The drive remains unlocked as long as it is powered on, or until it is explicitly locked by issuing a specific ATA command. An attacker can exploit this fact by disconnecting the SATA cable from a running SED and connecting it to their own computer while keeping the power connection intact. This way, the SED remains unlocked and the attacker gains access to the decrypted data, without knowledge of the password. Initially, this attack was tested with SEDs utilizing the ATA Security standard, but it has since been confirmed to be effective against Opal SSC-based SEDs [21].

While unplugging the SATA connection without cutting power is possible on desktop computers, most laptops use a single connector for both SATA and power, making this procedure significantly more difficult to perform on a running machine. There is, however, a way to circumvent this limitation by abusing another flaw in most SED implementations. Most laptops only require the password in the boot process, but unlock the disk automatically when resuming from sleep mode [6], [21]. The attacker can therefore remove the SED while the laptop is in sleep mode and install extension cables between it and the mainboard, to allow unplugging the SATA and power connection independently. When returning from sleep mode, the laptop will automatically unlock the SED and the attacker can proceed with the regular hot plug attack.

Some laptops protect against hot plug attacks by detecting that a drive has been disconnected in sleep mode and refusing to automatically unlock it if that is the case. Boteanu and Fowler circumvent this mechanism with a variant they call *hot unplug attack* that targets running laptops [21]. They describe sliding in a SATA data and power connector with exposed pins between the drive and mainboard contacts, therefore keeping the connection to the laptop intact. Then they supply power through the inserted connector and disconnect the drive from the victim's machine. This way, the drive will remain unlocked and the attacker gains access to the unencrypted

data by connecting the drive's SATA interface to their own machine.

Both papers [20], [21] conclude that effective countermeasures against hot plug attacks cannot be implemented by the host system, because the host can only lock the drive if the SATA connection is still intact. Instead, SED manufacturers would need to implement a feature that detects when the drive's SATA interface is disconnected, and subsequently locks the drive.

3.4.2 Forced Restart Attack

This is another attack that exploits the fact that SEDs will remain unlocked until powered off or explicitly locked. While Müller *et al.* refer to it as *cold boot attacks on SEDs* [20], Boteanu and Fowler call it a *forced restart attack* [21]. Most machines did not lock the drive when doing a soft-reset, thus allowing an attacker to reboot into another OS and access the unlocked SED. If the OS does not allow rebooting the machine while on the lock screen, a reboot can be forced through multiple means. This is done either by shorting two pins on the mainboard that are responsible for inducing a soft reset, or by triggering a blue screen on Windows, which was accomplished by fuzzing the USB bus or shorting two memory pins [21]. Of course, the target machine has to be configured to allow booting from alternative sources, such as USB, for this attack to be successful. Otherwise, the attacker will have to attempt to manipulate the necessary BIOS settings, which is not guaranteed to work, but has been demonstrated in the past [60]. Additionally, some machines do lock the SED when rebooting, but this can be subverted by briefly disconnecting the SATA data connection during the reboot [20].

3.4.3 Design and Implementation Flaws

A major problem in the realm of SEDs is that instead of a few well-tested software-based FDE implementations (BitLocker, dm-crypt, FileVault), there are many more hardware-based FDE implementations across all the different SED manufacturers. Inevitably, this will negatively affect the quality of these implementations, not least because they cannot be audited as easily as their software-based counterparts. In 2015, Alendal, Kison, *et al.* took a deeper look into the inner workings of a series of SEDs from one manufacturer [77]. They identified several vulnerabilities that result in the decryption of user data, without the knowledge of any user credentials. The models they analyzed were for external use and thus did not implement the TCG Opal SSC standard, unlike modern internal SEDs.

In 2019, Meijer and van Gastel discovered several security flaws affecting a wide range of SEDs

that do implement the Opal SSC standard [78]. They analyzed SEDs from various manufacturers by reverse engineering their firmware and found weaknesses that allow the recovery of all stored data without knowledge of the secret key or password. Their attack model focused on machines that are completely powered off, which is usually considered to be the best-case scenario for FDE. Nevertheless, they were able to exploit the majority of analyzed drives, due to specification, design, or implementation issues. Most attacks required some form of low-level control over the drive's storage controller, to access parts of the device that are not exposed through the SATA or NVMe interface. This was achieved by using the JTAG debugging interface or, in case JTAG was disabled, gaining arbitrary code execution on the storage controller by installing a modified firmware image. Their case studies of nine SEDs identified, among others, the following flaws:

- *DEKs not Derived from Secrets:* The Opal SSC standard does not mandate *if* and *how* the DEK should be derived from the password. To offer any meaningful cryptographic protection, the DEK must be derived from a secret that does not reside on the drive. Certain drives store all the necessary decryption secrets on the drive itself, using the password verification process solely for access control [78]. This allows an attacker to bypass the password verification and access the unencrypted data by manipulating the controller's memory.
- *Single DEK for the Entire Drive:* Opal SSC drives support multiple *locking ranges*, which can be locked or unlocked separately with their distinct password. Ideally, each range should be encrypted with a distinct DEK as well, but four out of nine tested drives failed to implement this correctly [78]. This is especially severe if BitLocker is used because an unprotected locking range is required for bootstrapping the system. Since the unprotected range shares the same DEK as other ranges, an attacker with low-level access to the storage controller can decrypt all the data.
- *ATA Master Password Re-enabling:* The ATA security standard includes a master password and a user password, both capable of unlocking the DEK. For effective data protection, users need to disable or change the default master password. However, certain drives failed to remove the DEK derived from the default master password when changing or disabling it [78]. Consequently, attackers with low-level access could decrypt the SED using the widely known default master password.
- *Wear Leveling Prevents Overwriting Secrets:* SSDs employ wear leveling to extend the lifespan of NAND flash cells by evenly distributing writes across the storage medium. For this purpose, the controller maintains a mapping of logical to physical sectors. Repeatedly writ-

ten logical sectors are typically redirected to different physical sectors by the drive controller. The original physical sector remains unchanged, with only the mapping adjusted accordingly. One of the analyzed drives uses the write-leveled NAND flash to store its internal key material, which allowed recovering the KEK to access all unencrypted data [78].

3.4.4 Summary

While SEDs might offer some benefits in terms of performance, their promised security benefits are overshadowed by several critical vulnerabilities. Hot plug attacks target systems that are running (S0) or in sleep mode (S3) and defeat the disk encryption entirely [20]. It is a practical attack that requires no special tools and has been demonstrated to work against all tested SATA-based SEDs [21]. It remains to be seen if NVMe-based SSDs are vulnerable as well.

The more recently discovered design and implementation flaws undermine the security of SEDs even if the system is completely turned off [78]. Bad cryptographic practices, as well as key management issues, allow an attacker to decrypt the data without knowledge of any secret key. Although these flaws are vendor- and model-specific and do not apply to all SEDs, they erode the trust in proprietary hardware encryption schemes that are difficult to audit. BitLocker used to rely entirely on hardware encryption by default for drives implementing the Opal SSC 2.0 and IEEE 1667 standards. However, these security issues prompted Microsoft to revert to using software-based encryption per default.[2] [3]

3.5 Attacks Against the TPM

FDE provides substantial security benefits but requires a strong password entered at startup, which can be an inconvenience. Especially in a time when biometrics are widely used to make security more seamless, there is a demand for FDE solutions that are less intrusive to the end-user. Because of their *sealing* functionality described in section 2.2, TPMs are commonly used to achieve this goal. Upon booting up, the integrity of a specific set of early boot components and configurations is measured using the TPM. If the verification succeeds, the TPM unseals the encryption key, allowing the system to boot up without any user interaction. Among major PC operating systems, Windows is the only one that heavily relies on the TPM for FDE. For this reason, most of this section addresses BitLocker's implementation of TPM-assisted disk encryption

[2] `https://portal.msrc.microsoft.com/en-us/security-guidance/advisory/ADV180028`
[3] `https://support.microsoft.com/en-us/help/4516071/windows-10-update-kb4516071`

and its weaknesses.

When the TPM is used to store the FDE key, it becomes a target for attacks that needs to be considered in the threat model of this thesis. There are significant differences in terms of attack surface between the different types of TPMs discussed in section 2.2. Despite dTPMs being considered the most secure type of TPM [14], they are vulnerable to a class of attacks that is mostly not applicable to fTPMs. These attacks target the communication bus between the dTPM and the host system. On the other hand, fTPMs are vulnerable to a distinct class of attacks targeting the Trusted Execution Environment (TEE) that they are running in. Being part of the CPU, the TEE also lacks the physical tamper resistance against fault injection, side channel, and invasive silicon attacks that dTPMs provide.

3.5.1 Data Bus Attacks Against discrete TPMs

While the TCG and dTPM manufacturers have put significant effort into hardening the security of the chip itself, the interfaces for communicating with the host system have become the primary targets for attacks. The host system typically communicates with the dTPM via data buses like SPI, I^2C, or Low Pin Count (LPC), which lack inherent security. The simplicity and low data rates of these interfaces, make them easy to manipulate and work with, even using inexpensive equipment. Yet, the integrity and confidentiality of the data exchanged between the dTPM and the host is crucial for protecting the FDE keys sealed by the dTPM—at least with current FDE solutions. This discrepancy has enabled several attacks, that led to the successful extraction of keys or forging of measurements.

Passive Attacks

In 2005, Kursawe *et al.* demonstrated the insecurity of the dTPM's communications channel by performing passive sniffing attacks [79]. They did their analysis on a dTPM that uses the LPC bus via a socket-connector, as this offers an easy way of probing the data lines. An FPGA was used to parse and filter the data monitored on the LPC bus and only forward TPM-related packets to a logic analyzer. All the data they captured was in plain text, thereby exposing secret keys during the unsealing operation.

One and a half decades later, this is still an exploitable attack vector, as documented by Andzakovic [80]. In this modern variation of the attack, a cheap FPGA board was used to extract the secrets from an LPC-connected dTPM on a Microsoft Surface 3. To filter out the interesting LPC

traffic, the FPGA board was programmed to only forward TPM-specific packets to the attacker's system [81]. Like in most portable devices, the dTPM targeted by this attack is soldered to the mainboard of the device instead of being connected via a socket. This complicated his attack, as wires had to be soldered on the small exposed pins of the dTPM to connect the data lines to the FPGA board. The machine was using BitLocker in TPM-only mode, meaning that the VMK is unsealed without user interaction during the boot process, as long as the measured boot components are genuine. By parsing the captured data, the VMK can be retrieved and used to decrypt the disk.

Pucher and Grebeniuk demonstrated improvements to the practicality of this attack and were able to perform it during a 30-minute presentation [82]. Instead of soldering, they used small probe hooks that can be attached to the exposed pins of the dTPM, making the process less intrusive and time-consuming. They also used a different, more practical approach for decrypting the data. Based on research by Kornblum [83], they implemented a tool for reconstructing the recovery password using the VMK and data from the BitLocker volume header. This way, no additional tools are required for decrypting the disk, as the recovery password can be used directly in windows to unlock the drive.

Active Attacks

One core security aspect of PCR measurements is that the registers' values cannot be reset while the system is running. The only time at which all PCR values are expected to be blank, is during the boot process before the first measurement by the CRTM has taken place. If it were possible to reset the TPM independently of the system, arbitrary measurements could be relayed to forge a desired state of PCRs and thus, among other things, unseal secret keys. As it turns out, exactly this can be achieved with a so-called *TPM reset attack*. This attack against dTPMs was described independently by Kauer [84] and Sparks [85] in 2007. It simply requires connecting the reset line of the LPC bus to ground using a piece of wire, which causes all connected devices to reset. Now, the dTPM is in the same state as when the system boots up, and all PCR registers are blank. The authors only described the steps following the dTPM reset in theory, but they are similar to how BitLeaker works (see section 3.5.3):

1. Obtain the PCR measurements of a genuine boot process from the firmware event logs.
2. Reset the dTPM by connecting the LPC reset pin to ground.
3. Initialize the dTPM with the same commands that are used by the BIOS/UEFI during

startup.

4. Replay the previously obtained PCR measurements to re-create the desired state.
5. Issue the necessary TPM commands to unseal the disk encryption key.

Winter and Dietrich introduced a variant of this attack, which they call *platform reset attack* [86]. Instead of resetting the dTPM independently of the host system, they reset the host system and keep the dTPM running. To accomplish this, the dTPM has to be shielded from the LPC reset signal that is triggered during a system reset. Additionally, the LPC framing signal has to be kept at the zero state during startup, effectively hiding the dTPM from the system and preventing any further interaction. This variant of the attack avoids having to retrieve and relay PCR measurements, and would instead involve the following steps:

1. Boot up the system normally, so that the PCR measurements allow the disk encryption key to be unsealed.
2. Shield the dTPM from the LPC reset signal and manipulate the framing signal.
3. Trigger a platform reset. The dTPM will be unaffected and the PCR values will remain unchanged.
4. Boot an attacker-controlled OS.
5. Restore the LPC framing signal and issue the necessary dTPM commands to unseal the disk encryption key.

In their research, Winter and Dietrich detailed the simple circuitry needed to perform this attack. Two lanes of the LPC connection to the dTPM have to be intercepted and run through this circuit which includes a switch to control the framing signal.

Besides their variation of the reset attack, Winter and Dietrich also introduced several active attacks targeting the Dynamic RTM (DRTM) capabilities provided by the TPM. Based on their insights, Boone developed *TPM Genie*, a tool that can intercept and modify the TPM's communications [87]. It is capable of spoofing PCR measurements, thus undermining the security of sealed storage and attestation. Boone also used it for attacking the host platform by sending malformed packets that cause unexpected behavior in response parsers. Through his research, 30 memory corruption vulnerabilities in open-source TPM drivers were revealed and reported to the software vendors.

3.5.2 Attacks Against Firmware TPMs

While fTPMs do not suffer from the data bus attacks discussed so far, they are susceptible to different types of vulnerabilities that affect the TEE they are running in. These can be classified into software vulnerabilities, targeting the fTPM implementation or underlying software that runs the TEE, or hardware vulnerabilities. In 1018, Cohen disclosed a stack buffer overflow vulnerability in the code responsible for parsing user-supplied Endorsement Key (EK) certificates [88]. By applying common exploitation techniques, these kinds of vulnerabilities could be leveraged to run arbitrary code in the TEE context, breaking the security guarantee of the fTPM.

In 2023, Jacob *et al.* explored the hardware-based attack surface, publishing their research on fault injection attacks against an fTPM from AMD [89]. Their threat model aligns very well with the one described in section 1.1, in that they target an FDE-protected laptop to gain access to the encrypted data. This is based on prior work by Buhren *et al.*, which demonstrated that carefully timed voltage fault injection can be leveraged to gain arbitrary code execution in AMD's TEE [90]. Jacob *et al.* took advantage of this vulnerability to extract the fTPM's secret key that is used to encrypt the non-volatile TPM data. This encrypted data could then be easily read and decrypted, as it is stored on the BIOS flash chip. With the complete internal state of the TPM exposed, they were able to unseal the cryptographic keys used for FDE. When this attack was applied to a BitLocker encrypted device with a TPM + PIN key protector, compromising the fTPM was not enough to decrypt the data—the PIN had to be brute forced first. BitLocker uses the PIN not only to authenticate against the TPM, but also to encrypt the VMK before it is sealed. Not every FDE solution implements the TPM + PIN combination as securely as BitLocker does. Jacob *et al.* found that *systemd-cryptenroll* uses the PIN only for authenticating against the TPM, therefore offering no protection if the TPM is compromised.[4]

3.5.3 Power Management Vulnerabilities

As already discussed, it is crucial that the TPM maintains the PCR values while the system is running. Otherwise, the security of PCR measurements cannot be guaranteed. However, PCRs are part of the TPM's volatile memory and therefore lose their state when the system enters sleep mode. For this reason, the TPM is supposed to save these values in its non-volatile RAM (NVRAM) before entering sleep mode, where they can remain until system wake-up. The TCG's TPM specification mandates how this process has to be implemented [13]. When the operating system

[4] `https://github.com/systemd/systemd/pull/27502`

wants to trigger sleep mode, it has to notify the TPM to save its state. The CRTM is responsible for deciding whether the TPM should restore its state or be re-initialized when transitioning back to S0.

In 2018, Han *et al.* published a vulnerability that arose from the improper handling of power-saving states in TPM version 2.0 [91]. It allowed PCR values of a TPM to be forged after transitioning from S3 to S0. While reviewing a change between the TPM 1.2 and 2.0 specifications, they found a flaw or ambiguity in the newer version that led to vulnerable implementations across many hardware vendors. The flaw is triggered by not notifying the TPM before entering S3 sleep, preventing it from storing its state. In version 1.2 of the specification, the TPM enters failure mode if no saved state is available when exiting sleep mode. In failure mode, the TPM no longer accepts new PCR measurements to protect itself from replay attacks. This fail-safe is not properly specified in version 2.0, allowing arbitrary measurements to be extended to the now empty PCRs. An attacker can therefore replay measurements from a previous, genuine boot sequence to access a sealed secret.

Han and Park later weaponized this vulnerability in a tool called *BitLeaker* [92]. It targets BitLocker encrypted systems in TPM-only mode and extracts the VMK to fully decrypt the disk. The tool comes in the form of a bootable Linux image that includes a custom bootloader, a special kernel module, and a customized TPM2 software stack. The ACPI state of the target computer is irrelevant for the success of this attack, and it has been demonstrated against dTPMs and fTPMs. The authors notified the affected hardware vendors, and some of them released firmware patches to mitigate this vulnerability. Nevertheless, this demonstrates how the complex interactions between the TPM and the host system can lead to unforeseen issues.

