MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Hardware-encrypted disks in Linux

Master's Thesis

ŠTĚPÁN HORÁČEK

Brno, Fall 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Hardware-encrypted disks in Linux

Master's Thesis

ŠTĚPÁN HORÁČEK

Advisor: Ing. Milan Brož, Ph.D.

Department of Computer Systems and Communications

Brno, Fall 2022


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Štěpán Horáček

Advisor: Ing. Milan Brož, Ph.D.

Abstract

The thesis aims to analyze existing approaches to using hardware-encrypted block devices (disks) in Linux.

The implementation part should provide basic low-level tools for tools configuration and status check of such devices. Tools should use generic Linux kernel interfaces (like ioctl calls).

Student should

- get familiar with and study available resources for self-encrypted drives, OPAL2 standard, block layer inline encryption,
- analyze and describe security of such drives,
- provide state-of-the-art overview of existing attacks,
- implement proof-of-concept low-level tools to access available devices,
- evaluate and discuss the result.

The student should be familiar with C code for low-level system programming and cryptography concepts.

Keywords

keyword1, keyword2, ...

Contents

1 Introduction

Whether it is a disk of a corporate laptop, personal desktop, or server storage, all of these devices may very likely contain sensitive information such as company internal data, medical records, or even session cookies.

According to Verizon 2022 Data Breach Investigation Report [1], lost or stolen assets made up almost 5% of the analysed security incidents (not necessarily data breaches due to the way it's hard to find out if there was one with lost or stolen assets). Furthermore, this percentage does not account for other possible attacks, where the disk could be compromised without stealing the device, something like infiltrating the server room and just reading the data.

As such, it can be seen that some form of protection of confidentiality of data on the disk is necessary. Disk encryption may seem like an obvious solution. However, the traditional software disk encryption ... (However, something about how the performance of software solutions might not be the best, or that someone can see hardware having better security... decide whether not to just mention that software disk encryption has disadvantages, and then talk about it later as advantage of hardware encryption in the chapter later on) But something how about compared to software solutions, the hardware ones are closed source, without any info available, and so easily vulnerable to bad implementations. Something about how it might introduce new attacks.

In this thesis, we will introduce the idea of hardware disk encryption, basic terms used in this area and something, and categorization of different approaches to hardware disk encryption. Afterwards, we will focus on the Opal standard as a representative of the self-encrypting disks approach to hardware disk encryption. Next, we will look at possible attacks on devices using hardware disk encryption. Finally, we will introduce our own tool to manage Opal disks and get their properties, and analyse data acquired using this tool.

2 Hardware disk encryption

Hardware disk encryption is a technology that provides confidentiality of data stored on a storage device using encryption provided by hardware.

Generally, disk encryption, is done by encrypting the saved data using a data encryption key (DEK), also sometimes known as the media encryption key. In order to allow changing of passwords and having multiple passwords, the DEK is not cryptographically bound to any particular password and instead is saved in an encrypted form. It is encrypted using a key encryption key (KEK), which is generated from a password and therefore is cryptographically bound to a specific password.

The data encryption process can be conducted in logically and physically different places, depending on the type of disk encryption. In the following sections of this chapter, we will further describe three such types.

2.1 Self-encrypting drives

Self-encrypting drive (SED) is a disk drive with a built-in disk encryption. Having a built-in disk encryption enables the disk to

Self-encrypting drives carry over other disk encryption approaches. Some of those advantages are something like no need for re-encryption of all the data to activate encryption or no need for modification of software on the system.

2.1.1 ATA Security Feature Set

ATA Security Feature Set [2] allows one to protect the data and configuration of the disk. It offers commands to set a password, unlock the disk, erase the data on the disk, freeze the device, and disable the password.

In order to authenticate to the disk, there are two passwords. The Master password, initially set by the manufacturer, and the User password. The usage of the Master password to unlock or disable the security can be disabled by setting the Master Password Capability

bit while setting the User password. Even if the Master password is disabled for unlocking the disk or disabling the locking, it can still be used for erasing the disk.

The disk protection is enabled by setting the User password. After the device is turned on again, the device is locked and to access the data on the disk or use some of the ATA commands, it needs to be unlocked by providing a password using the unlock command. By disabling the User password, the protection of the device can be disabled.

However, the ATA Security Feature Set itself does not provide encryption of the data saved on the disk, only access control. But, some of the SED solutions use ATA Security commands as their interface to provide access to their own implementation of the disk encryption and disk unlocking.

2.1.2 Proprietary standards

There also exist proprietary alternatives to the previously introduced standards. These are often managed either by software downloaded from the site of the manufacturer or by using software from a special partition on the disk. An example of such are disks from Western Digital's My Passport series.

TODO: Fact check this. Maybe just integrate the subsection into something else.

2.2 Inline encryption hardware

Compared to the previously described self-encrypting drives, inline encryption hardware is separated and independent from the disk. Such devices usually do not have any access control or authentication capabilities. Instead, they allow to simply insert a key into keyslot, select a keyslot, and remove a key from a keyslot. The data passing through the inline encryption device to the disk is then encrypted using the key from the selected keyslot. Similarly, data passing from the disk are decrypted by the inline encryption hardware.

TODO: something about how it provides actually a way to check the encrypted content, right? Seems to be primarily in android , source? Find how much inline enc hardware there really is

2.3 Software encryption

As an alternative to hardware encryption, there exists software encryption. Compared to hardware encryption, this encryption does not require specialized hardware. Even though specialized hardware is not required, performance is often improved using cryptography coprocessors.

probably just a quick overview, ... maybe change the chapter to just "disk encryption"... or mention this just shortly at the start...