3.5.4 Invasive Silicon Attacks Against Discrete TPMs

In 2010, Tarnovsky demonstrated that, despite being fairly difficult, invasive silicon attacks can be successful at extracting keys from a tamper-resistant Infineon SLE66 chip [93]. Using a focused ion beam (FIB) machine—a device resembling a scanning electron microscope—and months of work, Tarnovsky was able to reverse engineer the design of the chip's inner circuitry. Microscopic conductive needles were then used as probes to intercept the internal data bus of the chip and extract secret keys. Two years later, Tarnovsky demonstrated a similar attack on STMicroelectronics' ST19WP18 dTPM chip [94]. Tarnovsky claims that, after the initial reverse engineering period of several months, an invasive silicon attack can be performed within a few hours and

with a very high success rate.

These attacks act as evidence that, with sufficient time and resources, even the tamper-resistant chips used as dTPMs can be defeated. Despite the required cost in terms of equipment and skilled professionals being significant, it is well within reach for specialized commercial laboratories or government agencies. Whether this is a realistic threat depends on the value of the data in question. It is certainly one of the more far-fetched attacks considered in this thesis, and in most cases, there will be easier attack vectors.

3.5.5 Side Channel Attacks

In 2020, Moghimi *et al.* disclosed a timing-based side channel vulnerability in a specific model of STMicroelectronics dTPMs and Intel fTPMs [95]. By analyzing the timing of thousands of signature operations, they were able to reconstruct the private signing key, which should never be accessible outside the TPM. This was possible due to secret-dependent execution times during these signing operations. They demonstrated that the timings are distinct enough to be exploitable not only with local access to the TPM, but over a low-latency network as well. It is possible, that similar side channel vulnerabilities could be found in the future, that affect the keys used for sealing operations. For now, both Intel and STMicroelectronics released firmware updates to mitigate these vulnerabilities.

Timing analysis not the only vector for side channel attacks against the TPM. *Differential Power Analysis* was famously introduced by Kocher *et al.* in 1999 [96]. It is the practice of inferring secret key material by measuring the power consumption of a device performing cryptographic operations. A leaked document from the Central Intelligence Agency (CIA), dated 2010, describes its alleged capability to extract secret keys from a dTPM using this technique [97], [98]. While none of these dTPM side channel attacks were ever demonstrated publicly, they are likely within the realm of possibility.

3.5.6 Authorization Sessions and Parameter Encryption

The TCG is well aware of the passive data bus attacks described in section 3.5.1 and the TPM 2.0 specification includes *Authorization Sessions*, a countermeasure that protects the integrity and confidentiality of TPM communications [13]. With authorization sessions, Hash-based Message Authentication Codes (HMACs) and nonces are used to authenticate each command and response to prevent replay attacks. Additionally, parameter encryption can be used to protect

sensitive data, such as secret keys during unsealing operations, from passive eavesdropping. To initialize an authorization session, the host generates a session key and encrypts it with a public key, for which only the TPM has access to the corresponding private key. This encrypted session key along with other session parameters are sent to the TPM when initializing an authorization session. A passive listener on the data bus between the TPM and host will not be able to decrypt the session key and command/response parameters of TPM commands sent over that session.

This defends against passive data bus attacks effectively, but is insufficient when dealing with active attacks that can be conducted with Man-in-the-Middle (MitM) devices such as Boone's *TPM Genie* [87]. Such a device can intercept and actively interfere with TPM-to-host communications, which opens the door for new styles of attacks. For example, when the host retrieves the TPM's public key that will be used for encrypting the session key, the MitM could instead provide its own public key to the host. Now, the MitM can decrypt and alter any data that is transmitted in the authorization session. To mitigate these kinds of attacks, the host needs to verify that it is talking to the actual TPM while initializing an authorization session and not a MitM.

3.5.7 Verifying TPM Authenticity

Of course, these kinds of MitM threats are not new in computer science and protocols such as Transport Layer Security (TLS) have solved this with Public Key Infrastructure (PKI). The *TCG EK Credential Profile* specification provides a similar approach to verify the authenticity of a TPM with PKI [99]. But this is a complex procedure to implement in an early boot environment, which is probably why there appears to be no FDE solution that has implemented it yet. The popular tool for enrolling TPM-based FDE on Linux, systemd-cryptenroll, opted for a Trust On First Use (TOFU)-based approach for verifying the TPM's authenticity[5].

3.5.8 The Case Against TPM-only FDE

Even if an FDE solution were to implement the authorization sessions with parameter encryption and prior verification of the TPM's authenticity, there is still a strong case to be made against TPM-only FDE. If the TPM unseals the FDE key based on PCR measurements alone, an attacker could just gather the PCR measurements during a genuine boot sequence and replay them using a MitM device resembling TPMGenie. That way, the host's precautions regarding TPM verification and the use of authorization sessions are circumvented, and the attacker can unseal the FDE key.

[5] `https://github.com/systemd/systemd/issues/22637`

Guarding the PCR measurements with authorization sessions would require all components of the boot chain (down to the SRTM) to support the use of authorization sessions and verify the authenticity of the TPM.

Another objection that highlights the need for user-supplied secrets (e.g., TPM + PIN combination) is the implication that TPM-only FDE has on cold boot, DMA, and SED attacks. In section 3.2 to section 3.4 these attacks were only considered to be a threat to FDE if the adversary gains physical access while the machine is powered on or in sleep mode. When the FDE key is solely protected by the TPM's sealing mechanism, as is the case in BitLocker's TPM-only mode, this assumption changes. Now, the encryption key is automatically unsealed during the boot process and resides in system memory, allowing the OS to fully boot up without a password. This not only makes these attacks effective against powered off computers but also increases the attacker's chance of success.

In a normal scenario, an attacker only has one attempt at launching a cold boot attack. If they fail (e.g., because of too many bit errors due to insufficient cooling), the RAM is now devoid of any data and no further memory-based attacks are possible. Similarly, a failed DMA or SED attack could cause the OS to crash in certain cases, ruling out any additional attempts. With a TPM-only setup, the attacker can simply boot up the machine and try again indefinitely. Additionally, the attacker can now attempt to disable any IOMMU-based protection or memory encryption technology through the BIOS.

3.5.9 Summary

The TPM has become a core technology in the realm of Full Disk Encryption, as it can provide security and usability benefits. But considering the attacks discussed in this section, over-reliance on the TPM for FDE needs to be avoided and a user supplied secret is still necessary to provide strong security guarantees. The TPM should be seen as a second factor that can compensate for a lower entropy FDE password. For this reason, Microsoft recommends using the TPM + PIN key protector for high security BitLocker deployments [100]. By using the PIN as authentication data in the key unsealing operation, the TPM can enforce its anti-hammering protection, which increases the difficulty of brute-force attacks considerably.

It is important that the PIN is used not only to authenticate against the TPM, but also to encrypt the DEK before it is sealed using the TPM. This way, even a fully compromised TPM only *weakens* the protection, but it does not completely *diminish* it, as the KEK is still protected by the PIN.

BitLocker already implements the TPM + PIN key protector in this secure manner [89, p. 11] and systemd-cryptenroll will likely follow a similar approach in the future.[6] Using a TPM + key file combination can eliminate the risk that a compromised TPM presents, but is currently only available for BitLocker.

The differing security properties between firmware TPMs and discrete TPMs are another significant consideration. While the general wisdom propagated by the TCG is that dTPMs are more secure than fTPMs [14], the attacks discussed in this section might create the opposite impression. Data bus attacks are only applicable to dTPMs, present relatively easy attack vectors and are hard to mitigate, because they abuse design flaws rather than fixable vulnerabilities. The attacks against fTPMs, on the other hand, are generally more involved and some of them rely on security vulnerabilities that can be mitigated with software patches. However, when the TPM is used in combination with a PIN, the scales clearly tip in favor of dTPMs. Now, the only currently known FDE-related threat for dTPMs are the complex and expensive invasive silicon attacks discussed in section 3.5.4. When using the TPM + PIN combination, even the lack of support for authorization sessions and parameter encryption in FDE solutions becomes much less of an issue.

In summary, FDE schemes that rely solely on the TPM should be avoided altogether. However, using a combination of TPM + PIN is clearly a net positive, *if* it is implemented securely by the FDE solution. In that case, dTPMs provide better security than an fTPMs, and either type is better than protecting FDE solely with a potentially weak password.

3.6 Evil Maid Attacks

If there is one common denominator among all the attacks discussed so far, it is that they can be executed in a single session and without any additional interaction from the user. They rely on the premise, that all the secrets required for decrypting the data reside somewhere on the device, such as in memory or the TPM. This means that our devices are also susceptible to these *drive-by attacks* when stolen or seized by law enforcement. By the time we realize our device is missing, there is often very little we can do to remediate the damage. Fortunately, robust and practical countermeasures exist for almost all of these attacks. Ensuring our devices are powered off or in hibernate mode when left unattended, combined with a solid FDE scheme and a strong password, are a very effective mitigation strategies.

[6] `https://github.com/systemd/systemd/pull/27502`

Still, a different class of attacks exists that can pose a threat even if we adhere to the strategies discussed so far. These attacks are conducted in two or more stages and aim to extract the secret key or password from the user. In the first stage, the attacker covertly tampers with the device, either through software or hardware, so that the encryption key or passphrase is captured when the user unlocks their device in the future. Once the user enters the necessary secret to unlock the device, it is either transmitted to the attacker remotely or retrieved when the attacker gains physical access for a second time. This initial physical access phase does not always need to be covert. For instance, there are situations where users are compelled to hand over their devices for inspection, such as during border security checks at an airport. In circumstances where keeping an eye on our devices at all times is not an option, detecting and preventing tampering becomes crucial.

Rutkowska coined the term *evil maid attack* in 2009, describing a scenario where the victim leaves their encrypted computer in a hotel room and the attacker, an evil maid, tampers with it in an undetectable way to defeat disk encryption [3], [101]. In the same year, Türpe *et al.* described a few variants of this attack that target BitLocker [4]. The term *evil maid attack* has since been used to describe a wide variety of attacks involving physical access, including cold boot and DMA attacks [102]. In this thesis, however, *evil maid attack* refers specifically to its original definition: a multi-stage attack targeting FDE, which requires user interaction to succeed.

3.6.1 The Original Evil Maid Attack

The original attack by Rutkowska and Tereshkin targets the Master Boot Record (MBR) of a TrueCrypt encrypted system partition and installs a sniffer that captures and stores the encryption password [3], [101]. Now, when the user enters their TrueCrypt password, it is saved on the disk. The attacker then needs to get physical access for a second time to retrieve the password and decrypt the data on the device. Over the years, several implementations of the evil maid attack have been developed for various FDE solutions, such as PGP Whole Disk Encryption [103] and Linux Unified Key Setup (LUKS)[7]. All of these simple implementations save the sniffed password somewhere on the disk. A conceivable improvement of this attack would be the implementation of remote password transmission, to allow conducting the attack with just a single physical encounter. The attacker could image the encrypted disk during the first visit, and the installed sniffer would supply the necessary secrets remotely. Götzfried and Müller implemented a similar

[7] `https://github.com/nyxxxie/de-LUKS` and `https://github.com/AonCyberLabs/EvilAbigail`

variant, which they refer to as *networked evil maid attack*, targeting Android's FDE feature [104].

While the TrueCrypt and LUKS variants of the evil maid attack were trivial to implement, it is more of a challenge for BitLocker, because of its use of the TPM. Ever since BitLocker's release in Windows Vista, the TPM's key sealing functionality has been used to secure the disk decryption key. It guarantees that the key can only be unsealed by the TPM if the measurement of certain boot components matches a defined state. The modifications necessary for logging the user's password would impact the PCR measurements, preventing the key from being unsealed and causing errors during the boot process. Nevertheless, Türpe *et al.* described five advanced attack scenarios in 2009 that specifically target the BitLocker boot process and circumvent this protection [4]. Two of these attacks (*tamper-and-revert*, *replace-and-relay*) are especially relevant and will be discussed in the following sections.

3.6.2 Anti Evil Maid

In an attempt to mitigate the original evil maid attack, Rutkowska published *Anti Evil Maid*, a mutual authentication scheme, in 2011 [105]. It uses the TPM's sealing functionality to store a key that is used to decrypt and display a user-defined secret message. This key will only be unsealed if the measured boot process results in the correct PCR state. The user is instructed to only enter their password if the correct authentication message appears, and assume that the system has been compromised if it does not. If the message were to be unsealed automatically, the attacker could boot the unmodified system, note down the message and replicate it to perform any type of evil maid attack. To prevent this from happening, anti evil maid offers two possible ways of protecting the message. The first option is storing the encrypted message on an external USB stick that has to remain in the user's possession at all times. Alternatively, a Storage Root Key (SRK) password for the TPM can be set up, which has to be provided each time the system boots to unseal the message encryption key. Both of these variants heavily rely on the confidentiality of the user's secret message and would require special caution when booting the device in public places.

3.6.3 Tamper-and-Revert Attack

This variant of the evil maid attack works despite the utilization of a TPM's key sealing feature. Depending on the implementation of the TPM + PIN combination, there is no way around having to gain physical access for a second time after sniffing the password. In the case of BitLocker, the

PIN is used to authenticate to the TPM and, together with the valid PCR measurements, allow the key to be unsealed. During the first visit, the attacker has no knowledge of the PIN and therefore cannot retrieve the key from the TPM (e.g., by performing a TPM sniffing attack). Therefore, physical access is required twice and there is no benefit to implementing remote password exfiltration. This variant of the attack was described independently by Rutkowska [101] and Türpe *et al.* [4]. It involves the following steps:

1. A forged password prompt or logger is installed by the attacker. It operates similar to the original evil maid code, but includes a cleanup function to uninstall itself.
2. The unsuspecting user enters their password, and it gets logged by the malicious software. Due to the modification of boot components, the TPM measurement will not be successful, and the key cannot be unsealed.
3. The malicious code uninstalls itself, optionally presents a plausible error to the user, and reboots the system.
4. Now, the boot components are back to their original state and the TPM will unseal the key. The user might be surprised by the reboot, but will most likely enter their password again and continue working on the computer.
5. The attacker gains physical access for a second time to retrieve the password and access the encrypted data.

Similar to Rutkowska, Türpe *et al.* argue that, to defend against this attack, the user needs a way of confirming the authenticity of their device, before entering any passwords. But as Müller *et al.* points out, the simple anti evil maid solution described in section 3.6.2 is not sufficient [106, p. 4]. An attacker could perform a three-stage attack to defeat the original anti evil maid implementation.

3.6.4 Improved Anti Evil Maid

Garrett presented a variant of anti evil maid that withstands tamper-and-revert attacks by replacing the static secret message with Time-based One-Time Passwords (TOTPs) [107]. Using the same mechanism that is commonly used for multi-factor authentication, a secret key is shared between two entities during setup and then hashed in combination with the current time to produce a six-digit code. On one side, the secret is sealed by the TPM, where it can only be unsealed if the measured boot process leads to the correct PCR state. The other copy of the secret key can be stored in any standard multi-factor app on the user's smartphone. When the system is started

in an unaltered state, the TPM unseals the secret and a six-digit code is calculated and displayed. The user has to proactively compare the code to the one currently shown in the app and only proceed with entering their FDE password if the codes match. This removes the requirement for keeping the authentication message secret, but now exposes the secret key for TOTP generation to TPM sniffing attacks. A newer implementation for TPM version 2.0 solves this issue by performing the calculation of the TOTP entirely within the TPM [108]. As noted by the authors, this method still has one important limitation: The time source used for calculating the TOTP is not particularly tamper-proof. If an attacker manages to spoof the system's clock, they can calculate future codes in advance and use them to launch a replace-and-relay attack.

Mutual authentication schemes that rely on a specialized trusted USB device, have also been proposed in the past. McCune *et al.* presented some of the research challenges involved in designing such a device, which they called *iTurtle* [109]. In 2014, Götzfried and Müller introduced *MARK*, a concrete proposal for implementing such a device, which signals the authenticity of the computer via colored LEDs [110]. MARK uses an elaborate protocol based on prior work by Müller *et al.* to mutually authenticate the device and user [106]. In 2018, the *Nitrokey Pro 2*[8] was introduced, which offers a feature similar to MARK when used together with the coreboot firmware.

3.6.5 Secure Boot

When the anti evil maid solutions were introduced, UEFI Secure Boot was not widely available yet. Nowadays, it is ubiquitous and provides an even simpler and, in some cases, better protection mechanism against tamper-and-revert attacks. Secure Boot can help to mitigate evil maid attacks by acting as an active security component that prevents booting tampered software. When Secure Boot is enrolled with custom keys, the user can control which EFI binaries the system is allowed to start. The signature verification process ensures the integrity and authenticity of the first EFI binary in the boot process—typically the bootloader. The first component is then responsible for checking all subsequently started components and deny loading them if they have been tampered with. This is the general concept behind the *Trusted Boot* process in Windows 10 [111]. When Secure Boot is used with custom keys and alongside a hardened UEFI setup (see section 4.2), it can protect against the evil maid and tamper-and-revert attack variants discussed so far.