2.3.1 Linux stack

In the Linux kernel, there are several available software encryption libraries and subsystems. In this section, we will mention only two of them, each representing a different approach to software encryption.

TODO: rethink where to put this... maybe tools?

dm-crypt The subsystem *dm-crypt* works on the block device level. It can encrypt everything, including metadata or partition tables, but it does not offer precise control over what is encrypted, and simply entire filesystem is encrypted. It also needs access to the block device.

TODO: I don't think that inline encryption is supported AFAIK, does not require any extra implementations on the side of drivers or file system or anything, right?

fscrypt The library *fscrypt* works on filesystem level. It can limit the encryption to only some files. Inline encryption is supported.

TODO: requires the filesystem to actually use the library In the following paragraphs, that should be redone, I am assuming ext4.

fscrypt - How to make it work? As of now, blk-layer inline encryption is supported only by two filesystems in Linux: ext4 and F2FS. Need to use a kernel with `CONFIG_FS_ENCRYPTION_INLINE_CRYPT` enabled. Need to first mount the filesystem with the inline encryption flag:

```
mount -t ext4 /dev/foo /mnt/foo -o inlinecrypt
```

It is not enough to specify the inline encryption flag, the encryption itself also must be enabled. Assuming ext4 filesystem, the encryption can be enabled after mounting like so:

```
tune2fs -O encrypt "/dev/loop0"
```

After this, `fscrypt` can be used as normal and it will use inline encryption for this filesystem.

In order to encrypt a folder using a `fscrypt`, the following must be done: an encryption key must be added and the encryption policy must be created.

```
int fd = open(pathname, O_RDONLY | O_CLOEXEC);

struct fscrypt_add_key_arg *key_request = calloc
   (1, sizeof(struct fscrypt_add_key_arg) +
   key_len);
struct fscrypt_policy_v2 policy_request = { 0 };

// add a key
key_request->key_spec.type =
   FSCRYPT_KEY_SPEC_TYPE_IDENTIFIER;
key_request->key_id = 0;
key_request->raw_size = key_len;
memcpy(key_request->raw, key, key_len);
ioctl(fd, FS_IOC_ADD_ENCRYPTION_KEY, key_request
   );

// set a policy
policy_request.version = 2;
policy_request.contents_encryption_mode =
   FSCRYPT_MODE_AES_256_XTS;
policy_request.filenames_encryption_mode =
   FSCRYPT_MODE_AES_256_CTS;
policy_request.flags =
   FSCRYPT_POLICY_FLAGS_PAD_8 |
   FSCRYPT_POLICY_FLAGS_PAD_16 |
   FSCRYPT_POLICY_FLAGS_PAD_32;
```

```
memcpy(policy_request.master_key_identifier,
   key_request->key_spec.u.identifier,
   FSCRYPT_KEY_IDENTIFIER_SIZE);
ioctl(fd, FS_IOC_SET_ENCRYPTION_POLICY, &
   policy_request);
```

This code sets up an encryption policy for the file specified by the pathname.

fscrypt - How does it work? - Setup During mounting the "inlinecrypt" `/SB_INLINECRYPT` flag is written into the `super_block` structure.

It all starts in `__ext4_new_inode`. This is the internal function used when creating new inodes, called by functions such as `ext4_create` when creating a new file.

The function (if it is not inode used for large extended attributes?) calls `fscrypt_prepare_new_inode`.

`fscrypt_prepare_new_inode -> fscrypt_setup_encryption_info -> setup_file_encryption_key -> fscrypt_select_encryption_impl`

In `fscrypt_select_encryption_impl` there is actually the only place where the `SB_INLINECRYPT` flag is used. ... Calls `blk_crypto_config_supported` to check the device's crypto profile. Afterwards, `fscrypt_select_encryption_impl` function sets the `(fscrypt_info *)ci->ci_inlinecrypt`.

`setup_per_mode_enc_key` then sets the `(fscrypt_info *)ci->ci_enc_key`.

fscrypt - How does it work? - Usage The bio function is stored in `(struct bio *)bio->(struct bio_crypt_ctx *)bi_crypt_context`

Function `fscrypt_set_bio_crypt_ctx` changes the file's bio to use inline encryption... simply calls the blk layer `bio_crypt_set_ctx`.

Calling `submit_bio` like normally ... `__submit_bio` calls `__blk_crypto_bio_prep`...

"If the bio crypt context provided for the bio is supported by the underlying device's inline encryption hardware, do nothing."

`__blk_crypto_rq_bio_prep` however sets the context of the request to the one of the bio... After the bio prep `blk_mq_submit_bio` gets called (which calls `blk_mq_bio_to_request`, and after that also `blk_crypto_init_request->blk_crypto_get_keyslot` which updates the devices keyslot to contain the new key..., but does nothing if the device does not have keyslots)..... the info about the key to use then has to be acquired by the driver from `request->`

Most important are probably structures `blk_crypto_ll_ops` and `blk_crypto_profile`... just two operations, program key and evict key.

how to get crypto profile from outside..

Where does the hardware come to play?

`ufshcd_exec_raw_upiu_cmd()->ufshcd_issue_devman_upiu_cmd()->ufshcd_prepare` sets the header with the correct keyslot.

3 TCG Opal 2.0

TCG Opal Security Subsystem Class (SSC) 2.0 (hereinafter referred to simply as "Opal standard") is a specification for storage devices, aiming to provide confidentiality of stored data while the conforming disk is powered off [3]. It is one of the representatives of the self-encrypting drive approach to hardware disk encryption. Information for this chapter comes primarily from the Core standard [4] and the Opal standard [3].

In the following sections we shall firstly describe the specification and it's features and capabilities as described by the Core and Opal standards, afterwards we will focus on the mandatory features and requirements on Opal devices, and lastly we will describe the interface available on a Linux host.

3.1 Structure of the standard

The Opal standard is defined as a subsystem extending the TCG Storage Architecture Core Specification (hereinafter referred to simply as "Core standard"). The Core standard [4] specifies the core features and properties shared among several different types of storage security subsystems, that extend the core functionality by specifying additional features or define the set of mandatory features. Each of these subsystems is focused on a different use case. These subsystems are namely:

- Opal — targeted at a corporate and personal usage. Described more closely in the rest of this chapter.
- Opalite — simplified Opal. Does not mandate features such as locking ranges, decreases the minimal number of admin and user authorities, or additional DataStore tables [5].
- Pyrite — encryption-less Opalite. Similar to Opalite, however it does not mandate encryption of data saved on disk, and instead may offer only logical access control [6].
- Ruby — focused on data centers and server drives. Offers only global range encryption, weaker configuration of access control,

no pre-boot authentication support [7]. Replacing the older Enterprise subsystem.

Other than the Core and Opal standards defining the fundamentals, there are defined also Feature Sets. These Feature Sets expand the standards with less fundamental features, such as Additional DataStore [**TODO**], Block SID Authentication [**TODO**], PSK Secure Messaging [**TODO**], PSID [**TODO**], or Single User Mode [**TODO**]. However, we will focus primarily on those that are mandatory in the Opal specification.

3.2 Architecture

The Core standard defines several parts of the trusted device.

3.2.1 Trusted Peripheral

Trusted peripheral (TPer) is a device located on the disk that provides the security of the data on disk. A TPer consists of multiple Security Providers.

3.2.2 Security Provider

Security Provider (SP) is defined as a set of tables, methods and an access control. Each SP is derived from a set of templates. These templates define a set of the tables and methods, aimed at one functionality, subset of which the SP implements. The templates are described more closely in later chapter 1. The Opal 2.0 standard defines that at least the Admin SP and Locking SP must be present in the TPer. The Admin SP tasked with administrating the TPer and other SPs, which may include creating new SPs, deleting existing SPs, or providing information about SPs. The Locking SP provides access to functionality such as managing locking ranges, locking the drive, or managing access control. Both of these SPs are described more closely in later chapter 1.

3.3 Capability discovery

In order to find out the properties and features of a particular device, there exists the so called Discovery process. This process is divided into three levels, each with different reported information and a different approach to access the information.

3.3.1 Level 0 Discovery

Level 0 Discovery provides basic information about the secure device, and is performed using only the IF-RECV and IF-SEND commands of the device. The information about the device is provided through feature descriptors. The presence of a feature descriptor header means that the feature is supported and the fields of the header describe the basic properties of that feature.

Some of the features described by these descriptors are the TPer Feature Descriptor (supported communication features such as ACK/-NACK support, ComID management, buffer management, async communication, ...), Locking Feature Descriptor (whether disk is locked, etc.), Geometry Feature Descriptor (parameters of the disk, such as block size), Opal V1.0 Feature Descriptor, SingleUser Feature Descriptor (...), DataStore Feature Descriptor (size of the table), Opal 2.0 Feature Descriptor (base ComID, number of ComIDs, default pin, number of locking users/admins).

3.3.2 Level 1 Discovery

Level 1 Discovery provides capabilities of the communication channel, and is performed using the `Properties` control session method. The Level 1 Discovery is used not only as a way for the host to find out the capabilities of the TPer, but to determine shared limits depending on the capabilities of both the TPer and the host, as host can also communicate its capabilities to the TPer.

The reported properties mandatory for Opal are:

- `MaxMethods` — maximum number of methods per received SubPacket.

- `MaxSubpackets` — maximum number of SubPacket per received Packet.
- `MaxPacketSize` — maximum size of a received Packet.
- `MaxPackets` — maximum number of Packets per received ComPacket.
- `MaxComPacketSize` — maximum size of a received ComPacket.
- `MaxResponseComPacketSize` — maximum size of sent ComPacket.
- `MaxSessions` — maximum number of active sessions.
- `MaxIndTokenSize` — maximum size of an individual token.
- `MaxAuthentications` — maximum possible authenticated authorities.
- `MaxTransactionLimit` — maximum number of active transactions.
- `DefSessionTimeout` — default length of session timeout in miliseconds.

There are also defined properties which are not required in Opal:

- `MaxReadSessions` — maximum number of reader sessions.
- `MaxAggTokenSize` — maximum size of a combined token.
- `MaxSessionTimeout` — maximum possible session timeout.
- `MinSessionTimeout` — minimum possible session timeout.
- `DefTransTimeout` — default transaction timeout.
- `MaxTransTimeout` — maximum possible transaction timeout.
- `MinTransTimeout` — minimum possible transaction timeout.
- `MaxComIDTime` — timeout for ComID.

- `ContinuedTokens` — support of continued tokens (token with size unknown at start, joins several tokens together).
- `SequenceNumbers` — support of packet numbering (in Packet).
- `AckNak` — support of ACK/NAK (in Packet) in communication.
- `Asynchronous` — support of asynchronous communication.

Other than the defined properties, the level 1 Discovery process may also report other, vendor specific, properties.

3.3.3 Level 2 Discovery

Level 2 Discovery is the act of reading any table and is provided by the `Get` method of an SP. This includes reading any table, such as the access control table or table of locking ranges. Some of the SPs' tables are described in later chapter 1.