The significance of deploying custom Secure Boot keys for enhanced security was underscored

[8] `https://www.nitrokey.com/files/doc/Nitrokey_Pro_factsheet.pdf`

by the *BootHole* vulnerability identified by Shkatov and Michael in 2020 [112]. This vulnerability was found in various versions of the GRUB2 bootloader that are (indirectly[9]) signed by Microsoft's third-party UEFI CA. Since the default set of Secure Boot keys on most computers trust Microsoft's third party CA, these GRUB2 vulnerabilities affect a majority of Secure Boot enabled devices, regardless of whether they use GRUB2 or not. The flaw enabled attackers to run arbitrary code within the bootloader, allowing them to load any unsigned EFI binary. This is only one example of many hundreds of EFI binaries once signed by Microsoft that are now known to be vulnerable. If not added to the *dbx*, they can be exploited by attackers to bypass Secure Boot.

Aside from vulnerable EFI executables, Secure Boot is susceptible to attacks targeting the UEFI firmware itself. In 2014, Kallenberg *et al.* demonstrated how improperly protected UEFI variables or SPI flash regions could be exploited to bypass Secure Boot [113]. And even if Secure Boot were impenetrable, Türpe *et al.* described an attack that works despite these platform hardening efforts: the *replace-and-relay* attack [4].

3.6.6 Replace-and-Relay Attack

To overcome Secure Boot and the requirement for a second visit, a *replace-and-relay* attack can be performed [4, p. 8]. In this attack, the device is stolen and replaced by an identical-looking *evil twin* that displays the same password prompt when powered on. The replacement device relays the user's password over the network, giving the attacker the necessary secrets to unlock the original machine and access the encrypted data. Instead of replacing the whole device, it might be more effective to replace just the main components inside, maintaining its visual appearance and condition. Additionally, emulating the exact software prompts and messages of the original system will result in a practically indistinguishable clone. After the user enters their password, they might notice that the device behaves unfamiliarly, but at that point, it is already too late. This type of attack has similarities to online phishing attacks, which is why Türpe *et al.* call it a *hardware-level phishing attack*.

Depending on the effort an attacker is willing to invest, this kind of attack can be very difficult to prevent. However, making the attacker's life as hard as possible is still a worthwhile security goal. The TOTP-based anti evil maid solutions described earlier can offer some form of protection, as cloning a dynamic six-digit code is more challenging than merely recreating static password

[9] While Microsoft does not sign Linux bootloaders directly, it signs a program known as *shim* that, in turn, uses the OS vendor's key to verify and execute approved Linux bootloaders.

prompts. Using TPM-based FDE also helps here, because it prevents the attacker from decrypting the data outside the original device. That way, even if the user's password is successfully phished, the attacker still needs to bypass OS security through other means (e.g., DMA or cold boot attack). Unless, of course, the attacker can make the evil twin convincing enough to also phish the user's lock screen password.

3.6.7 Advanced Replace-and-Relay Attack

Türpe *et al.* describe an extension [4, p. 9] of the replace-and-relay attack that remains effective despite all software-based mitigations mentioned thus far. Instead of merely emulating the prompts and messages of the original system, the evil twin could act as a terminal for it. By capturing and relaying all screen contents, user inputs, and USB communications, most attempts at authenticating the system to the user will be ineffective. Any interface (e.g., screen, keyboard, NFC, USB) that a mutual authentication scheme could use to authenticate the device can be tunneled through the evil twin to the original device that is in the attacker's possession. Detecting an evil twin device is no easy task, but because relaying inevitably introduces latency, it might be feasible with timing-based attestation. If an attestation device like iTurtle [109] or MARK [110] were to measure the response time while interacting with the TPM, it could certainly detect the added latency of interacting with an evil twin. The concept of timing-based attestation is certainly not new in computer science, and it has been proposed to complement TPM-based attestation for at least a decade [114].

3.6.8 Keystroke Logging

When delving into the complex nature of certain physical access attacks, it's easy to overlook simpler attacks such as hardware keyloggers. While significant strides have been made in defending against software-based evil maid attacks, our detection and prevention ability of hardware-based attacks remains limited. Hardware keyloggers, which can be seen as the physical counterpart of evil maid attacks, have been readily available for many years. They are usually inserted between the USB keyboard and the computer, and feature wireless connectivity for remote exfiltration of the logged data.[10] With increasingly small sizes, they have become difficult to detect through mere visual inspection, and they can even be soldered into the keyboard itself, making them virtually undetectable. Some keyloggers are disguised as regular USB cables that are indistinguishable

[10] `http://www.airdrivewifi.com`

from the original[11].

At first glance, hardware keylogging seems to be a threat only to users of external keyboards. After all, that is why the original evil maid attack, a form of software keylogging, had to be developed instead. But laptops are not immune to hardware keylogging and Mini-PCI-based keyloggers[12] have been available in the past. Although there does not seem to be a newer version of this product on the market, it would certainly be possible to develop one that targets the flat ribbon cable which connects most laptop keyboards to the mainboard. The NSA likely has such a device, as suggested by the product brief of SURLYSPAWN, their keystroke monitor with wireless transmission capability, from 2009 [115]. The ability to replace keyboards in some laptops without opening the case would complicate the detection of such attacks, even with tamper-evident technology.

Over the years, several side channel keylogging attacks have been published that target desktops and laptops indiscriminately [116]. Most notably, acoustic emanations have been demonstrated to contain enough information to infer typed words by analyzing the acoustic signature of computer keyboards [117]. With advancing analysis techniques, these attacks have progressed since their debut in 2004 to target random passwords [118], [119] and even do so over voice chat [120], [121].

There are some early approaches for detecting USB keyloggers, but they are neither universally applicable nor practical [122], [123]. Due to the manifold ways in which keystroke logging can be implemented, it is nearly impossible to fully eliminate the threat. Instead, the goal is to eliminate the keyboard as the sole mechanism for entering secret keys for FDE. The use of a key file, either in addition to or instead of a PIN, can enhance security and reduce the risk of keylogging. FDE solutions that use the TPM + PIN combination can also help by making hardware keylogging attacks more difficult. As discussed in section 3.6.3, the attacker has no knowledge of the PIN during the first visit and therefore cannot retrieve the key from the TPM. Therefore, physical access is required twice, and there is no benefit to implementing remote password exfiltration.

3.6.9 Other Hardware Implants

Although keyloggers are the most prevalent, they are not the only type of hardware implant to be concerned about. In the context of this thesis, there are other ways in which specialized hardware

[11] `https://shop.hak5.org/products/omg-cable`

[12] `https://web.archive.org/web/20161207054545/http://www.keycarbon.com/products/keycarbon_laptop/overview`

can be used by attackers to circumvent FDE in an evil maid scenario.

In 2013, Appelbaum *et al.* published leaked documents from the NSA's Tailored Access Operations (TAO) unit [124], [125]. These leaks included a catalog of product briefs for hardware implants made by the TAO unit and meant for sale within the intelligence agency [115]. Among the 50 implants listed in this catalog were the previously mentioned SURLYSPAWN keylogger, COTTONMOUTH, which is a series of covert USB-based implants for remote access, and RAGEMASTER, an implant for wirelesses relaying of VGA signals. Given that these product briefs originate from 2008 and 2009, it's likely that intelligence agencies now have much more advanced implants at their disposal.

The leak of the ANT catalog sparked a lot of interest in the hardware hacking community and led to the creation of the *NSA Playset*, a public community effort to recreate some of the NSA's implants [126]. Besides keyloggers that specifically target wireless keyboards, this project has introduced a few other implants that are relevant for our threat model. Among them is a PCIe implant for DMA attacks, a JTAG implant that attacks the OS, and various malicious adapters. *PicoDMA* is another DMA implant, which supersedes the one from the NSA Playset and features wireless connectivity [127]. DMA is a powerful attack vector for hardware implants because it changes our previous assumptions about a device's susceptibility to DMA attacks. In section 3.2, we examined the DMA threat almost exclusively for computers in ACPI states S0–S3. Covert DMA implants give the attacker the flexibility to wait until the device is in the desired state and execute the attack when the device is most vulnerable (e.g., during boot). Of course, hiding such an implant in the confined space of a laptop's chassis is challenging. However, if a docking station is utilized, an adversary might opt to embed the implant there, as demonstrated by Davis in 2013 [128].

These examples demonstrate that capable hardware implants are not exclusive to well-funded intelligence agencies; they can also be constructed using readily available off-the-shelf components. This also highlights that peripherals are often easier targets for implant-based attacks. Consequently, they should be safeguarded and examined with the same vigilance as the computers they are connected to.

3.6.10 Tamper-Evident Technology

The threat of hardware-based attacks discussed in this section highlights the need for *tamper-evident technology*. This refers to a class of tools that allows users to detect when their devices

may have been physically tampered with. Using that information, the user can make an informed decision to stop using the compromised device, particularly refraining from entering any confidential keys. TPM-based attestation solutions are a form of software-based tamper-evident technology and have been addressed in the previous sections. However, these can only protect against software-based threats and are ineffective when faced with hardware keyloggers or implants. This is where physical tamper-evident technology becomes important.

Tamper-Evident Seals

The simplest type of devices for this purpose are tamper-evident seals, a technology that has been in use for thousands of years [129]. These seals come in various shapes, materials, and price points, each with different security properties. Manufacturers typically imprint a unique serial number on the seal to prevent it from being replaced with an identical counterpart. For our specific use case, the most relevant types are metal/plastic loops and adhesive label seals [130]. They can be applied to a computer in such a way that, in theory, the case cannot be opened without visibly damaging the seal. As long as the user is diligent about inspecting the seal before using the device, this can help against evil maid attacks that target the inner components. However, a study of 244 different commercially available seals found that most could be easily defeated within 2 minutes using low-tech methods and all of them could be thwarted within 30 minutes [129]. Interestingly, there are annual competitions for defeating tamper-evident seals, hosted at major security conferences [131].

One promising approach for improving tamper-evident seals was proposed by Michaud and Lackey in 2013 and has since gained a lot of popularity [132]. This method involves applying nail polish infused with glitter or other small particles over the screw holes of a device, so that any tampering with the screws would visibly damage this unique seal. Given that the patterns formed by small glitter particles are inherently random, replicating a damaged seal becomes very challenging. A method known as *blink comparison*, which was formerly employed by astronomers, can be used to contrast a before-and-after image of the seal to spot any discrepancies. An Android app specifically designed to help with the alignment and comparison of these two images is also available [133]. However, a fundamental drawback of passive tamper-evident technology persists in these advanced seals: users must inspect the seal carefully every time.

Active Tamper Detection

An alternative method that requires less intervention on the user's part is the use of active sensors to detect hardware tampering. With these sensors, the user can be proactively alerted to potential physical interference. Some notebooks are equipped with small sensors that detect when the bottom cover has been removed, even if the device was powered off during removal [134], [135]. Upon subsequent startup, the user is alerted to the tampering attempt, and the BIOS password is required to continue booting. Nonetheless, an adept attacker might be able to bypass this single sensor. In the past, there was an initiative to produce a highly tamper-resistant personal computer, named *ORWL*.[13] This computer incorporated multiple intrusion sensors and an enveloping active mesh to detect physical breaches of the device's enclosure.

For computers that were not designed with that level of security in mind, external sensors can be used to supplement the tamper detection capabilities. One notable example is *MetaSensor*[14], a small battery-powered device equipped with an array of sensors that can be attached to a laptop. This device alerts the user via their smartphone of any detected movement, provided the phone is within Bluetooth range. To overcome the range limitation of Bluetooth, a spare Android phone with a cellular internet connection can be repurposed into a comprehensive tamper detector with the *Haven* app [136]. Haven uses the smartphone's accelerometer, camera, microphone, and light sensor to detect any changes in the environment and notify the user via a secure channel. For instance, if both the laptop and the Haven-enabled phone are placed inside a safe or drawer, it becomes extremely difficult to tamper with the laptop without triggering one of the sensors. Unfortunately, as of August 2023, Haven's alerting functionality has been broken for at least two years and the development appears to have stalled.

The necessity of a secondary device could be eliminated by implementing a Haven-like application, that runs on the computer itself. We will name this proposed solution Evil Maid Misconduct Avoidance (EMMA). EMMA could be part of a minimal Linux-based OS that can be booted whenever the device will be left unattended in an untrusted environment. Once activated and connected to the internet, EMMA monitors the laptop's sensors and sends periodic status updates with signed nonces to a trusted server. A USB-connected accelerometer could be used to complement the standard sensors commonly found in laptops. If sensors are triggered or unexpectedly disconnected, EMMA immediately sends an alert to the trusted server, which subsequently warns

[13] `https://www.crowdsupply.com/design-shift/orwl/`
[14] `https://metasensor.com`

the user. Similarly, any interruption in EMMA's routine status updates to the server prompts an immediate notification to the user, signaling a potential security breach. A solution like EMMA would provide comprehensive protection against all types of evil maid attacks without requiring a dedicated monitoring device.

Combining Solutions

Each of the tamper-evident technologies discussed so far offers a unique trade-off between *sensitivity* and *specificity*, terms which we will borrow from the field of epidemiology for this explanation. A test that offers high sensitivity has a low false *negative* rate, but usually a high false *positive* rate. In our realm, the Haven app or our EMMA proposal fit into this category. Armed with numerous sensors and an active heartbeat mechanism, these solutions are unlikely to miss a tampering attempt. However, their high sensitivity makes them prone to false alerts, such as when a *not-so-evil* maid moves the laptop to clean the desk.

At the other end of the spectrum are high-specificity tests. They have a low false *positive* rate, but have a higher chance of producing false *negative* results. Tamper-evident seals and case intrusion sensors fall into this category. When designed properly, they very rarely produce false alerts, but they are also more likely to be defeated by a prepared adversary. Much like in epidemiology, a combination of both categories provides the best results. While the high-sensitivity solution helps to detect every potential tampering attempt, the high-specificity solution helps to discern between real intrusions and false alarms.

Another aspect to consider is the applicability of each solution in different situations. A high-sensitivity solution like Haven is unsuitable for situations where the user *expects* someone to inspect their computer, such as during airport security checks. In such situations, a high-specificity solution like case intrusion sensors is ideal to determine if the potential adversary opened the device.

Active Tamper Prevention

The discussion around tamper-evident sensors has largely focused on notifying users of potential tampering attempts. However, there is potential to expand these concepts to initiate automatic preventative measures. For instance, HP's TamperLock technology already implements configurable proactive security measures when the bottom cover is removed [134]. Besides performing an immediate shutdown and demanding the BIOS password upon the next reboot, TamperLock is

capable of resetting the TPM, thus permanently destroying the FDE key. Similarly, the aforementioned ORWL computer featured key destruction features that activate in response to unauthorized physical entry. Such mechanisms improve the device's resilience against evil maid, as well as cold boot and DMA attacks. With these safeguards, a device is considerably less vulnerable when left in ACPI states S0 to S3.

Our EMMA proposal could be extended to introduce active tamper-prevention capabilities to any computer, by deleting encryption keys in addition to alerting the user. To protect the device in S0, EMMA could operate as a privileged daemon inside the primary OS, rather than running in its own minimal OS. When a user invokes the lock screen, it arms EMMA. From now on, EMMA monitors the specified sensors and takes action when any tampering is detected. Upon detection, EMMA can erase the encryption keys stored in the TPM and either shut down or hibernate the computer. Choosing hibernation over shutdown could be preferable, as it preserves the system's state, minimizing potential data loss during false alarms. However, introducing similar protections for the S3 sleep state, akin to what TamperLock offers, presents challenges since the CPU is inactive during S3.

3.6.11 Summary

Microsoft's third immutable law of security states that if an attacker has unrestricted physical access to your computer, it is no longer your computer [137]. However, the ongoing efforts by Microsoft and other entities to counteract physical access attacks suggest that the security community remains dedicated to mitigating the issue. While widely available technologies like TPM-based attestation and Secure Boot effectively counter the original evil maid attack and some of its successors, they fall short against hardware-based evil maid attacks such as replace-and-relay, physical keyloggers, and other implants. For these, we need to shift our focus towards hardware-based mitigation strategies.

Homemade tamper-evident seals can be an effective and affordable solution for detecting some types of hardware-based attacks, but their reliance on regular active inspection by the user makes them cumbersome. Many business-oriented notebooks are already equipped with case intrusion detection that alerts the user if the bottom cover is removed, and some are even capable of resetting the TPM in response. However, these solutions are more likely to be defeated by a prepared adversary due to their low sensitivity, which was designed to reduce false alarms.

For optimal security, these low-sensitivity technologies should be combined with others that

are harder to defeat. Aftermarket solutions like *Haven* and our proposed EMMA could fill this role by utilizing a whole range of sensors and instantly notifying the user via an active internet connection. The concept of EMMA could be further enhanced by incorporating preventative actions, like resetting the TPM and hibernating the device. This would make is considerably more challenging to tamper with the computer without alerting the user or activating these protective measures.

However, these protections are specific to the computer itself, leaving peripherals vulnerable to attacks. Extending the same safeguards to all peripherals quickly becomes impractical. Therefore, in situations that necessitate a stringent defense against evil maid attacks, one must inevitably reduce the number of peripherals, opting to use the laptop in its standalone configuration whenever feasible.

3.7 Summary and Evaluation

In this chapter, we examined a range of physical access attack vectors targeting unattended computers, with a particular focus on circumventing FDE. We discussed the available literature on practical attacks, as well as theoretical attacks that have not yet been demonstrated or implemented. Additionally, we examined both academic and readily available countermeasures and the mechanisms by which they attempt to mitigate the attacks. The intricate interactions between attacks and countermeasures can sometimes be obscure. To help clarify these interactions, table 3.1 provides an overview that contrasts the most important attacks and readily available countermeasures. The effectiveness of each countermeasure varies between attacks and is represented by distinct symbols in the table.

- **DMA port authorization** (section 3.2.2): Thunderbolt and USB4 can block/allow PCIe tunneling on a per-device basis to mitigate DMA attacks from external devices. However, this security feature has several inherent vulnerabilities and can be bypassed (section 3.2.4).
- **DMA remapping** (section 3.2.3): Using the IOMMU, operating systems can restrict a device's DMA capability to confined memory regions. However, this cannot protect against more advanced DMA attacks that exploit temporal and spacial vulnerabilities of IOMMU-based protections (section 3.2.4).
- **Memory encryption** (section 3.3.4): Transparent hardware-based memory encryption prevents cold boot attacks and requires no special OS support.
- **Software-based FDE** (section 2.3.1): Software-based FDE prevents various attacks that

exclusively target SEDs (section 3.4).

- **Power off or hibernate** (section 2.4): DMA, cold boot, and some SED attacks can be fully prevented by turning off or hibernating the device when left unattended.
- **TPM + PIN** (section 2.2): Prevents the original evil maid attack and makes certain other types of evil maid attacks more difficult to execute (section 3.6.3). It even makes hardware keylogging more difficult by requiring the attacker to make a second visit and diminishing the benefit of remote keystroke exfiltration (section 3.6.8). The PIN is required to prevent TPM sniffing, TPM reset, and platform reset attacks to which TPM-only FDE solutions are vulnerable. As an added benefit, the TPM's anti-hammering protection prevents PIN brute-force attacks on the PIN.
- **Strong TPM + PIN implementation** (section 3.5.2): Strong implementations of the TPM + PIN combination use the PIN not only to authenticate to the TPM, but also to encrypt the KEK before it is sealed. This protects against attacks that target the TPM itself to extract its secret keys (section 3.5.2 and section 3.5.4). With a strong TPM + PIN implementation, an attacker still needs to brute-force the PIN after compromising the TPM.
- **TPM + key file** (section 3.6.8): Using an externally stored key file instead of a PIN to encrypt the KEK before sealing it with the TPM, provides even stronger protection. Given that the attacker cannot brute-force a high-entropy key file, they gain very little by compromising the TPM. This even mitigates the risk of hardware keyloggers.
- **TPM authorization sessions** (section 3.5.6): TPM authorization sessions with parameter encryption prevent TPM sniffing attacks (section 3.5.1) regardless of whether a PIN is used or not.
- **Anti evil maid** (section 3.6.2): The original anti evil maid implementation only offers weak protection against evil maid (section 3.6) and tamper-and-revert (section 3.6.3) attacks. This mitigation relies on proactive inspection by the user.
- **TOTP-based anti evil maid** (section 3.6.4): The more recent TOTP-based anti evil maid solutions prevent the original evil maid and tamper-and-revert attacks. They also help to mitigate replace-and-relay attacks (section 3.6.6) by increasing the difficulty of cloning the password prompt. This mitigation relies on proactive inspection by the user.
- **Hardened Secure Boot** (section 3.6.5): Protects against platform and TPM reset attacks (section 3.5.1) by preventing the attacker from booting an unauthorized operating system. Secure boot also prevents some types of evil maid attacks without relying on active inter-

vention by the user.

- **Tamper-evident seals** (section 3.6.10): Help the user to detect physical tampering, and mitigate attacks that involve hardware implants (section 3.6.9). Seals that are unique and unclonable can additionally mitigate replace-and-relay attacks. Ineffective against hardware implants and keyloggers that target peripherals. This mitigation relies on proactive inspection by the user.
- **Case intrusion sensors** (section 3.6.10): Alert the user when physical tampering is detected to mitigate attacks that involve hardware implants. Do not require active inspection by the user. Ineffective against hardware implants and keyloggers that target peripherals.
- **External tamper sensors** (section 3.6.10): Solutions like Haven can utilize an array of sensors to detect potential threats and alert the user. They are hard to bypass and can prevent all evil maid attacks that aim to manipulate the computer itself. Ineffective against hardware implants and keyloggers that target peripherals.

In summary, there are readily available countermeasures for most of the attacks described in this chapter. However, certain attack vectors, particularly hardware implants targeting peripheral devices, remain difficult to fully eliminate. The responsibility for securing these peripherals falls to the user, a task that may not be feasible for desktop computers. The reliance on active user intervention is an important consideration when evaluating the effectiveness of a given countermeasure. Ideally, countermeasures would actively alert the user and autonomously prevent the attack, but such solutions are not always available. While some computers have built-in tamper prevention mechanisms, there is an unmet need for a solution like our proposed EMMA that provides active tamper prevention for existing computers. It also remains to be seen whether the countermeasures discussed in this chapter can deliver what they promise and how well they can be used in combination with each other. The next chapter will shed some light on these aspects by guiding through the process of setting up a hardened Linux-based system.

Table 3.1: Summary of physical access attacks and readily available countermeasures.

	Drive-by Attack									Evil Maid Attack					
Countermeasure	DMA attack (ext. port)	DMA attack (int. port)	Cold boot attack	SED attacks	TPM sniffing	TPM reset attack	Platform reset attack	fTPM fault injection	Invasive silicon attacks	Original evil maid	Tamper & revert	Replace & relay	Advanced R & R	Hardware keylogger	Hardware implants
DMA port authorization	◐	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DMA remapping	◐	◐	-	-	-	-	-	-	-	-	-	-	-	-	-
Memory encryption	-	-	●	-	-	-	-	-	-	-	-	-	-	-	-
Software-based FDE	-	-	-	●	-	-	-	-	-	-	-	-	-	-	-
Power off or hibernate	●	●	●	◐	-	-	-	-	-	-	-	-	-	-	-
TPM + PIN	-	-	-	-	●	●	●	-	-	●	◐	◐	-	◐	-
Strong TPM + PIN impl.	-	-	-	-	●	●	●	◐	◐	●	◐	◐	-	◐	-
TPM + key file	-	-	-	-	●	●	●	●	●	●	◐	◐	-	●	-
TPM auth. sessions	-	-	-	-	●	-	-	-	-	-	-	-	-	-	-
Anti evil maid	-	-	-	-	-	-	-	-	-	◐	◐	-	-	-	-
TOTP-based AEM	-	-	-	-	-	-	-	-	-	●	●	◐	-	-	-
Hardened Secure Boot	-	-	-	-	-	◐	●	-	-	●	●	-	-	-	-
Tamper-evident seals	-	-	-	-	-	-	-	-	-	-	-	◐	◐	◐	◐
Case intrusion sensors	-	-	-	-	-	-	-	-	-	-	-	-	-	◐	◐
External tamper sensors	-	-	-	-	-	-	-	-	-	●	●	●	●	◐	◐
Sum of countermeasures	●	●	●	●	●	●	●	●	●	●	●	●	●	●	◐

● = prevents; ◐ = partially mitigates; - = does not mitigate;

4 Hardening Guideline

In the preceding chapter, we have discussed physical access attacks and their associated countermeasures. Our assessment concluded that almost all such attacks can be prevented with existing solutions. However, this assessment did not consider the compatibility of these countermeasures with modern operating systems and with each other. This chapter seeks to fill those gaps. We will walk through the process of choosing and configuring the right hardware and software to build a hardened Linux-based setup. We will discuss the challenges along the way, and analyze the deviation between our theoretical best case (presented in table 3.1) and the result of following our practical hardening guideline.

4.1 Test Setup and Requirements

This section details the hardware prerequisites and the software that was used throughout this chapter. Listings that document shell commands are titled `user@hostname` to differentiate between the two hardware platforms that were used.

4.1.1 Hardware Platform Prerequisites

There are certain hardware requirements for implementing the hardening steps discussed in this chapter. The following things should be kept in mind when attempting to recreate these steps or when making future purchasing decisions.

UEFI and Secure Boot

Since UEFI is only a specification [11], the essential security features it provides, like Secure Boot, are only as good as the specific manufacturers' UEFI implementation. Due to the closed source nature of most UEFI implementations, it is hard to reliably evaluate whether a particular one is secure. And it is especially difficult to do so at scale, such as when trying to make a purchas-

ing decision. Nevertheless, it is a good idea to investigate a manufacturer's reputation regarding firmware security and updates.

Dedicated TPM

As outlined in section 3.5.9, dTPMs offer greater security than fTPMs for our specific use case. The type of TPM is usually declared in the specification sheet of a laptop or desktop computer. In the case of desktop computers, it is common to encounter motherboards that come without a pre-installed TPM, but provide a header for installing an aftermarket TPM module. Another important consideration is that some of the tools discussed in this chapter are only available for TPMs of version 2.0.

IOMMU

The mitigations in section 4.4 rely heavily on the IOMMU, which is a feature found in an increasing number of modern CPUs. Nevertheless, the specifications of a particular CPU should be checked before making a purchase decision. In the case of Intel, the IOMMU is part of their VT-d [37] feature set, whereas AMD offers it as part of AMD-Vi [38]. For Intel CPU, the ARK product database[1] is a good resource to query CPUs by a set of features.

Memory Encryption

Determining whether a CPU supports memory encryption can be a tedious task, as the feature is often hidden behind various marketing terms. The technical name for memory encryption in AMD CPUs is *SME* or *TSME*, but this term is rarely found in the CPU specification sheet. The term to look for instead is *AMD Memory Guard* or *AMD Infinity Guard*, which can be found in the Ryzen PRO and EPYC series of AMD CPUs, respectively [138], [139].

Intel introduced its memory encryption technology *TME* with the 3rd Gen Xeon Scalable and the 11th Gen Core vPro series of processors [74], [140]. However, only some CPUs of these series list *Intel Total Memory Encryption* in the specification list. Therefore, Intel's ARK database should always be checked first.

[1] `https://ark.intel.com`

Physical Tamper Detection/Protection

An ideal computer for our scenario would feature robust physical tamper protection, reminiscent of the ORWL computer discussed in section 3.6.10. Unfortunately, there are very few computers with this level of physical security. That said, some business-oriented laptops do include basic anti-tamper technologies in the form of case intrusion sensors. Most Lenovo ThinkPads come with such a feature [135], and some HP business laptops feature a similar technology called *TamperLock* [134]. TamperLock stands out for its superior protection capabilities, such as clearing the TPM and initiating an immediate shutdown when an intrusion is detected. Lenovo's case tamper switches, on the other hand, can only warn the user and prompt for the BIOS password when tampering is detected.

4.1.2 Test Platforms

While the mitigations in this chapter should theoretically work on any computer that fulfills the prerequisites listed in section 4.1.1, they were only tested with the hardware and software described in this section.

Hardware

Most of the steps described in this chapter were first tested in a QEMU virtual machine (hostanme `qemu`), before being replicated on the main hardware platform (hostname `t495s`). The OS, kernel, and software package versions were kept in sync between the two platforms to ensure consistent results. The following laptop served as the main hardware platform for this chapter:

- *Model:* Lenovo ThinkPad T495s (20QJ0012GE)
- *CPU:* AMD Ryzen 7 PRO 3700U
- *RAM:* 16 GB DDR4 (soldered)
- *dTPM:* Nuvoton Technology NPCT75x (firmware version 7.2.1.0)
- *System firmware:* Version 1.28 (R13ET54W)

For the reasons outlined in section 3.5.7, it is important that our dTPM comes with an EK certificate that is signed by the TPM manufacturer. We can retrieve the TPM's EK from NVRAM using the `tpm2_getekcertificate` command from the tpm2-tools suite[2], as seen in listing 1. The certificate contains information regarding the manufacturer and exact model of the TPM, as well

[2] `https://github.com/tpm2-software/tpm2-tools`

root@t495s

```
# Obtain the EK certificate from the TPM:
$ tpm2_getekcertificate | openssl x509 -inform DER -out TPM_EKCert.pem

# Print EK certificate in text form to stdout:
$ openssl x509 -in TPM_EKCert.pem -noout -text
[...]

# Obtain the TPM Root CA from the manufacturer's website:
$ URL='https://www.nuvoton.com/security/NTC-TPM-EK-Cert/
↪  Nuvoton%20TPM%20Root%20CA%201110.cer'
$ curl -s ${URL} | openssl x509 -inform der -out TPM_Root_CA.pem

# Verify the TPM EK certificate using the Root CA:
$ openssl verify -CAfile TPM_Root_CA.pem TPM_EKCert.pem
output: TPM_EKCert.pem: OK
```

Listing 1: Printing and verifying the TPM's EK certificate.

as the root CA that signed the certificate. In our case, Nuvoton maintains a document on their website with links for retrieving the root CA certificates [141]. Listing 1 also demonstrates how to verify that the EK certificate was signed with the root CA from Nuvoton.

OS and Software Stack

Windows does not provide the necessary flexibility to customize Secure Boot and the boot process. The mutual authentication solutions discussed in section 3.6.4 are also not available for Windows, so Linux was a better choice for testing these countermeasures. Specifically, Arch Linux was chosen for its flexible installation process, comprehensive documentation, and large package repository with up-to-date software. Arch Linux was installed with an EFI System Partition (ESP) mounted at `/efi` and a root partition encrypted with LUKS2. Instead of a swap partition, a swap file was created within the encrypted root partition. A temporary LUKS password was set during the installation process, which was removed after setting up TPM-based FDE (see section 4.6.3). Instead of installing a traditional bootloader, the system was configured to boot a Unified Kernel Image (UKI) directly, as described in section 4.5.1. To ensure reproducibility, all versions of relevant Arch Linux packages are specified in listing 2.

```
efibootmgr 18-2
efitools 1.9.2-5
linux 6.4.12.arch1-1
qemu-base 8.1.0-2
sbctl 0.11-1
swtpm 0.8.1-1
systemd 254.1-1
tpm2-tools 5.3-2
tpm2-totp-git 0.3.0.r38.826c103-1
tpm2-tss 4.0.1-1
```

Listing 2: Relevant Arch Linux packages and versions used in this setup.

4.2 UEFI Security

The UEFI is a critical aspect of a computer's overall security posture. While users have limited control over the security of their device's UEFI *implementation*, there are certain things to keep in mind regarding its *configuration*. The NSA published a UEFI defense guideline in 2017, which provides security best practices for securing this fundamental system component [142]. This section covers the key points of this guideline that are important in light of our threat model.

4.2.1 UEFI Configuration Lockdown

Many of the countermeasures discussed in the following sections rely on the fact that UEFI settings can only be changed by authorized users. Usually, the mechanism for locking down the UEFI configuration is to set a strong administrator/supervisor password. However, on some machines, merely setting this password does not automatically protect the UEFI configuration. Instead, there is an explicit option to lock UEFI settings in the presence of an administrator password. Furthermore, some UEFI implementations use several credentials beyond just the administrator password. While these are not necessary for our use case, it is worth noting their functionalities:

- *User Password:* Typically constrains non-administrative settings, such as boot device selection.
- *System Password:* Sometimes called power-on password. When this is activated, the boot process is paused, prompting the user to input the password before proceeding.
- *Storage Password:* These are not directly tied to UEFI security. Used for managing ATA/Opal

passwords of storage devices.

On our main test machine, these UEFI settings are located in the *Security* > *Password* menu. There, the supervisor password was set and the *Lock UEFI BIOS settings* option was enabled.

4.2.2 Fast Boot

Some UEFI implementations incorporate a feature called *Fast Boot* or *Minimal Boot* that is intended to minimize boot times by skipping certain boot time checks. Unfortunately, this might also affect security-critical checks, such as Secure Boot signature verification, on some devices [12]. Given that the manufacturer's documentation may not specify the exact security implications of fast boot, it is better to err on the side of caution and disable this feature.

4.2.3 Firmware Updates

Like any software, system firmware requires regular updates to address vulnerabilities discovered throughout its lifetime. With version 2.8, the UEFI specification introduced *Update Capsules*, a universal method for securely updating UEFI firmware. Both Windows and Linux have the necessary infrastructure and tools to update the firmware on supported devices. On Linux, firmware updates can be managed using the `fwupdmgr` utility. Manufacturers distribute these singed update capsules through the Linux Vendor Firmware Service, which `fwupdmgr` uses to find and retrieve updates.

On our primary test machine, support for UEFI update capsules is controlled by the *Windows UEFI firmware update* BIOS option. Additionally, *Secure Rollback Prevention* was enabled to prevent rollback to older, potentially vulnerable firmware versions. With these UEFI settings, `fwupdmgr` was able to update our laptop's system firmware to the latest version. Preparing for a firmware update, `fwupdmgr` copies the capsule and the updater program (`fwupd.efi`) to the ESP. It then creates a temporary EFI boot entry for the updater and reboots the machine to start the update process.

However, when considering the other recommendations in this chapter, two countermeasures interfere with the firmware update process. First, Secure Boot blocks the execution of `fwupd.efi`. The solution is to temporarily disable Secure Boot or, alternatively, sign the `fwupd.efi`, as seen in listing 8. To reduce the attack surface of Secure Boot, it is important to minimize the number of EFI binaries we sign with our keys, so the former approach is preferred. However, this leads to the second countermeasure that interferes with the update process: Disabling Secure Boot changes

```
root@t495s
# Load the tsme-test kernel module:
$ insmod tsme-test.ko

# Check TSME status:
$ cat /sys/kernel/tsme
1

$ dmesg | grep TSME
[21658.850418] TSME Test: TSME is active
```

Listing 3: Verifying the status of TSME on Linux.

the value of PCR 7, which affects our TPM-based FDE solution that is described in section 4.6. This is not a major problem if we disable Secure Boot just before booting `fwupd.efi` and enable it right after the update. In any case, we can use the recovery key deployed in section 4.6.2 to unlock FDE while Secure Boot is disabled.