3.4 Life cycle

In order to keep information about the state and the working capacity of an SP, the concept life cycle is introduced. Life cycle describes the condition of an SP using one of several states. In the Core standard the following states are introduced:

- "nonexistent" — the SP does not exist. The SP might have been not created or already deleted.
- "issued" — the SP is in functional state.
- "issued-disabled" — the SP can only be authenticated to and enabled, all other functionality is disabled. An SP can be enabled and disabled using the SPInfo table of the corresponding SP.
- "issued-frozen" — no functionality of the SP is enabled. An SP can be frozen and unfrozen using the SP table of the Admin SP.
- "issued-disabled-frozen" — the SP is both disabled and frozen as described in the two previous items.
- "issued-failed" — SP is in fatal failure from which it cannot recover.

3.4.1 SP Issuance

SP issuance is the process of creation of a new SP from the Base Template and a set of other templates. This is achieved by calling the Admin SP's method IssueSP. After an SP is issued, it can be personalized. Personalization of an SP is the process of creating new tables (using method CreateTable), filling in existing tables (using methods CreateRow and Set), and setting up access control (using method AddACE).

3.4.2 Life cycles in Opal

Opal extends the states defined in the Core specification with a new set of life cycle states called "manufactured" — "manufactured-inactive", "manufactured", "manufactured-failed", "manufactured-disabled", "manufactured-disabled-frozen", "manufactured-disabled". Each of these new states mirrors the corresponding "issued" state from the Core specification. Compared to the "issued" states, the "manufactured" states are created during manufacturing by the manufacturer, instead of during the subsequent use by the TPer owner. This is reflected by the fact that these manufactured SPs are not issued and deleted in Opal, instead they are activated and reverted. Activation automatizes the personalization using hardcoded, preconfigured values. Reverting SP maintains it's existence, returning it to the factory state instead. Note that this means that the "manufactured-inactive" corresponds to the "nonexistent" state.

3.4.3 Taking ownership

In order to take ownership of a TPer, the basic steps are: 1. initialize a session with the Admin SP as the Anybody authority, 2. read the value in the `C_PIN_MSID` row of `C_PIN` table. This row has a static UID. 3. finish session 4. initialize a session with the Admin SP as the SID authority, using the PIN from `C_PIN_MSID`. 5. change the PIN 6. finish session

however, note that the entire process of taking ownership is entirely optional and the device will work fine if it is skipped.

3.5 Communication

In order to send commands to the TPer or the SPs and receive responses, a specific protocol must be used. The communication protocol is split into several layers. Starting from the lowest layer described in this thesis is the interface layer. The interface layer corresponds to the communication with the control unit of the disk. Depending on the disk interface there can be different security commands used and the Core standard abstracts these commands into only two commands: IF-RECV and IF-SEND. These commands facilitate all the communication necessary to communicate with the higher layers and are described more closely in section 1. Following layer is the TPer layer, a layer used primarily for allocation of ComID. Since on this layer the ComIDs are not used yet and so the state of the communication cannot be kept, the communication is only one way, each "session" consisting only of one command. The Communication layer supports ComIDs and therefore allows two-way communication, and is used primarily for further ComID management, such as getting the state of ComID, or resetting state of ComID. The Management layer facilitates establishing of session between an SP and the host. Since session number is not issued yet, this layer uses Control Sessions with a static session number. The final layer is the Session layer. At this layer the session is already established and a communication can be performed using methods sent in packets. Most of the communication occurs in this layer with the use of method invocations.

3.5.1 ComID

One of the information required in order to send a command is the ComID, a number identifying the caller (e.g. application of the host). It ensures that responses to method calls will be received by the correct application, since there can be at one time multiple callers. There are three types of ComIDs: statically allocated ComID, dynamically allocated ComID and special ComIDs. Statically allocated ComID can be acquired through the level 0 Discovery process. The Discovery process simply informs about the base ComID number and the number of of static ComIDs. Dynamically allocated ComID can be acquired

using the GET_COMID command. There exists also special ComIDs that are used to invoke the Level 0 Discovery or for ComID management.

Every ComID is in one of few states. These states are Inactive, Issued and Associated. If a ComID is in Issued or Associated state it is considered Active. After GET_COMID a ComID becomes Issued and if a session is opened under it, it becomes Associated. Associated state becomes Inactive again once all sessions opened under it have been closed.

In Opal, statically allocated ComIDs are always Active.

3.5.2 Packetization

The communication in the last two layers is based on packets. There are three packet types that are nested, each packet type providing different service. ComPacket is the outermost one, each ComPacket contains data for communication under only a single ComID. Each ComPacket can contain several Packets. The usage of Packet is intended for session reliability, specifying sequential numbers and the acknowledgements for them. Every Packet then consists of one or more SubPackets. Finally, SubPacket contains one or more method calls, or results of method calls. In case that the TPer supports buffer management, it may also use SubPackets for credit control, to inform the other party about remaining buffer amount.

3.5.3 Methods

The commands to TPer and SPs and their respective responses are transported in the form of methods. Methods are a sequence of tokens ...

The data of the SubPacket carry either the method call or method response. These are expressed using a set of tokens in a specific order.