4.3 Memory Encryption

Looking back at section 3.3, memory encryption is the most effective mitigation against cold boot attacks. AMD's TSME [73] and Intel's TME [74] are transparent memory encryption features, that are enabled through the BIOS and do not require special support from the OS or applications. In contrast, AMD's SME requires OS support to mark memory pages with a special encryption bit [73]. This is controlled via the `mem_encrypt` Linux kernel parameter.

Our test machine supports TSME, thus enabling transparent memory encryption is as simple as enabling the BIOS option at *Security* > *TSME*. From now on, the CPU's memory controller automatically encrypts every page with a random key that is generated at startup. To verify that TSME is active, we can compile and load AMD's `tsme-test` kernel module[3], as seen in listing 3.

4.4 DMA Protection

DMA attacks pose a serious physical access threat to our devices when they are in ACPI states S0–S3. As outlined in section 3.2, protection against DMA attacks is still severely lacking, and the best course of action is to power off or hibernate the device. However, there are some simple

[3] `https://github.com/AMDESE/mem-encryption-tests`

mitigations that protect against the less advanced attack variants and have very little impact from a performance or usability perspective.

4.4.1 Thunderbolt and USB4 Hardening

Systems with Thunderbolt (version 2+) or USB4 come with an access control feature named *Security Levels*. It aims to guard against malicious peripherals, by requiring the user's approval to initialize Thunderbolt devices. Despite its fundamental flaws discussed in section 3.2.2, it is still beneficial to set up, because it at least complicates DMA attacks through external devices. The names of each specific level can differ between device manufacturers, but these are the most common notations in the BIOS:

- *SL0 (No Security):* Disables port authorization completely, sometimes named *Legacy Mode* in the BIOS.
- *SL1 (User Authorization):* Allows the user to authorize devices based on their unique ID, sometimes named *Unique ID* in the BIOS.
- *SL2 (Secure Connect):* Similar to SL1 but with a supposedly stronger challenge-response protcol, sometimes named *One time saved key* in the BIOS.
- *SL3 (DisplayPort and USB only):* Disables PCIe tunneling and only allows the use of DisplayPort and USB protocols over that interface.
- *SL4 (Disable Daisy-Chaining):* Only supported by a few devices, allows PCIe tunneling only for the first device in the chain.

SL3 is the most secure mode and highly preferred when there is an increased demand for security. Of course, this breaks compatibility with Thunderbolt peripherals that require PCIe tunneling, such as Thunderbolt docking stations or external graphics cards. Since pure USB-based peripherals and docking stations are sufficient for most use cases, this should be a viable compromise for most users. Our test machine has no Thunderbolt or USB4 ports, so no configuration steps were necessary.

4.4.2 Disable Unused DMA Ports

In addition to hardening Thunderbolt and USB4, unused PCIe slots should be disabled in the BIOS. This applies not only to desktop computers, which often come with an abundance of PCIe slots, but also to laptops. Our test machine came with an empty slot for WWAN modules, which is an M.2 PCIe slot and therefore DMA capable. It was disabled in the *Security > I/O Port Access*

BIOS menu to reduce this one possible attack vector.

4.4.3 Enabling IOMMU Support

DMA remapping (DMAr) allows the OS to restrict a device's DMA capability to a confined memory region, helping to mitigate DMA attacks. DMAr utilizes the IOMMU, which has to be enabled in the BIOS, usually in the *Security* or *Virtualization* menu. The exact name of the BIOS option depends on the CPU manufacture: On Intel systems, the option is typically called *Intel VT-d* or *Intel VT for directed I/O*. AMD systems usually have an option called *AMD-V* or *SVM* that enables virtualization features such as the IOMMU. On our test machine, the option is found under *Security* > *Virtualization* > *AMD V Technology*.

4.4.4 Using DMA remapping on Linux

Once the IOMMU is enabled in the BIOS, the OS can use DMAr to protect against DMA attacks. While Linux automatically enables DMAr on supported AMD systems, Intel systems typically require adding the `intel_iommu=on` kernel parameter [43]. In addition to enabling DMAr, there are Linux kernel parameters for further hardening against DMA attacks using the IOMMU. The `efi=disable_early_pci_dma` parameter was introduced to disable DMA in the small time window during the boot process, when the IOMMU is not yet initialized.[4] To mitigate the temporal vulnerabilities discussed in section 3.2.4, strict invalidation of IOTLB entries can be enabled with the `iommu.strict=1` parameter. The performance impact of strict invalidation can be significant for IO intensive workloads such as 10+ Gb/s networking [48], but it is unlikely to have a noticeable impact on any typical desktop use case.

After enabling the IOMMU in the BIOS, Linux automatically made use of it on our AMD-based test system. Only the two hardening-related settings had to be manually configured by setting the `efi=disable_early_pci_dma` and `iommu.strict=1` kernel parameters. The status of the IOMMU can be confirmed in the kernel logs, as seen in listing 4.

4.4.5 Address Translation Services

ATS is a performance optimization for Linux that allows PCIe devices to handle IOMMU translations themselves instead of relying on the system's IOMMU. This is problematic from a security

[4] `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=4444f8541dad16fefd9b8807ad1451e806ef1d94`

```
root@t495s
# Check IOMMU status in the kernel log:
$ dmesg | grep -i IOMMU
[    0.429136] pci 0000:00:01.0: Adding to iommu group 0
[    0.429156] pci 0000:00:01.2: Adding to iommu group 1
[    0.429175] pci 0000:00:01.3: Adding to iommu group 2
[...]
[    0.608583] AMD-Vi: AMD IOMMUv2 loaded and initialized
```

Listing 4: Verifying the status of the IOMMU in Linux.

```
root@t495s
# Check which devices support ATS:
$ lspci -v | grep -B 16 ATS
05:00.0 VGA compatible controller: Advanced Micro Devices, Inc. [AMD/ATI]
↪  Picasso/Raven 2
    [...]
    Capabilities: [2b0] Address Translation Service (ATS)

# Check ATS status:
$ dmesg | grep ATS
[    0.000000] PCIe: ATS is disabled
```

Listing 5: Check ATS support and status.

perspective, because a malicious device can exploit ATS to circumvent any IOMMU-based protections [42, p. 97]. ATS can be disabled by setting the `noats` option for the `pci` kernel parameter [43]. In our setup, this corresponds to adding `pci=noats` to `/etc/kernel/cmdline`.

PCIe devices that make use of ATS still work without it, although with a potential loss in performance. Listing 5 shows how to determine which PCIe devices support ATS; in our case, it is only the integrated GPU. Disabling ATS resulted in no perceivable reduction in graphics performance, neither in general desktop usage nor in benchmarks (glsmark2 2023.1). Unfortunately, this parameter had to be reverted because it caused unrecoverable GPU errors after a few hours of use, turning the screen black and requiring a reboot. This leaves our test setup without protection from ATS-based IOMMU bypass attacks.

4.5 Secure Boot

Secure Boot serves as one of our primary defense mechanisms against software-based tampering, particularly in the case of evil maid attacks. When properly configured, Secure Boot ensures that the system will only boot software that has been signed with trusted keys. This section explores the intricacies of implementing Secure Boot for Linux and addresses the associated challenges.

4.5.1 Unified Kernel Images

Secure Boot generally only validates the signature of the *first* component in the boot chain, typically an EFI bootloader such as GRUB or systemd-boot. This bootloader then starts the kernel, which subsequently mounts the initial RAM file system (initramfs). Given that these early boot components reside on the unencrypted EFI boot partition, they are all susceptible to manipulation from evil maid attacks. To provide a meaningful level of protection, each boot component must therefore validate the integrity of the one that follows. This validation requirement extends to bootloader configuration files and the kernel parameter list that are located on the boot partition. Considering that in addition to the EFI firmware, both the bootloader and the kernel must now enforce signature verification on the components they load, it becomes evident why this process can be complex.

Unified Kernel Images (UKIs) can solve this problem by combining all early boot components into a single EFI-executable file [143]. Like other EFI executables, a UKI follows the Portable Executable (PE) file format, which consists of multiple sections, each representing a distinct boot component. First, the UKI contains an EFI boot stub, a rudimentary bootloader that is executed by the EFI. This is followed by sections for the Linux kernel, its command line, the initramfs, and optional CPU microcode, among others. The most important advantage of UKIs is that they can be signed like any other EFI executables, protecting all the components they contain at once. For our setup, this means that we only need one file on our unencrypted boot partition, and that file is verified by Secure Boot to prevent tampering.

Given that they are regular PE files, UKIs can be created with a relatively simple `objcopy` command. However, there are tools that further simplify their creation and signing process, such as `sbctl` [144] and `ukify`[5]. Listing 6 demonstrates how to create a UKI with `sbctl` and a suitable EFI boot entry with `efibootmgr`. Since the latter also changes the boot order to prioritize our

[5] `https://www.freedesktop.org/software/systemd/man/ukify.html`

root@t495s

```
# Create the UKI:
$ sbctl bundle --save \
               --efi-stub /usr/lib/systemd/boot/efi/linuxx64.efi.stub \
               --kernel-img /boot/vmlinuz-linux \
               --cmdline /etc/kernel/cmdline \
               --amducode /boot/amd-ucode.img \
               --initramfs /boot/initramfs-linux.img \
               /efi/EFI/Linux/archlinux-linux.efi

# Create an EFI boot entry:
# (notice how the path is relative to our ESP root)
$ efibootmgr --create \
             --disk /dev/nvme0n1 \
             --part 1 \
             --label "Arch Secure Boot" \
             --loader EFI/Linux/archlinux-linux.efi

BootCurrent: 0000
BootOrder: 0003,0000,0001,[...]
[...]
Boot0003* Arch Secure Boot HD([...]/File(EFI\Linux\archlinux-linux.efi)
```

Listing 6: Creating a UKI and the corresponding EFI boot entry.

newly created entry, the system is now ready to boot the UKI.

4.5.2 Signing UKIs with Custom Keys

As outlined in section 3.6.5, the Secure Boot keys that come preinstalled by the manufacturer cannot be trusted for high security use-cases. Fortunately, they can be replaced with a custom set of keys that are either created manually with `openssl` or with the help of `sbctl`, as seen in listing 7. Among these new keys is the signature database key, which is used to sign UKIs and other EFI executables, allowing them to boot. The signing process using either `sbsign` or `sbctl` is demonstrated in listing 8.

4.5.3 Enrolling Custom Secure Boot Keys

To install our newly generated keys, Secure Boot must be set to *Setup Mode*. On most systems, setup mode can be enabled through a similarly named BIOS option. However, on some ma-

root@t495s

```
# Create the Secure Boot keys:
$ sbctl create-keys
Created Owner UUID d465b0ff-ec78-4464-a1cd-3440ebe43668
[...]
Secure boot keys created!

# List the newly created keys:
$ find /usr/share/secureboot/keys/ -type f
/usr/share/secureboot/keys/PK/PK.key
/usr/share/secureboot/keys/PK/PK.pem
/usr/share/secureboot/keys/db/db.key
/usr/share/secureboot/keys/db/db.pem
/usr/share/secureboot/keys/KEK/KEK.key
/usr/share/secureboot/keys/KEK/KEK.pem
```

Listing 7: Creating a set of Secure Bot keys with `sbctl`.

root@t495s

```
# Sign a UKI with the newly created db key:
$ cd /usr/share/secureboot/keys/db/
$ UKI=/efi/EFI/Linux/archlinux-linux.efi
$ sbsign --key db.key --cert db.crt --output $UKI $UKI
Signing Unsigned original image

# Alternatively, use sbctl:
$ sbctl sign /efi/EFI/Linux/archlinux-linux.efi
Signed /efi/EFI/Linux/archlinux-linux.efi
```

Listing 8: Signing EFI executables with `sbsign` or `sbctl`.

chines, this option may be called *Clear all Secure Boot Keys* instead. On our test system, the appropriate option is located at *Security* > *Secure Boot* > *Reset to Setup Mode*. Switching to setup mode deletes the preinstalled platform key and allows the installation of our own keys. In setup mode, enrolling the keys can be as simple as invoking the `sbctl enroll-keys` command. The `efi-readvar` command can then be used to verify that our custom keys have been enrolled and that no preinstalled keys remain. Once a new platform key is enrolled, Secure Boot exits setup mode and begins enforcing signature verification with our custom keys.

However, this process becomes more involved on systems with certain types of PCIe expansion cards, such as dedicated GPUs or storage controllers. These PCIe components often come with *Option ROMs* that contain their UEFI firmware, which needs to be loaded very early in the boot process. For instance, a machine without integrated graphics relies on the dedicated GPU for video output, even during the early UEFI boot stages. Given that the UEFI cannot be equipped with video drivers for every potential dedicated GPU, it depends on the driver located within the GPU's option ROM. When Secure Boot is active, option ROMs have to pass the same signature validation process as other EFI executables before they are loaded [145]. Generally, option ROMs are signed with Microsoft's third-party UEFI CA or the CA of their respective hardware vendor, both of which are trusted when the preinstalled Secure Boot keys are in place. This becomes a challenge when using Secure Boot with custom keys, because we do not want to trust these CAs and cannot easily re-sign the option ROMs with our own keys either.

One viable solution to this challenge is adding the option ROM's hashes to the *db*. This explicitly allows loading of that particular option ROM, even in the absence of a valid signature. The easiest way of retrieving the hashes is by inspecting the TPM event log, as demonstrated in listing 9. Once collected, these hashes can be converted to an EFI Signature List (ESL) and then integrated into *db*. Again, the `sbctl` utility automates this process of retrieving and adding option ROM hashes, as shown in the third command of listing 9. However, the author of that tool warns that this approach is considered experimental [144]. Special care has to be taken when updating the firmware on these PCIe expansion cards, as the resulting change in the option ROM's hash could prevent the system from booting. Fortunately, this is not a concern with most laptops, as they rarely require any option ROMs.

root@qemu

```
# Search for option ROM measurements in the TPM event log:
$ cp /sys/kernel/security/tpm0/binary_bios_measurements eventlog
$ tpm2_eventlog eventlog | grep -C 10 "BOOT_SERVICES_DRIVER"
[...]
- EventNum: 14
  PCRIndex: 2
  EventType: EV_EFI_BOOT_SERVICES_DRIVER
  DigestCount: 1
  Digests:
  - AlgorithmId: sha256
    Digest: "4019aabdc583879c8a95b2c00c15142911ac6fba9925aacde7[...]"
  EventSize: 84
[...]

# Attempt to enroll Secure Boot keys on a system with option ROMs:
$ sbctl enroll-keys
Found OptionROM in the bootchain. This means we should not enroll keys into UEFI
 ↪  without some precautions.

# Enroll the keys along with option ROM hashes from the TPM event log:
$ sbctl enroll-keys --tpm-eventlog
Enrolling keys to EFI variables...
With checksums from the TPM Eventlog...
Enrolled keys to the EFI variables!
```

Listing 9: Enroll Secure Boot keys with `sbctl` on a system with an option ROM.

4.5.4 Update Handling

To minimize the administrative overhead with Secure Boot on Linux, the UKI should be automatically rebuilt and signed whenever the kernel or CPU microcode is updated. When installed from the official Arch Linux repositories, the `sbctl` package already includes a package manager hook that facilitates this process. Adapting this to other Linux distributions should be trivial, given that most package managers offer similar hooking-functionality.

The bigger challenge lies in establishing rollback protection to prevent the following attack scenario: An attacker could collect old UKIs and wait until a vulnerability within one of them is discovered. This could be a Linux kernel vulnerability or a flaw within another UKI component that allows loading arbitrary code, effectively bypassing Secure Boot. Because the vulnerable UKI still features a valid signature, it could now be used for a tamper-and-revert attack. With our proposed Secure Boot setup, this threat is less pronounced, because every computer uses a distinct set of Secure Boot keys. To exploit this, an attacker would require regular physical access to the target computer to copy the UKIs, which makes this scenario less likely. Nevertheless, we will explore two approaches for implementing a mechanism that prevents old UKIs from being executed, each with its own set of challenges.

The most straightforward method is to append the hash of every deprecated UKI to the *dbx*, thereby explicitly prohibiting its execution regardless of its signature. However, there are two limitations to keep in mind with this approach. First, it is not infinitely sustainable because the *dbx* only has 32 KB of capacity[6], which amounts to about 500 hashes. Should this limit ever be reached, rotating the Secure Boot keys and resetting the *dbx* would be a viable solution. The second limitation is related to our TPM-based FDE proposal described in section 4.6. Every update to the *dbx* interferes changes PCR 7, which is our preferred PCR for FDE key sealing. This PCR was chosen for key sealing specifically because of its low fragility, as discussed in section 4.6.2. Using the *dbx* for rollback protection would therefore require an alternative approach to TPM-based FDE.

Another approach utilizes a special feature of TPMs: monotonic counters. The TCG specifies these as integer counters that can only be incremented, never decremented [13]. When sealing secrets using the TPM, future unsealing can be controlled not only by a set of PCR values, but also by specific counter values or ranges. The safeboot project already uses this mechanism for rollback protection [9].

[6] `https://github.com/rhboot/shim/blob/main/SBAT.md`

4.6 TPM-based FDE

TPM-based FDE serves a multitude of purposes in protecting against physical access attacks. First, it dramatically reduces the negative security impact of a low-entropy FDE password by providing anti-hammering protection. Furthermore, it mitigates or at least complicates some forms of evil maid attacks discussed in section 3.6.

4.6.1 Available Solutions

In 2022, Wetzels compared five tools that facilitate TPM-based FDE on Linux, focusing on their resilience to TPM-sniffing [146]. Since then, most of these tools received feature updates, warranting a new comparison. This re-evaluation was conducted by analyzing their source code and available documentation. As a reference, Microsoft's BitLocker was also included in the comparison, despite its lack of source code availability. The data for BitLocker had to be deducted from Microsoft's technical documentation instead [16]. The results are summarized in table 4.1.