Each method invocation contains invoking UID, method UID, parameters, and status code list. Invoking UID is used to identify which structure the method is invoked on. Other than tables and objects, there are also special invoking UIDs. These special invoking UIDs are "thisSP", signifying the SP with which the current session is open, and "SMUID", signifying the session manager. Method UID specifies the method to be called. Methods for "SMUID", managing session, cannot

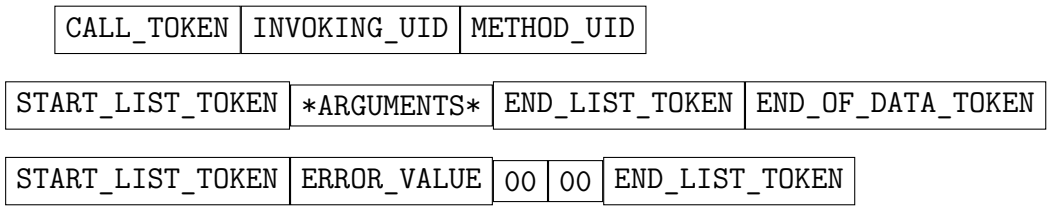

Figure 3.1: Caption

be found in the MethodID table. There are two kinds of parameter: mandatory, identified by their position, and optional, used in a structure using an integer to identify the parameter, followed by the data of the parameter.

The method has the following structure:

```
 CALL_TOKEN | INVOKING_UID | METHOD_UID
| START_LIST_TOKEN | *ARGUMENTS* | END_LIST_TOKEN | END_OF_DATA_TOKEN
| START_LIST_TOKEN | ERROR_VALUE | END_LIST_TOKEN |
```

Both the invocation of the method and the respond use the same general structure. Response method uses parameters to return data. There are two types of arguments: mandatory arguments that all have to occur in the specified order, and optional arguments which can be left out and each of them is formatted as follows:

```
 START_NAME_TOKEN | ARGUMENT_ID | ARGUMENT_VALUE | END_NAME_TOKEN .
```

3.5.4 Sessions

Whereas ComID is used to differentiate the senders and, therefore, also the recipients of the methods, sessions are used for parallelization of communication of one such actor. Each session can have different authorized authorities and methods in process, but each session is still bound to one ComID. Since each ComID corresponds to one user, both the user and the TPer can send multiple methods calls or responses in one ComPacket.

TODO: not user or actor but host... e.g. if it would be network storage, there can be some thingie that would split the communication

Sessions use a system of readers-writer locks to enable several concurrently running read sessions without causing concurrency issues.

There are two types of sessions: regular session and control session (not going to care about control much, just between the TPer session

manager and host session manager)... Each ComID has one associated control session with a lifespan the same as the ComID. On the other hand, there can be multiple regular sessions.

Since the control session is used to establish a regular session with an SP, it is not connected to any SP. The control session is used for managing the regular sessions and, as such, provides methods such as:

- The `properties` method enables to find a compromise between the communication capabilities of the host and the TPer or simply find the communication capabilities of the TPer. Using this method, the properties defined earlier in section 1, such as maximum packet size, are established.
- The StartSession and SyncSession methods provide a way to start an unencrypted session.
- The StartTrustedSession and SyncTrustedSession methods provide a way to complete challenge-response authentication or to setup secure messaging/key exchange.
- The CloseSession method used to close the session.

Each is identified by it's session number. The session number consists of the TPer session number selected by the TPer and the host session number selected by the host.

Secure messaging

Depending on the authorities used, it is also possible to start a secure messaging session, which provides message authenticity and/or confidentiality. Authority's operation type decides the what trusted messaging is to be used. These are further described in the section 1. With other cases there is no secure messaging.

3.5.5 Transactions

In order to facilitate safe execution of sequences of methods, the standard specifies transactions. Similarly to transactions in database systems, this feature enables one to revert effects of sequence of methods.

This is done automatically in case the transaction is not finished, or if the transaction is manually aborted. However, not all the effects of methods are rolled back, such as logs. Nested transactions are supported, in which case the transaction is committed when the outermost transaction is finished.

3.5.6 Memory structures

In order to maintain the state of the device, tables are exclusively used. Tables are defined by their columns and each row is called an object. Each SP contains a meta table called Table table which contains information about all the other tables in the SP.

Each template also defines its tables such as the key tables containing keys for different LRs.

3.6 Access Control

In order to provide access to methods only to authorized actors, the standard also defines access control. The access control provides a way to allow access to methods of the SP only after the knowledge of a secret has been proved. Information required to verify knowledge of the secret is called a credential and is stored in one of the corresponding SP's credentials tables. Each SP may contain several credential tables, one for each type of credentials, such as a password, an RSA key pair, or an AES key.

In order to support authentication of more than one user or require multiple users at once, access control rules are defined using Access Control Lists (ACL) containing Access Control Elements (ACE). Each entry in an ACL has an owner .... ACEs are Boolean expressions with inputs being authentication of authorities.

Each combination of method and object has defined ACL, which needs to be satisfied in order for the method to be invoked on the object. An ACL is satisfied once one of the ACEs contained within it is satisfied, where authenticated authority variables evaluate as true.

Other than explicit authentication, implicit authentication is also possible. Implicit authentication may be used with, e.g. the Exchange

operation type where the session key by the SP is encrypted using a pre-shared key and returned to the host.

Authority may be authenticated either during session startup using parameters of the SessionStart method, or, if there is more than one authority, the additional authorities may be authenticated using Authenticate method. The Authenticate method authenticates only using explicit methods. In case Authenticate method is used to authenticate using a password, one invocation of the method with a password as the parameter is enough. In the case of challenge and response operation type (Sign, SymK, HMAC), the first invocation only specifies the authority to be authenticated and receives challenge in response, and the second invocation provides the response.

Each authority is an object in the authority table of the corresponding SP. Each authority has a type which is either individual or class. Class authorities correspond to a set of authorities so that they can be easily assigned and changed in bulk.