Of all the solutions examined, only systemd-cryptenroll[7] uses parameter encryption and verifies the authenticity of the TPM when unsealing the FDE key. Interestingly, while safeboot [9] uses parameter encryption for some of its functionality, it does not do so for FDE. Except for Clevis[8], all solutions support the TPM + PIN combination. However, as elaborated in section 3.5.2, only BitLocker implements this as a robust second factor that can withstand a compromised TPM. The Linux solutions use the PIN only to authenticate to the TPM, not as part of the key derivation function. In addition, only BitLocker allows the use of a key file as a true second or third factor with TPM-based FDE. While *LUKS TPM2*[9] uses TPM-sealed key files, it does not support storing them externally.

In conclusion, systemd-cryptenroll currently stands out as the best choice for Linux, despite its limitations. While its insecure TPM + PIN implementation presents a considerable risk for fTPMs, it is less of an issue for dTPMs due to their greater resilience to compromise.

4.6.2 Challenges

When deploying TPM-based FDE, a core decision revolves around the selection of PCRs that represent the conditions under which the key can be unsealed. The goal is to permit unsealing of the

[7] `https://www.freedesktop.org/software/systemd/man/systemd-cryptenroll.html`
[8] `https://github.com/latchset/clevis`
[9] `https://github.com/electrickite/luks-tpm2`

Table 4.1: Feature comparison of TPM-based FDE solutions for Linux.

Solution (version)	Param. enc.	TPM authenticity check	TPM + PIN	TPM + key file
Clevis (19)	-	-	-	-
LUKS TPM2 (2.1.2)	-	-	◐	◐
safeboot (0.7)	-	-	◐	-
systemd-cryptenroll (254)	●	●	◐	-
BitLocker	-	-	●	●

● = implemented; ◐ = partially/insecurely implemented; - = not implemented;

FDE key exclusively when the system is in a trustworthy state. Selecting the wrong set of PCRs can lead to fragility, where benign configuration changes or updates result in changed PCR values that prevent unsealing. For instance, selecting PCR 0 for sealing the FDE key would render the key inaccessible after a BIOS update. This means a BIOS update would require the cumbersome procedure of temporarily suspending FDE and resealing the encryption key afterward. The ideal scenario is to select PCRs that attest the system's trustworthiness but remain unaffected by specific software or firmware version changes.

Microsoft addresses this challenge by primarily relying on PCR 7 for UEFI-based systems [100]. PCR 7 contains the measurement results of the system's Secure Boot state and its associated keys. When PCR 7 holds the expected value, it implies that the booted EFI executable was signed by the Secure Boot keys that we trust. One could argue that relying solely on PCR 7 is too permissive for systems that trust the default set of Secure Boot keys. This is a reasonable position to take since these keys have been used to sign thousands of EFI executables, many of which now have known vulnerabilities, as discussed in section 3.6.5. However, in our specific setup, the Secure Boot keys are unique to each machine and are used exclusively for signing UKIs. In this context, PCR 7 provides sufficient assurance regarding the system's state and avoids PCR fragility.

Another important aspect to consider is the recovery strategy in the event of a TPM failure. In such cases, a high-entropy recovery key that can decrypt the data independently of the TPM is essential. This key should be enrolled and presented to the user during setup, who must then store it securely. Listing 10 demonstrates how to enroll a suitable recovery key with the `systemd-cryptenroll` utility. If a situation arises that necessitates the use of the recovery key, it is essential to follow the security considerations in section 4.9.3.

root@t495s

```
# Enroll a recovery key:
$ systemd-cryptenroll --recovery-key /dev/vda2
Please enter current passphrase for disk /dev/vda2: [...]
A secret recovery key has been generated for this volume:

    rdrfbhjf-bbifvnic-[...]

[...]
New recovery key enrolled as key slot 1.
```

Listing 10: Enrolling a high-entropy recovery key with `systemd-cryptenroll`.

4.6.3 Enrolling TPM-based FDE

Once the prerequisites are addressed, enrolling TPM-based FDE becomes a simple process, as shown in listing 11. First, the TPM-based key slot for LUKS is enrolled with the `systemd-cryptenroll` command, specifying PCR 7 and a PIN for TPM authentication. Despite the use of the word PIN, the character set is not limited to digits. On devices equipped with multiple TPMs (e.g., fTPM and dTPM), it is important to specify the exact device instead of `auto`. Then, a few adjustments are made to the kernel command line and initramfs to use `systemd-cryptsetup` instead of `cryptsetup`. Now, the system should present a prompt that reads *Please enter LUKS2 token PIN* at boot time, and the drive will be unlocked after entering the PIN. If any password-based key slots were configured previously, they should now be removed using the `cryptsetup luksRemoveKey` command, ensuring only the TPM and recovery key remain.

4.7 Mutual Authentication

While it is very common for users to authenticate to a computer, the inverse, where a computer proves its authenticity to a user, is less prevalent. Before trusting a computer with secret keys, users should be able to verify its authenticity and integrity to rule out the possibility that it is controlled by an attacker. The challenges of implementing such a mechanism robustly have been explored in section 3.6.7. While current solutions may not be able to withstand the most sophisticated replace-and-relay attacks, they still offer value by raising the bar against such security threats.

root@t495s

```
# Enroll TPM-based LUKS key slot:
$ systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=7 --tpm2-with-pin=yes
 ↪  /dev/nvme0n1p2
Please enter current passphrase for disk /dev/nvme0n1p2: [...]
Please enter TPM2 PIN: [...]
Please enter TPM2 PIN (repeat): [...]
New TPM2 token enrolled as key slot 2.

# Adapt the initramfs for systemd-cryptsetup:
$ sed -i '/s/^HOOKS.*/HOOKS=(base systemd autodetect modconf kms keyboard
 ↪  sd-vconsole block sd-encrypt filesystems fsck)/' /etc/mkinitcpio.conf

# Adapt the kernel parameters for systemd-cryptsetup-generator:
$ cat > /etc/kernel/cmdline « EOF
rd.luks.name=UUID=root rd.luks.options=UUID=tpm2-device=auto root=/dev/mapper/root
 ↪  rw [...]
EOF

# Rebuild the initramfs and UKI:
$ mkinitcpio -P && sbctl generate-bundles -s
[...]
```

Listing 11: Enrolling TPM-based FDE with `systemd-cryptenroll`.

4.7.1 Available Solutions

Sections 3.6.2 and 3.6.4 gave an overview of the mutual authentication solutions available for conventional computers. On the one hand, there is the solution by Rutkowska and its successors. The most recent of these is tpm2-totp, which is still actively maintained [108]. On the other hand, there are solutions that require specialized external hardware, akin to the iTurtle proposed by McCune *et al.* in 2007 [109]. One of these is *MARK*, which was released in 2014 but has not received any updates beyond the initial academic prototype [110]. The more recent device that provides such a functionality comes from Nitrokey[10] and is a commercially available product. However, this functionality is exclusive to computers running the coreboot firmware, which is a very limited number of devices. Given these limitations, tpm2-totp is the only viable solution for our setup.

4.7.2 Enrolling TOTP-based Anti Evil Maid

Like TPM-based FDE, tpm2-totp works by sealing a key with the TPM and defining a policy that specifies the conditions under which the key can be unsealed. The key is then used to compute a dynamic six-digit code that the user must validate before entering the FDE password. This raises the same question regarding PCR selection that was addressed in section 4.6.2. The exclusive use of PCR 7 is a reasonable choice here as well.

The process of enrolling tpm2-totp is detailed in listing 12. Once the TOTP secret has been generated, it is displayed as a QR code to the user, who must scan it with a TOTP smartphone application. After adjusting the initramfs to include the `sd-tpm2-totp` hook, a six-digit code will be displayed in future boot processes. It is essential that the clocks of the computer and phone align closely, ideally with only a minor delay of a few seconds. This synchronization is easily achieved if both devices set their clocks using an NTP server.

4.8 Physical Tamper-Evident Technology

The mutual authentication solution discussed in the previous section acts as a form of tamper-evident technology, but it is limited to detecting software-based threats. When faced with hardware keyloggers or implants, there is an additional need for *physical* tamper-evident technology. Various such technologies have been covered in section 3.6.10 with their respective strengths and

[10] `https://www.nitrokey.com/files/doc/Nitrokey_Pro_factsheet.pdf`

root@t495s

```
# Generate a new TOTP secret:
$ tpm2-totp generate -p 7
[...]
otpauth://totp/TPM2-TOTP?secret=JDZTB[...]

# Generate a six-digit code to compare it with the TOTP app:
$ tpm2-totp -t calculate
2023-09-05 17:11:17: 263806

# Add the sd-tpm2-totp hook before sd-encrypt:
$ sed -i '/^HOOKS/s/sd-encrypt/sd-tpm2-totp sd-encrypt/' /etc/mkinitcpio.conf

# Rebuild the initramfs and UKI:
$ mkinitcpio -P && sbctl generate-bundles -s
[...]
```

Listing 12: Enrolling tpm2-totp and adapting the initramfs.

limitations. We have also discussed the trade-off between the sensitivity and specificity of these solutions and the benefits of combining solutions of both categories.

4.8.1 Case Intrusion Detection

The case tamper switch on our test laptop serves as the high-specificity solution in our suite of safeguards. When the bottom cover is removed, it displays an alert during the next boot, requiring the supervisor password to continue. On our machine, this is configured through the BIOS at *Security* > *Internal Device Access*. As noted in the BIOS documentation, this feature only takes effect if a supervisor password is set [135]. In addition, there is an option to treat power loss or emergency reset events in the same way as bottom cover removals. This is important because it eliminates one of the ways to bypass the tamper detection.

Four experiments were conducted to test the effectiveness of our machine's tamper detection mechanism. These tests involved removing the bottom cover under various conditions: while the machine was operational, during sleep mode, when it was powered off, and following a power loss event. In all four cases, the alert and password prompt appeared during the next boot cycle. Nonetheless, this reveals a significant limitation of the feature: It only notifies the user upon the next boot-up. This delay could be critical if tampering occurs while the system is active or in sleep mode. Not all case intrusion detection mechanisms suffer from this limitation. HP's TamperLock

documentation explicitly advertises support for forcing a shutdown upon cover removal [134].

4.8.2 Active Tamper Detection

The Haven app [136], referenced in section 3.6.10, provides a useful addition to our device's native tamper detection capabilities, due to its high sensitivity. It was installed on an old Nokia 7.1 smartphone running Android 10 and configured to monitor the device's camera, microphone, ambient light sensor, and accelerometer. Each sensor's detection threshold can be customized within the app. With the Haven-enabled phone placed on top of the laptop, and both stored in a drawer, it was not possible to open the drawer and retrieve the laptop without triggering the light sensor or camera. However, it was possible to avoid accelerometer motion detection by performing these actions very slowly. A determined attacker could therefore attempt to do this in complete darkness and slowly enough not to trigger any sensors. Another limitation is Haven's current inability to send proactive alerts, requiring users to manually inspect the Haven logs instead. While this limits Haven's effectiveness, it is still a valuable tool in high-risk environments.

4.9 The Human Element

So far, this chapter has focused on technical mitigations against physical access attacks. However, there are certain attack vectors that cannot be fully prevented by these technical measures alone. Protecting against them requires the proactive participation of informed users. This section discusses the key steps users can take to bridge these security gaps.

4.9.1 ACPI States

The impact of a device's ACPI state on its attack surface has been highlighted throughout this thesis. For instance, while DMA attacks are an unavoidable threat to devices in states S0 to S3, they are almost irrelevant when a device is powered off or in hibernation. And while we have configured a mutual authentication scheme to verify the integrity of our device at boot time, there is no such solution for the lock screen. Consequently, our hardened device is more vulnerable to replace-and-relay attacks when it is running or in sleep mode.

It is therefore generally advisable that users either power off their devices or put them into hibernation mode when leaving them unattended. The slight delay when waking a device from hibernation compared to sleep mode is a small price to pay for the considerably smaller attack

surface it provides. Sleep mode can still be a useful tool when users want to save energy in low-risk situations. The delayed hibernation functionality in Windows [100] and Linux[11] can also be useful. In this mode, the system automatically wakes from sleep and enters hibernation after a configurable delay.

Making the right choices regarding ACPI states does not have to be the sole responsibility of the user. Operating systems could do a better of job informing users about the security implications of certain power states. For centrally managed devices, administrators can force delayed hibernation or disable sleep mode altogether.

4.9.2 Device Inspection

Effective tamper detection is not only about technical solutions, but also about how users respond to them. Passive measures, like tamper-evident seals and tpm2-totp, rely on the user's active inspection to determine the trustworthiness of the device. Conversely, active solutions, such as our device's case intrusion sensors, proactively alert users to potential attacks. Yet, they still rely on the user to take the appropriate action to prevent the attack.

The appropriate action depends on the tamper detection solution and the situation in which it was triggered. For example, if a high-sensitivity solution like Haven detects movements in a busy office environment, it could very well be a false positive. In such scenarios, high-specificity solutions, such as case intrusion sensors or seals, should be used to verify or dismiss the possible incident. However, if such a detection occurs while the device is locked in a safe, there is compelling evidence of unauthorized access. Whenever there is sufficient doubt regarding the trustworthiness of the computer, it should no longer be used.

4.9.3 FDE Recovery

Previous research indicates that FDE recovery mechanisms can be exploited as a vector for evil maid attacks [4]. Therefore, users must carefully evaluate situations that require the use of the recovery key deployed in section 4.6.2. If the computer fails to unseal the FDE key after an update or configuration change, and there are no other signs of an attack, it is safe to enter the recovery key to unlock the computer. On the other hand, if the computer unexpectedly asks for the recovery key at boot time, this could be a sign of tampering. In such cases, users are advised not to enter any confidential information on the potentially compromised computer. The safest way to

[11] `https://www.freedesktop.org/software/systemd/man/systemd-sleep.conf.html`

Table 4.2: Results of the hardening process.

	Drive-by Attack									Evil Maid Attack					
Countermeasure	DMA attack (ext. port)	DMA attack (int. port)	Cold boot attack	SED attacks	TPM sniffing	TPM reset attack	Platform reset attack	fTPM fault injection	Invasive silicon attacks	Original evil maid	Tamper & revert	Replace & relay	Advanced R&R	Hardware keylogger	Hardware implants
DMA port authorization	◐	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DMA remapping	◐	◐	-	-	-	-	-	-	-	-	-	-	-	-	-
Memory encryption	-	-	●	-	-	-	-	-	-	-	-	-	-	-	-
systemd TPM + PIN FDE	-	-	-	●	●	●	●	-	-	●	◐	◐	-	◐	-
discrete TPM	-	-	-	-	-	-	-	●	-	-	-	-	-	-	-
Power off or hibernate	●	●	●	◐	-	-	-	-	-	-	-	-	-	-	-
TOTP-based AEM	-	-	-	-	-	-	-	-	-	●	●	◐	-	-	-
Hardened Secure Boot	-	-	-	-	-	◐	●	-	-	●	●	-	-	-	-
Case intrusion sensors	-	-	-	-	-	-	-	-	-	-	-	-	-	◐	◐
Haven app	-	-	-	-	-	-	-	-	-	●	●	●	●	◐	◐
Sum of countermeasures	●	●	●	●	●	●	●	●	-	●	●	●	●	◐	◐
Reference: Table 3.1	●	●	●	●	●	●	●	●	●	●	●	●	●	●	◐

● = mitigated; ◐ = partially mitigated; - = not mitigated;

recover data in these scenarios is to connect the computer's encrypted storage device to a trusted workstation and use the recovery key there.

4.10 Summary and Results

In this chapter, we explored the process of hardening our Linux-based laptop against physical access attacks using existing countermeasures. We were able to implement countermeasures for most of the attacks discussed in chapter 3, but there are some attack vectors that remain unaddressed. Table 4.2 summarizes the results of this chapter and highlights the remaining vulnerabilities.

4.10.1 Limitations

Comparing the results with the expectations in table 3.1, there are two distinct attack vectors that we could not mitigate to the desired degree. In both cases, this was due to shortcomings in

the countermeasures available on Linux. There were other instances, where solutions did not work as advertised and only provided partial protection. As of September 2023, these limitations include:

- The kernel parameter for protecting against IOMMU bypass via ATS caused instabilities on our test machine and had to be disabled. There is also no working solution to prevent spatial IOMMU vulnerabilities in the Linux kernel version we used. Without these protections, our machine remains vulnerable to more advanced types of DMA attacks.
- There is currently no TPM-based FDE solution for Linux that securely implements the TPM + PIN combination discussed in section 3.5.2. Until systemd merges the related patch[12], the FDE key is protected only by the TPM. While this would present a significant risk for fTPMs, it is less of an issue for our setup because dTPMs are more resistant to compromise.
- There is currently no FDE tool for Linux that implements the TPM + key file combination, leaving our setup vulnerable to keylogging.
- The Haven app, in its current state, lacks proactive alert mechanisms, requiring the user to actively inspect the logs instead.