Other than the Admin and User authorities, there is also a special SID authority. This authority represents the owner of the TPer, and, as such, provides access of admins extended by the capabilities related to TPer management, such as the ability to activate an SP or enabling Admin authorities. Compared to the other authorities, SID authority is shared between all SPs. There is also the Anybody authority that provides access to public information on the TPer. No authentication is needed for this authority.

Other than PIN authentication, the Core standard also defines other approaches to authentication.

3.6.1 Authorities

Authorities are constructs used to represent an actor. Authorities are saved in the Authority table of every SP, provided by the Base template, and so are separate for each of the SP. Each authority is associated with one credential. This credential is an object from a table, depending on the type of the credential.

The Base template defines several optional authorities such as the Admin authorities, representing the owner of the SP, the Makers authorities, representing the manufacturer of the TPer, the Security Identifier authority (SID), representing the owner of the TPer, and

TPer authorities, representing the TPer itself. The Base template also defines a special authority Anybody without any credentials. This authority can be used in reading public info of the SPs or to take ownership of the TPer.

Each authority has assigned one operation type. This operation type determines how can this authority be used to authenticate.

- None —
- Password — also called PIN, authenticated using a password as a parameter.
- Sign — authenticated using challenge and response with asymmetric crypto.
- Exchange — credentials of this authority are used to encrypt session key. Can be used only in StartSession.
- SymK — authenticated using challenge and response with symmetric crypto.
- HMAC — challenge and response with HMAC. E.g. during session startup the HMAC is sent in a parameter and contains the HMAC of the method excluding the HMAC parameter.
- TPerSign — used to authenticate the TPer.
- TPerExchange —

3.7 Templates

Since SPs may share some of their functionality such as authentication, modification of tables, or retrieval of data from tables, there exists templates to define this shared functionality. Each of the SP then implements a subset of functionality of one or more templates. Since templates

3.7.1 Base template

The Base template defines the shared subset required by every SP, and is therefore mandatory. The most important functionality that is defined by this template is access control and metadata.

3.7.2 Admin template

The Admin template is a template specific for the unique Admin SP. It provides access to methods managing the TPer.

3.7.3 Crypto template

Template providing methods providing cryptographic methods.

3.7.4 Locking template

Template specific for the unique Locking SP. Provides access to methods managing the lock state of the device, locking ranges...

3.7.5 Clock template

Template providing methods for indicating current time, measuring lag, and access to monotonic counter.

3.7.6 Log template

Template providing methods for logging activity. The logging can be either carried out manually by an user (or an user application) or automatically by the TPer (as a result of invocations of methods of SPs containing this template, including invocations during read-only sessions). Only one system log table may exist on each SP, but multiple user log tables may be present on one SP. The logs are saved on non-volatile storage, depending on the security setting of the table, they may be either buffered or be saved after each entry.

3.8 Admin SP

The first of the Manufactured SPs is the Admin SP. This SP holds information about the TPer and all the present SPs, it allows creation, deletion and general modification of the SPs. It is ses base and admin template

unique

This SP is initialized as "manufactured", so that it can function as a starting point of TPer management.

3.8.1 Methods

Even though the templates from which the Admin SP consists define many more methods, the Opal standard mandates only the following ones:

- “Next” — returns UIDs of objects in the table. Allows to specify the returned amount and the first UID.
- “GetACL” — returns the ACL of the method and object or method and table combination.
- “Get” — returns the value of the object or the contents of the byte table.
- “Set” — sets the value of the object or the contents of the byte table.
- “Authenticate” — performs an additional explicit authentication.
- “Revert” — reverts the SP referred to by the invoking ID to its initial state.
- “Activate” — activates the SP referred to by the invoking ID.
- “Random” — returns specified amount of random bytes.

3.9 Locking SP

Locking SP procures the disk encryption and the locking and unlocking associated with it. This means that it provides access to manipulation of locking ranges, key generation and ...

also unique
uses base and locking template

3.9.1 Methods

Some of the methods Opal mandates for Locking SP are shared ("Next", "GetACL", "Get", "Set", "Authenticate", "Random"), but there are also two new ones:

- "GenKey" — regenerates a key or a credential. But, in the case of Opal, this method is mandated to be able to be used at least on the `K_AES_*` DEK keys.
- "RevertSP" — reverts the Locking SP. Compared to "Revert" this method can revert only the SP to which the session is connected, allowing reverting without having access to Admin SP's Admin authority.

3.9.2 Locking range

The locking range feature gives the user a way to specify an LBA range on the disk that can be locked independently on the rest of the disk. Each locking range also has its own ACE to control who can lock and unlock the range. Because each LR is also encrypted using its own DEK, it is possible to regenerate new DEK for a single LR, discarding data of only one LR.

There also always exists a special range called the global locking range that covers any area on the disk that is not already covered by a regular locking range.

In order to control the access to the data on a disk covered by a specific locking range, the Locking table contains columns ReadLockEnabled, WriteLockEnabled, ReadLocked, and WriteLocked. The data inside the LR cannot be read when ReadLockEnabled and ReadLocked are both set to true. Otherwise, the LR range is locked, and

the data cannot be read. Similarly, for writing and WriteLockEnabled and WriteLocked. The separation of the locking right into the LockEnabled and Locked column gives the ability to have separate ACE for the LockEnabled column, meaning that, for example, an admin can decide that the user can only lock reading for a specific LR but not writing.

Each locking range also has a defined column LockOnReset containing a list of types of resets on which the LR gets locked. Although there are defined types such as HotPlug (which is not really defined anywhere, but we suspect that it corresponds to the ATA Hot Plug reset)... Opal requires the support of only the Power Cycle, Programmatic[1] and optionally Hardware Reset reset.