4.10.2 Future Work

While the scope of this chapter was confined to the integration of existing countermeasures to create an optimally protected system, the presented landscape offers significant opportunities for improvement:

- Proposals to improve DMA protection on Linux date back to 2016. The work of Markuze *et al.* describes an alternative usage model of the IOMMU that eliminates both spatial and temporal vulnerabilities while ensuring minimal performance degradation [48]. Coupled with more robust protection against IOMMU bypass vulnerabilities, this could pave the way for computers that are reasonably secure in ACPI states S0 to S3.
- Implementing an FDE solution that is compatible with LUKS but provides more secure key protection schemes would greatly improve the current situation on Linux. Similar to BitLocker, PINs and key files should serve as a true second or third factor in protecting the KEK.
- The limitations of current tamper detection and prevention systems highlight the need for a cross-platform solution that is not limited to specialized computers. Our EMMA proposal

[12] `https://github.com/systemd/systemd/pull/27502`

offers some ideas for realizing such a solution.

In addition to these solution-oriented efforts, there is always a need for offensive security research to validate the effectiveness of these countermeasures. Some of the solutions used in this chapter, such as hardware-based memory encryption, are relatively new and may not have received the necessary scrutiny from independent security researchers.

5 Conclusion

Physical access threats, ranging from cold boot to evil maid attacks, underscore the fact that we cannot rely solely on FDE to protect the sensitive data on our devices. To address these threats, we first needed to gain a thorough understanding of the underlying attack vectors. Armed with this knowledge, we delved deep into the process of hardening our systems using the available countermeasures, highlighting not only the strengths of these security solutions, but also their weaknesses.

Our comprehensive literature review in chapter 3, summarized in table 3.1, painted an optimistic picture. We concluded that, except for hardware implants, all attacks could be prevented with readily available countermeasures. However, our conclusion did not consider whether these countermeasures can deliver what they promise and how well they can be used in combination with each other. Chapter 4 attempted to answer these questions by walking through the process of selecting and configuring the right hardware and software to build a hardened Linux-based setup. It concluded with table 4.2, showing that some of our security goals were not achievable with current solutions.

A significant gap in the defense of our system is the lack of a robust TPM-based FDE solution for Linux. This leaves our system susceptible to attacks aimed at extracting secret keys by compromising the TPM itself. While the discrete TPM in our setup presents a challenge to attackers—requiring specialized equipment and expertise—this attack vector could have been avoided if Linux FDE schemes were implemented as robustly as BitLocker. Similarly, no Linux FDE solution supports the *TPM + key file* scheme that is required to mitigate the threats posed by hardware keyloggers.

We found that our only viable high-sensitivity tamper detection solution, the Haven app for Android, failed to provide the active alerting capabilities it advertises. Furthermore, our device has very limited physical tamper *prevention* capabilities, and we found no aftermarket solutions to supplement this capability. These limitations highlight the need for a cross-platform solution that is not limited to specialized computers. Our EMMA proposal offers some ideas for realizing

such a solution.

To counter DMA attacks, our only option at the moment is to advise users to always power off or hibernate their devices when left unattended. Looking forward, future improvements in DMA protection may pave the way for computers that are reasonably safe from physical access attacks, even when they are running or in sleep mode.

In closing, while we have made significant progress in understanding and mitigating physical access threats, there is still a long way ahead. In addition to developing solutions to the aforementioned challenges, there is a pressing need for proactive offensive security research to evaluate the effectiveness of our countermeasures. And as the offensive side of security research continues to evolve and uncover new and improved attacks, research on defensive research will be required to keep pace.

List of Tables

List of Listings

Acronyms

ACPI	Advanced Configuration and Power Interface
AES	Advanced Encryption Standard
AES-NI	AES New Instructions
ATS	Address Translation Services
BIOS	Basic Input/Output System
CA	Certificate Authority
CPU	Central Processing Unit
CRTM	Core Root of Trust for Measurement
db	Signature Database
dbx	Forbidden Signature Database
DEK	Data Encryption Key
DMA	Direct Memory Access
DMAr	DMA remapping
DRTM	Dynamic RTM
dTPM	discrete TPM
EFI	Extensible Firmware Interface
EK	Endorsement Key
EMMA	Evil Maid Misconduct Avoidance
ESL	EFI Signature List
ESP	EFI System Partition

FDE	Full Disk Encryption
FPGA	Field Programmable Gate Array
fTPM	firmware TPM
FVEK	Full Volume Encryption Key
HMAC	Hash-based Message Authentication Code
initramfs	initial RAM file system
IOMMU	Input-Output Memory Management Unit
IOTLB	Input-Output Translation Lookaside Buffer
IOVA	I/O Virtual Address
KEK	Key Encryption Key
KEK*	Key Exchange Key
LPC	Low Pin Count
LUKS	Linux Unified Key Setup
MBR	Master Boot Record
MitM	Man-in-the-Middle
MMU	Memory Management Unit
MOR	Memory Overwrite Request
NVRAM	non-volatile RAM
OS	Operating System
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PCR	Platform Configuration Register

PE	Portable Executable
PIN	Personal Identification Number
PK	Platform Key
PKI	Public Key Infrastructure
RAM	Random-Access Memory
ROM	Read-Only Memory
S0	ACPI state S0 (running)
S3	ACPI state S3 (sleep)
S4	ACPI state S4 (hibernate)
S5	ACPI state S5 (off)
SED	Self-Encrypting Drive
SGX	Software Guard Extensions
SHA	Secure Hash Algorithm
SME	Secure Memory Encryption
SRTM	Static Root of Trust for Measurement
SSC	Security Subsystem Class
TCG	Trusted Computing Group
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TME	Total Memory Encryption
TOFU	Trust On First Use
TOTP	Time-based One-Time Password
TPM	Trusted Platform Module
TSME	Transparent Secure Memory Encryption
UEFI	Unified Extensible Firmware Interface
UKI	Unified Kernel Image
USB	Universal Serial Bus

UUID	Universally Unique IDentifier
VM	Virtual Machine
VMK	Volume Master Key
VT-d	Virtualization Technology for Directed I/O

Bibliography

[1] David Hylender, Philippe Langlois, Alex Pinto, and Suzanne Widup, "Data breach investigations report," *Verizon DBIR*, 2023.

[2] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten, "Lest we remember: Cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, pp. 91–98, 2009-05.

[3] Joanna Rutkowska and Alexander Tereshkin, "Evil maid goes after truecrypt," *The Invisible Things Lab*, 2009. [Online]. Available: `https://blog.invisiblethings.org/2009/10/15/evil-maid-goes-after-truecrypt.html`.

[4] Sven Türpe, Andreas Poller, Jan Steffan, Jan-Peter Stotz, and Jan Trukenmüller, "Attacking the bitlocker boot process," in *International Conference on Trusted Computing*, Springer, 2009, pp. 183–196.

[5] Hassan Mohamad, "Physical access attacks against unattended computers," Bachelor's Thesis, St. Pölten University of Applied Sciences, 2021.

[6] T. Müller and F. C. Freiling, "A systematic assessment of the security of full disk encryption," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, pp. 491–503, 2015.

[7] Johannes Götzfried, "Trusted systems in untrusted environments: Protecting against strong attackers," doctoralthesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2017.

[8] Trammell Hudson. "Heads: the other side of TAILS." (2016), [Online]. Available: `https://github.com/osresearch/heads` (visited on 2023-09-03).

[9] Trammell Hudson. "Safe Boot: Booting Linux Safely." (2020), [Online]. Available: `https://github.com/osresearch/safeboot` (visited on 2023-09-03).

[10] Wilkins Richard and Richardson Brian, *UEFI Secure Boot in modern computer security solutions*, 2013.

[11] "Unied Extensible Firmware Interface (UEFI) Specication, Release 2.10," UEFI Forum, Inc., Tech. Rep., 2023. [Online]. Available: `https://uefi.org/sites/default/files/resources/UEFI_Spec_2_10_Aug29.pdf`.

[12] "UEFI Secure Boot Customization," National Security Agency, Tech. Rep., 2020.

[13] "Trusted Platform Module Library Specification, Family 2.0, Revision 01.59," Trusted Computing Group (TCG), Tech. Rep., 2019.

[14] "TPM 2.0 - A Brief Introduction," Trusted Computing Group (TCG), Tech. Rep., 2019. [Online]. Available: `https://trustedcomputinggroup.org/wp-content/uploads/2019_TCG_TPM2_BriefOverview_DR02web.pdf`.

[15] "TCG PC Client Platform Firmware Profile Specification, Family 2.0, Version 1.05 Revision 23," Trusted Computing Group (TCG), Tech. Rep., 2021.

[16] Microsoft. "Bitlocker drive encryption technical overview." (2012), [Online]. Available: `https://web.archive.org/web/20230709213454/https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc732774%28v=ws.10%29` (visited on 2023-07-09).

[17] "TCG Storage Security Subsystem Class: Opal, Version 2.02, Revision 1.0," Trusted Computing Group (TCG), Tech. Rep., 2022.

[18] Microsoft. "Encrypted hard drive device guide." (2011), [Online]. Available: `http://go.microsoft.com/fwlink/p/?LinkId=227287` (visited on 2023-07-16).

[19] "Advanced Configuration and Power Interface (ACPI) Specification, Release 6.5," UEFI Forum, Inc., Tech. Rep., 2022. [Online]. Available: `https://uefi.org/sites/default/files/resources/ACPI_Spec_6_5_Aug29.pdf`.

[20] Tilo Müller, Tobias Latzo, and Felix C Freiling, "Self-encrypting disks pose self-decrypting risks," in *the 29th Chaos Communinication Congress*, 2012, pp. 1–10.

[21] Daniel Boteanu and Kevvie Fowler, "Bypassing self-encrypting drives (SED) in enterprise environments," *BlackHat Europe*, 2015.

[22] Maximillian Dornseif, "0wned by an iPod," *Presentation, PacSec*, 2004.

[23] Michael Becher, Maximillian Dornseif, and Christian N Klein, "Firewire: All your memory are belong to us," *Proceedings of CanSecWest*, p. 67, 2005.

[24] Adam Boileau, "Hit by a bus: Physical access attacks with FireWire," *Presentation, Ruxcon*, vol. 3, 2006.

[25] Benjamin Böck, "Firewire-based physical security attacks on Windows 7, EFS and BitLocker," *Secure Business Austria Research Lab*, 2009.

[26] David Hulton, "Cardbus bus-mastering: 0wning the laptop," *Proceedings of ShmooCon*, vol. 6, 2006.

[27] Damien Aumaitre and Christophe Devine, "Subverting Windows 7 x64 kernel with DMA attacks," *HITBSecConf Amsterdam*, 2010.

[28] Joe FitzPatrick and Miles Crabill, "Stupid PCIe Tricks," *44CON*, 2014.

[29] Ulf Frisk, "Direct memory attack the kernel," *DEFCON*, vol. 24, 2016.

[30] ALEX Ionescu, "Getting physical with USB Type-C," *Recon Brussels*, 2017.

[31] Ulf Frisk. "PCILeech." (2016), [Online]. Available: `https://github.com/ufrisk/pcileech` (visited on 2023-06-14).

[32] A Theodore Markettos, Colin Rothwell, Brett F Gutstein, Allison Pearce, Peter G Neumann, Simon W Moore, and Robert NM Watson, "Thunderclap: Exploring vulnerabilities in operating system IOMMU protection via DMA from untrustworthy peripherals," in *NDSS*, 2019.

[33] "Thunderbolt 3 and Security on Microsoft Windows 10 Operating system," Intel Corporation, Tech. Rep., 2018.

[34] Björn Ruytenberg, *Breaking Thunderbolt Protocol Security: Vulnerability Report*, Public version., 2020. [Online]. Available: `https://thunderspy.io/assets/reports/breaking-thunderbolt-security-bjorn-ruytenberg-20200417.pdf`.

[35] Intel. "Thunderbolt security helps keep your computer ports more secure." (2020), [Online]. Available: `https://www.mouser.com/pdfDocs/thunderbolt-security-brief.pdf` (visited on 2023-05-24).

[36] Rajib Dutta, Matt Chung, and Xavier Chu. “USB4 on Windows.” (2019), [Online]. Available: https://web.archive.org/web/20230611151613/https://www.usb.org/sites/default/files/D1T2-2%20-%20USB4%20on%20Windows.pdf (visited on 2023-06-11).

[37] “Intel Virtualization Technology for Directed I/O Architecture Specification, revision 3.2,” Intel Corporation, Tech. Rep., 2020.

[38] “AMD I/O Virtualization Technology (IOMMU) Specification, revision 3.07,” Advanced Micro Devices, Inc., Tech. Rep., 2022.

[39] “Apple Platform Security,” Apple Inc., Tech. Rep., 2022.

[40] Microsoft. “Kernel DMA protection.” (2022), [Online]. Available: https://web.archive.org/web/20220921221152/https://learn.microsoft.com/en-us/windows/security/information-protection/kernel-dma-protection-for-thunderbolt (visited on 2022-09-21).

[41] Microsoft. “Kernel DMA protection.” (2023), [Online]. Available: https://web.archive.org/web/20230611141427/https://learn.microsoft.com/en-us/windows/security/information-protection/kernel-dma-protection-for-thunderbolt (visited on 2023-06-11).

[42] Colin Lewis Rothwell, “Exploitation from malicious PCI express peripherals,” Ph.D. dissertation, University of Cambridge, 2017.

[43] Linux kernel development community. “The kernel’s command-line parameters.” (2023), [Online]. Available: https://web.archive.org/web/20230611161539/https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html (visited on 2023-06-11).

[44] Lu Baolu. “IOMMU/VT-d: Several minor adjustments.” (2021), [Online]. Available: https://lore.kernel.org/linux-iommu/20210720013856.4143880-4-baolu.lu@linux.intel.com/T/ (visited on 2023-06-11).

[45] Rafal Wojtczuk, Joanna Rutkowska, and Alexander Tereshkin, “Another way to circumvent intel trusted execution technology,” *Invisible Things Lab*, pp. 1–8, 2009.

[46] Rafal Wojtczuk and Joanna Rutkowska, “Following the white rabbit: Software attacks against intel VT-d technology,” *Invisible Things Lab*, 2011.

[47] Benoît Morgan, Eric Alata, Vincent Nicomette, and Mohamed Kaâniche, "Bypassing IOMMU Protection against I/O Attacks," in *7th Latin-American Symposium on Dependable Computing (LADC'16)*, ser. Proceeding of the 7th Latin-American Symposium on Dependable Computing (LADC'16), Cali, Colombia, 2016-10, pp. 145–150. DOI: 10.1109/LADC.2016.31. [Online]. Available: https://hal.science/hal-01419962.

[48] Alex Markuze, Adam Morrison, and Dan Tsafrir, "True IOMMU Protection from DMA Attacks: When Copy is Faster than Zero Copy," *SIGPLAN Not.*, vol. 51, no. 4, pp. 249–262, 2016-03, ISSN: 0362-1340. DOI: 10.1145/2954679.2872379. [Online]. Available: https://doi.org/10.1145/2954679.2872379.

[49] Gil Kupfer, Dan Tsafrir, and Nadav Amit, "IOMMU-resistant DMA attacks," M.S. thesis, Computer Science Department, Technion, 2018.

[50] Alex Markuze, Shay Vargaftik, Gil Kupfer, Boris Pismeny, Nadav Amit, Adam Morrison, and Dan Tsafrir, "Characterizing, exploiting, and detecting dma code injection vulnerabilities in the presence of an iommu," in *Proceedings of the Sixteenth European Conference on Computer Systems*, ser. EuroSys '21, Online Event, United Kingdom: Association for Computing Machinery, 2021, pp. 395–409, ISBN: 9781450383349. DOI: 10.1145/3447786.3456249. [Online]. Available: https://doi.org/10.1145/3447786.3456249.

[51] Alex Markuze, Igor Smolyar, Adam Morrison, and Dan Tsafrir, "DAMN: Overhead-Free IOMMU Protection for Networking," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18, Williamsburg, VA, USA: Association for Computing Machinery, 2018, pp. 301–315, ISBN: 9781450349116. DOI: 10.1145/3173162.3173175.

[52] Björn Ruytenberg, "When lightning strikes thrice: Breaking thunderbolt security," M.S. thesis, Eindhoven University of Technology, 2022.

[53] Walter Link *et al.*, "Eigenschaften von MOS-Ein-Transistorspeicherzellen bei tiefen Temperaturen," 1979.

[54] Steve H Weingart, "Physical security for the μABYSS system," in *1987 IEEE Symposium on Security and Privacy*, IEEE, 1987, pp. 52–52.

[55] Sergei Skorobogatov, "Low temperature data remanence in static RAM," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-536, 2002-06. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.pdf.

[56] M. Gruhn and T. Müller, "On the practicability of cold boot attacks," in *2013 International Conference on Availability, Reliability and Security*, 2013, pp. 390–397.

[57] Johannes Bauer, Michael Gruhn, and Felix C Freiling, "Lest we forget: Cold-boot attacks on scrambled DDR3 memory," *Digital Investigation*, vol. 16, S65–S74, 2016.

[58] S. F. Yitbarek, M. T. Aga, R. Das, and T. Austin, "Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 313–324.

[59] Praveen Mosalikanti, Chris Mozak, and Nasser Kurd, "High performance DDR architecture in intelő core processors using 32nm cmos high-k metal-gate process," in *Proceedings of 2011 International Symposium on VLSI Design, Automation and Test*, IEEE, 2011, pp. 1–4.

[60] Olle Segerdahl and Pasi Saarinen, "An ice-cold boot to break bitlocker," 2018.

[61] "TCG Platform Reset Attack Mitigation Specification 1.0," Trusted Computing Group (TCG), Tech. Rep., 2018.

[62] "TCG Platform Reset Attack Mitigation Specification 1.10," Trusted Computing Group (TCG), Tech. Rep., 2019.

[63] Microsoft. "Secure MOR implementation." (2021), [Online]. Available: `https://web.archive.org/web/20230703085348/https://learn.microsoft.com/en-us/windows-hardware/drivers/bringup/device-guard-requirements` (visited on 2023-07-03).