3.9.3 Single User Mode

In many cases, it might be desirable to prevent admins from accessing user data, even though the admins generally have more competence. The Single User Mode Feature Set [8] defines a way to achieve this. After activating Single User Mode, only a single User authority is capable of changing their authority object (this includes changing their PIN), changing the proprieties of LRs assigned to them (which includes the lock state of the LR) and generating new keys for those LRs. The only actions available to the Admin authorities related to that LR are destructive actions.

3.9.4 MBR shadowing

This feature enables the disk to provide a fake master boot record (MBR). Instead of the MBR saved on the disk, the disk instead provides the shadow MBR saved in a table on bootup. The shadow MBR may contain software to enable the host to authenticate itself to the disk and unlock it. After the host is authenticated, the shadow MBR may be deactivated, and the regular disk data will be available again.

This feature is controlled using tables `MBR`, a byte table containing the data to be presented while the shadowing is active, and `ControlMBR`, an object table used to control the shadowing. Table `ControlMBR` one row with columns `Enable`, `Done`, and `MBRDoneOnReset`. When `Enable`

1. Activated by TPER_RESET, command on TPer layer.

is true and Done is false, the shadowing is active, and the data from MBR table can be read from the disk. The column MBRDoneOnReset specifies a list of types of resets during which the Done is set to false.

3.10 Opal SSC

Even though the Core specification introduces many powerful and interesting features, this might increase the cost of design and manufacturing of a fully compliant device. To solve this problem, the Opal SSC determines only a small set of mandatory features, leaving most of the features optional. Together with the set of mandatory features it also states range limits of certain properties.

For access control, Opal is mandating only password authentication, this means that any other authentication such as implicit authentication of the host or any authentication of the TPer (and therefore also secure messaging) may not be available. For communication Opal requires only support of synchronous communication. For table management, Opal does not require support of creation or deletion of tables. Issuance of SPs is also not required, and Opal instead uses SPs preconfigured by the manufacturer. Out of the previously mentioned SPs, Opal requires only that the Admin SP and the Locking SP are supported. Since SP issuance is not one of the mandatory features, the Locking SP may also be preconfigured by the manufacturer and be initially in the "manufactured-inactive" state.

Due to the range limits specified in the Opal SSC, the following features also might not be available.

- Opal specifies that the device must be able to handle: at least one method per SubPacket, at least one SubPacket per Packet, and at least one Packet per ComPacket. This means that any ...

- At least one active transaction per session, at least one active session per all the ComIDs and at least one ComID.

This means that an Opal-compliant device implementing only the required minimum will also not be able to provide any

But Opal does not only reduce the feature set from the Core standard, but also expands it with some extra feature sets. Even though

the Core specification does not require the following feature sets, the Opal SSC requires them additionally.

- Physical Presence SID (PSID) is a special authority that is authorized to call only the Revert method. The PIN of this authority may not be found out using the interface of the disk, every one of our disk supplied this information on a label on the disk. This authority provides a way to reset the TPer into factory state even in case the SID PIN is lost.
- DataStore tables are byte tables accessible for any use. Using the Activate method, the number of the DataStore tables with their sizes can be specified.
- Block SID Authentication feature disables SID authority until device restart. Can be used by BIOS to protect somehow... this functionality seems kind of a reach...

From our experience with Opal disks, most of them did not implement more than the required minimum (TODO: fact recheck later on). Some of the tested disks, even though they were described by the vendor and/or manufacturer as Opal-compliant, did not implement every required feature set. Out of the 6 tested disks, only 2 implemented the required Block SID Authentication feature set. However, even those 2 implementing the feature set did not support the actual feature.

3.11 Host-side implementation

Other than using generic SED software introduced in chapter 1, there are more low-level approaches to controlling Opal hardware available, which offer more control over the device. In this section, we will introduce two such approaches available in Linux systems.

3.11.1 Linux ioctl

Since version 4.11 the Linux kernel offers set of ioctl requests to facilitate control over an Opal disk [9]. Although these ioctl requests offer a simple access to control of the disk, not every feature mandated by Opal is implemented. Some of the limitations are the following:

- Access to only single admin authority and up to 9 user authorities.
- No way to change SID PIN. This means that the authority representing the owner of the device, that often has control over the entire disk, is stuck with the PIN that was chosen during the taking of the ownership.
- no "write only" lock
- no configuration of lockout, and no way to reset lockout — doesn't matter since opal has read-only lockout values -> not implemented
- The last mentioned is the missing ability to read or write to object table rows or iterating tables. This prevents the possibility of replacing the missing features using a direct setting of values in a table.

The individual implemented ioctl requests for Opal functionality and their function as of Linux kernel 5.19 are:

- `SAVE` — adds a key for a locking range into a unlock list, so that it can be used after waking the disk up from suspend. To wake the disk up, exported symbol `opal_unlock_from_suspend` can be used to unlock the disk with the saved data. Note that the data is saved only in the RAM, and so this cannot be used to wake the computer (??? What did i mean by this?... anyway this is entry point for some vulnerability analysis because it basically gives us back the cold boot attack!).
- `LOCK_UNLOCK` — locks or unlocks reading or writing for the selected locking range.
- `TAKE_OWNERSHIP` — changes the admin authority password from the default one to the selected one. TODO: write about initialisation of the TPer to the generic chapter, `C_PIN_MSID` etc.
- `ACTIVATE_LSP` — changes state from "Manufactured-Inactive" to "Manufactured" state. Also facilitates setup of single user mode. TODO: write about TPer states in the generic chapter.