[64] Tilo Müller, Felix C. Freiling, and Andreas Dewald, "TRESOR runs encryption securely outside RAM," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC11, San Francisco, CA: USENIX Association, 2011, p. 17.

[65] Tilo Müller, Benjamin Taubmann, and Felix C Freiling, "Trevisor: Os-independent software-based full disk encryption secure against main memory attacks," in *International Conference on Applied Cryptography and Network Security*, Springer, 2012, pp. 66–83.

[66] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo, and Kazuhiko Kato, "Bitvisor: A

thin hypervisor for enforcing I/O device security," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE 09, Washington, DC, USA: Association for Computing Machinery, 2009, pp. 121–130, ISBN: 9781605583754. DOI: 10.1145/1508293.1508311. [Online]. Available: https://doi.org/10.1145/1508293.1508311.

[67] Johannes Götzfried, Tilo Müller, Gabor Drescher, Stefan Nürnberger, and Michael Backes, "Ramcrypt: Kernel-based address space encryption for user-mode processes," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS 16, Xi'an, China: Association for Computing Machinery, 2016, pp. 919–924, ISBN: 9781450342339. DOI: 10.1145/2897845.2897924. [Online]. Available: https://doi.org/10.1145/2897845.2897924.

[68] J. Götzfried, N. Dörr, R. Palutke, and T. Müller, "Hypercrypt: Hypervisor-based encryption of kernel and user space," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, 2016, pp. 79–87.

[69] Lianying Zhao and Mohammad Mannan, "Hypnoguard: Protecting secrets across sleep-wake cycles," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS 16, Vienna, Austria: Association for Computing Machinery, 2016, pp. 945–957, ISBN: 9781450341394. DOI: 10.1145/2976749.2978372. [Online]. Available: https://doi.org/10.1145/2976749.2978372.

[70] Fabian Franzen, Manuel Andreas, and Manuel Huber, "Fridgelock: Preventing data theft on suspended linux with usable memory encryption," in *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '20, New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 215–219, ISBN: 9781450371070. DOI: 10.1145/3374664.3375747. [Online]. Available: https://doi.org/10.1145/3374664.3375747.

[71] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13, Tel-Aviv, Israel: Association for Computing Machinery, 2013, ISBN: 9781450321181. DOI: 10.1145/2487726.2488368. [Online]. Available: https://doi.org/10.1145/2487726.2488368.

[72] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas, "Intel software guard extensions (intel SGX) support for dynamic memory management inside an enclave," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, ser. HASP 2016, Seoul, Republic of Korea: Association for Computing Machinery, 2016, ISBN: 9781450347693. DOI: 10.1145/2948618.2954331. [Online]. Available: https://doi.org/10.1145/2948618.2954331.

[73] David Kaplan, Jeremy Powell, and Tom Woller, "AMD memory encryption," *White paper*, 2016. [Online]. Available: https://web.archive.org/web/20230821132119/https://www.amd.com/system/files/TechDocs/memory-encryption-white-paper.pdf.

[74] Intel Corporation, "Intel architecture memory encryption technologies specification - rev: 1.4," *White paper*, 2022. [Online]. Available: https://web.archive.org/web/20230321232159/https://cdrdv2-public.intel.com/679154/multi-key-total-memory-encryption-spec-1.4.pdf.

[75] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi, "A comparison study of intel SGX and amd memory encryption technology," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2018, pp. 1–8.

[76] Ponemon Institute, *Research report perceptions about self-encrypting drives: A study of it practitioners*, 2012.

[77] Gunnar Alendal, Christian Kison, *et al.*, "Got HW crypto? on the (in) security of a self-encrypting drive series," *Cryptology ePrint Archive*, 2015.

[78] C. Meijer and B. van Gastel, "Self-encrypting deception: Weaknesses in the encryption of solid state drives," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 72–87.

[79] Klaus Kursawe, Dries Schellekens, and Bart Preneel, "Analyzing trusted platform communication," in *ECRYPT Workshop, CRASH-CRyptographic Advances in Secure Hardware*, 2005, p. 8.

[80] Denis Andzakovic, "Extracting BitLocker keys from a TPM," *Pulse Security*, 2019. [Online]. Available: https://pulsesecurity.co.nz/articles/TPM-sniffing.

[81] Denis Andzakovic. "TPM specific LPC sniffer." (2019), [Online]. Available: https://github.com/denandz/lpc_sniffer_tpm (visited on 2023-08-09).

[82] Roland Pucher and Stepan Grebeniuk, “Angriff auf die bitlocker verschlüsselung,” *IT-SECX 2019*, 2019.

[83] Jesse D Kornblum, “Implementing BitLocker drive encryption for forensic analysis,” *Digital Investigation*, vol. 5, no. 3-4, pp. 75–84, 2009.

[84] Bernhard Kauer, “OSLO: Improving the security of trusted computing,” in *USENIX Security Symposium*, vol. 24, 2007, p. 173.

[85] Evan R Sparks, “A security assessment of trusted platform modules,” 2007.

[86] Johannes Winter and Kurt Dietrich, “A hijackers guide to communication interfaces of the trusted platform module,” *Computers & mathematics with applications*, vol. 65, no. 5, pp. 748–761, 2013.

[87] Jeremy Boone, “TPM Genie: Interposer attacks against the trusted platform module serial bus,” *NCC Group Whitepaper*, 2018.

[88] Cfir Cohen. “AMD-PSP: fTPM remote code execution via crafted EK certificate.” (2018), [Online]. Available: `https://seclists.org/fulldisclosure/2018/Jan/12` (visited on 2023-04-03).

[89] Hans Niklas Jacob, Christian Werling, Robert Buhren, and Jean-Pierre Seifert, “Faultpm: Exposing amd ftpms’ deepest secrets,” *arXiv e-prints*, arXiv–2304, 2023.

[90] Robert Buhren, Hans-Niklas Jacob, Thilo Krachenfels, and Jean-Pierre Seifert, “One glitch to rule them all: Fault injection attacks against amd’s secure encrypted virtualization,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2875–2889.

[91] Seunghun Han, Wook Shin, Jun-Hyeok Park, and HyoungChun Kim, “A bad dream: Subverting trusted platform module while you are sleeping,” in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, 2018-08, pp. 1229–1246, ISBN: 978-1-939133-04-5. [Online]. Available: `https://www.usenix.org/conference/usenixsecurity18/presentation/han`.

[92] Seunghun Han and Jun-Hyeok Park, “Bitleaker: Subverting bitlocker with one vulnerability,” *Blackhat Europe 2019*, 2019.

[93] Christopher Tarnovsky, “Semiconductor security awareness today and yesterday,” *Blackhat*, 2010.

[94] Christopher Tarnovsky, "Attacking TPM part two," *DEFCON*, vol. 20, 2012.

[95] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger, "TPM-FAIL: TPM meets Timing and Lattice Attacks," in *29th USENIX Security Symposium (USENIX Security 20)*, Boston, MA: USENIX Association, 2020-08. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi.

[96] Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential power analysis," in *Advances in CryptologyCRYPTO99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, Springer, 1999, pp. 388–397.

[97] Jeremy Scahill and Josh Begley, "The CIA Campaign to Steal Apples Secrets," *The Intercept*, 2015. [Online]. Available: https://theintercept.com/2015/03/10/ispy-cia-campaign-steal-apples-secrets/.

[98] Central Intelligence Agency (CIA). "TPM Vulnerabilities to Power Analysis and An Exposed Exploit to Bitlocker." (2010), [Online]. Available: https://web.archive.org/web/20230728160253/https://legacy.theintercept.com/document/2015/03/10/tpm-vulnerabilities-power-analysis-exposed-exploit-bitlocker/ (visited on 2023-07-28).

[99] "TCG EK Credential Profile For TPM Family 2.0; Level 0, Version 2.5, Revision 2," Trusted Computing Group (TCG), Tech. Rep., 2023. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/EK-Credential-Profile-For-TPM-Family-2.0-Level-0-V2.5-R1.0_28March2022.pdf.

[100] Microsoft. "Bitlocker countermeasures." (2023), [Online]. Available: https://web.archive.org/web/20230709212322/https://learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker/bitlocker-countermeasures (visited on 2023-07-09).

[101] Joanna Rutkowska, "Why do i miss microsoft bitlocker?" *The Invisible Things Labs Blog*, 2009. [Online]. Available: http://theinvisiblethings.blogspot.com/2009/01/why-do-i-miss-microsoft-bitlocker.html.

[102] F-Secure, *F-secure's guide to evil maid attacks*, 2018.

[103] Alexander Tereshkin, "Evil maid goes after PGP whole disk encryption," in *Proceedings of the 3rd International Conference on Security of Information and Networks*, 2010, pp. 2–2.

[104] Johannes Götzfried and Tilo Müller, "Analysing android's full disk encryption feature.," *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 5, no. 1, pp. 84–100, 2014.

[105] Joanna Rutkowska, "Anti evil maid," *The Invisible Things Lab*, 2011. [Online]. Available: http://theinvisiblethings.blogspot.com/2011/09/anti-evil-maid.html.

[106] Tilo Müller, Hans Spath, Richard Mäckl, and Felix C Freiling, "Stark - tamperproof authentication to resist keylogging," in *International Conference on Financial Cryptography and Data Security*, Springer, 2013, pp. 295–312.

[107] Matthew Garrett, "Anti evil maid 2 turbo edition," *mjg59's journal*, 2015.

[108] Andreas Fuchs. "Tpm2-totp." (2018), [Online]. Available: https://github.com/tpm2-software/tpm2-totp (visited on 2023-09-04).

[109] Jonathan M McCune, Adrian Perrig, Arvind Seshadri, and Leendert van Doorn, "Turtles all the way down: Research challenges in user-based attestation.," *HotSec*, 2007.

[110] Johannes Götzfried and Tilo Müller, "Mutual authentication and trust bootstrapping towards secure disk encryption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 2, pp. 1–23, 2014.

[111] Microsoft. "Secure the windows boot process." (2023), [Online]. Available: https://web.archive.org/web/20230709213856/https://learn.microsoft.com/en-us/windows/security/operating-system-security/system-security/secure-the-windows-10-boot-process (visited on 2023-07-09).

[112] Mickey Shkatov and Jesse Michael. "There's a Hole in the Boot." (2020), [Online]. Available: https://eclypsium.com/blog/theres-a-hole-in-the-boot/ (visited on 2023-09-03).

[113] Corey Kallenberg, Sam Cornwell, Xeno Kovah, and John Butterworth, "Setup for failure: Defeating secure boot," in *The Symposium on Security for Asia Network (SyScan)(April 2014)*, 2014.

[114] Xeno Kovah, Corey Kallenberg, Chris Weathers, Amy Herzog, Matthew Albin, and John Butterworth, "New results for timing-based attestation," in *2012 IEEE Symposium on Security and Privacy*, IEEE, 2012, pp. 239–253.

[115] Jacob Appelbaum, Judith Horchert, Ole Reissmann, Marcel Rosenbach, Jörg Schindler, and Christian Stöcker, "Interactive graphic: The NSA's spy catalog," *Der Spiegel*, 2013. [Online]. Available: https://web.archive.org/web/20131231034448/http://www.spiegel.de/international/world/a-941262.html.

[116] John V. Monaco, "Sok: Keylogging side channels," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 211–228. DOI: 10.1109/SP.2018.00026.

[117] Dmitri Asonov and Rakesh Agrawal, "Keyboard acoustic emanations," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, IEEE, 2004, pp. 3–11.

[118] Tzipora Halevi and Nitesh Saxena, "A closer look at keyboard acoustic emanations: Random passwords, typing styles and decoding techniques," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, 2012, pp. 89–90.

[119] Tong Zhu, Qiang Ma, Shanfeng Zhang, and Yunhao Liu, "Context-free attacks using keyboard acoustic emanations," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 453–464.

[120] Alberto Compagno, Mauro Conti, Daniele Lain, and Gene Tsudik, "Don't skype & type! acoustic eavesdropping in voice-over-ip," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 703–715.

[121] Joshua Harrison, Ehsan Toreini, and Maryam Mehrnezhad, "A practical deep learning-based acoustic side channel attack on keyboards," in *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2023, pp. 270–280.

[122] Fabian Mihailowitsch, "Detecting hardware keyloggers," *HITB SecConf, Kuala Lumpur, Malaysia (October 2010)*, 2010. [Online]. Available: https://archive.conference.hitb.org/hitbsecconf2010kul/materials/D1T1%20-%20Fabian%20Mihailowitsch%20-%20Detecting%20Hardware%20Keyloggers.pdf.

[123] Ryan M Gerdes and Saptarshi Mallick, "Physical-layer detection of hardware keyloggers," in *Research in Attacks, Intrusions, and Defenses: 18th International Symposium, RAID 2015, Kyoto, Japan, November 2–4, 2015. Proceedings 18*, Springer, 2015, pp. 26–47.

[124] Jacob Appelbaum, Judith Horchert, Ole Reissmann, Marcel Rosenbach, Jörg Schindler, and Christian Stöcker, "NSAs secret toolbox: Unit offers spy gadgets for every need," *Der Spiegel*, 2013. [Online]. Available: https://web.archive.org/web/20131230221032/

https://www.spiegel.de/international/world/nsa-secret-toolbox-ant-unit-offers-spy-gadgets-for-every-need-a-941006.html.

[125] Jacob Appelbaum, Judith Horchert, and Christian Stöcker, "Shopping for spy gear: Catalog advertises NSA toolbox," *Der Spiegel*, 2013. [Online]. Available: https://web.archive.org/web/20131230224749/http://www.spiegel.de/international/world/catalog-reveals-nsa-has-back-doors-for-numerous-devices-a-940994.html.

[126] "NSA Playset." (2022), [Online]. Available: https://web.archive.org/web/20221206035742/http://www.nsaplayset.org/ (visited on 2022-12-06).

[127] Ben Blaxill and Joel Sandin, *Picodma: Mda attacks at your finger-tips*, 2019. [Online]. Available: https://i.blackhat.com/USA-19/Wednesday/us-19-Sandin-PicoDMA-DMA-Attacks-At-Your-Fingertips.pdf.

[128] Andy Davis, "To dock or not to dock, that is the question: Using laptop docking stations as hardware-based attack platforms," *NCC Group*, 2013.

[129] Roger G Johnston, "Tamper-indicating seals," *American Scientist*, vol. 94, no. 6, p. 515, 2006.

[130] Roger G Johnston, "Tamper-indicating seals: Practices, problems, and standards," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2003.

[131] Datagram, "Introduction to tamper evident devices," *DEFCON*, vol. 19, 2016.

[132] Eric Michaud and Ryan Lackey, "Thwarting evil maid attacks: Physically unclonable functions for hardware tamper detection," *Chaos Communication Congress*, vol. 30, 2013.

[133] Yaroslav Pronin. "Blink Comparison." (2022), [Online]. Available: https://github.com/proninyaroslav/blink-comparison (visited on 2023-08-12).

[134] Hewlett-Packard. "HP TamperLock." (2020), [Online]. Available: https://web.archive.org/web/20230814113343/https://h20195.www2.hp.com/v2/getpdf.aspx/4AA7-8167ENW.pdf (visited on 2023-08-14).

[135] Lenovo Product Security Office. "Lenovo BIOS Security Features for ThinkPad X250." (2015), [Online]. Available: https://web.archive.org/web/20230814114112/https://download.lenovo.com/pccbbs/mobiles_pdf/lenovo_thinkpad_x250_bios_security_white_paper.pdf (visited on 2023-08-14).

[136] Freedom of the Press Foundation and Guardian Project. “Haven: Keep Watch.” (2017), [Online]. Available: https://guardianproject.github.io/haven/ (visited on 2023-08-14).

[137] Scott Culp, “The ten immutable laws of security,” *Microsoft Security*, 2000.

[138] Advanced Micro Devices, “AMD memory guard,” *White paper*, 2020. [Online]. Available: https://www.amd.com/system/files/documents/amd-memory-guard-white-paper.pdf.

[139] Advanced Micro Devices. “AMD Infinity Guard.” (2021), [Online]. Available: https://www.amd.com/en/technologies/infinity-guard (visited on 2023-07-29).

[140] Intel Corporation, “Intel Hardware Shield - Intel Total Memory Encryption,” *White paper*, 2021. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/white-paper-intel-tme.pdf.

[141] Nuvoton Technology Corporation. “TPM Endorsement Key (EK) Certificate Chain.” (2021), [Online]. Available: https://www.nuvoton.com/export/sites/nuvoton/files/security/Nuvoton_TPM_EK_Certificate_Chain.pdf (visited on 2023-08-20).

[142] “UEFI Defensive Practices Guidance,” National Security Agency, Tech. Rep., 2017.

[143] The Linux Userspace API (UAPI) Group. “Unified Kernel Image (UKI).” (2022), [Online]. Available: https://uapi-group.org/specifications/specs/unified_kernel_image/ (visited on 2023-08-31).

[144] Morten Linderud. “sbctl - Secure Boot key manager.” (2020), [Online]. Available: https://github.com/Foxboron/sbctl (visited on 2023-08-31).

[145] Jiewen Yao and Vincent Zimmer, “Building secure firmware,” *Apress: New York, NY, USA*, 2020.

[146] Jos Wetzels. “TPM Sniffing Attacks Against Non-Bitlocker Targets.” (2022), [Online]. Available: https://www.secura.com/blog/tpm-sniffing-attacks-against-non-bitlocker-targets (visited on 2023-08-20).