- `SET_PW` — changes the password of the selected authority, using the admin password.
- `ACTIVATE_USR` — enables specified user in the Opal tables.
- `REVERT_TPR` — reverts the TPer to the manufactured state using admin password.
- `LR_SETUP` — sets locking range position/locking enable
- `ADD_USR_TO_LR` — sets the ACE for locking and unlocking to the designated user. Does not actually add a user, instead replaces any existing one by the new one.
- `ENABLE_DISABLE_MBR` — changes `Enable` parameter of the `MBRControl`.
- `ERASE_LR` — calls the erase method, 00 00 06 00 00 08 03, can't find it in opal and in core it's "reserved for SSC", ... it's in single user mode standard — not only destroys data, also removes pin and user authority (for SUM).
- `SECURE_ERASE_LR` — regenerate the data encryption key of a range to destroy the previous one.
- `PSID_REVERT_TPR` — resets the TPer to the manufactured state using PSID.
- `MBR_DONE` — changes `Done` parameter of the `MBRControl`, only when both the Done and Enable parameters are enabled, MBR shadowing is active.
- `WRITE_SHADOW_MBR` — writes data into the MBR table.
- `GENERIC_TABLE_RW` — reads a byte table or writes into a byte table. Neither object tables nor iteration are supported.

Currently, there is no documentation to be found for the ioctl requests for Opal functionality. The information in the previous list was acquired from our code analysis. Short program showcasing the usage of this interface can be seen in the appendix 1.

3.11.2 Direct communication

Alternative to the Opal ioctl requests are disk controller ioctl requests. Depending on the disk protocol, a different way of passing the Opal commands is required, such as using `SG_IO` ioctl and `sg_io_hdr_t` structure for SCSI disks or `NVME_IOCTL_ADMIN_CMD` ioctl and `nvme_admin_cmd` structure for NVMe disks. Using these structures, the Opal commands described in chapter 1 can be sent to the TPer. Compared to the Opal ioctl requests this has the advantage of not being limited only to a subset of features that the Opal ioctl requests implement, and instead being able to use every Opal feature the device offers. This is not limited only to ... (e.g. improve performance by using optional features such as concurrency, etc, described earlier) ...

Although this approach gives the user access to every feature of the Opal disk, it also requires them not only to implement the command hand-over for each type of disk separately, but also to create the methods and parse the method results, both described in chapter 1, on their own.

TODO disk interface popsat[**NVME**]

The commands that are called through the disk controller ioctl requests are called IF-RECV/IF-SEND in TCG Storage standards [4]. These commands corresponds to several other commands in different interface specification. Some of them are the Security Receive/Security Send commands in NVMe [10], the TRUSTED RECEIVE/TRUSTED SEND commands in ATA [2], or the SECURITY PROTOCOL IN/SECURITY PROTOCOL OUT commands in SCSI [11].

4 Existing tools

In order to give users the ability to manage their SED disks without the need to create their own programs to access existing interfaces, there exist several tools.

In this chapter we will look at several existing tools that allow one to manage encryption of disks.

4.1 Cryptsetup

Cryptsetup[1] is a tool for disk encryption setup. Although this tool supports several formats and volumes, such as ..., all of them are software, and there is no support hardware encryption support as of now.

4.2 sedutil

sedutil[2] is a tool for management of SED disks, maintained by the Drive Trust Alliance. It currently supports the Enterprise and Opal SSCs (and additionally the remaining Pyrite, Opalite and Ruby SSCs in a fork of the project[3]). Even though it is currently the largest open-source project to control SED disks, the code repository does not seem to be currently active.

This tool does not use the Linux Opal ioctl interface and instead uses the approach described in the subsection 3.11.2. This allows the tool not only to be multi-platform but also to use the Opal features in their entirety instead of only the subset introduced in the Opal ioctl interface. ... although there currently certain limits for e.g. having only one admin ...

The tool offers the following commands, grouped by their functionality:

- Discovery:

1. `https://gitlab.com/cryptsetup/cryptsetup`
2. `https://github.com/Drive-Trust-Alliance/sedutil`
3. `https://github.com/ChubbyAnt/sedutil`

  - `isValidSED` — lists SSCs supported by the selected disk.
  - `scan` — performs `isValidSED` command on every disk in the system.
  - `query` — performs Level 0 and Level 1 Discovery on the selected disk.
  - `printDefaultPassword` — prints the MSID password.

- TPer management:
  - `initialSetup` — takes ownership of the disk, and initializes Locking SP, global LR and shadow MBR.
  - `setSIDPassword` — changes the password of SID authority.
  - `setAdmin1Pwd` — changes the password of Locking SP's first admin authority.
  - `setPassword` — changes the password of any supported Locking SP's authority.

- Locking range management:
  - `listLockingRanges` — prints information about all LRs,
  - `listLockingRange` — prints information about a single LR,
  - `rekeyLockingRange` — regenerates the LR's key (destroying the data in the LR in the process)
  - `setupLockingRange` — sets the start and length of locking range,
  - `setLockingRange` — sets ReadLocked and WriteLock of the LR depending on the chosen configuration (read-write, read-only, or locked).
  - `enableLockingRange` — sets ReadLockEnabled and WriteLockEnabled of the LR.
  - `disableLockingRange` — unsets ReadLockEnabled and WriteLockEnabled of the LR.

- Shadow MBR management:

  - `setMBREnable` — sets Enable column of the MBRControl table.
  - `setMBRDone` — sets Done column of the MBRControl table.
  - `loadPBAimage` — writes the contents of a file into the MBR table to be used as the shadow MBR.

- Reverting the device:
  - `revertTPer` — reverts the TPer using the MSID password.
  - `yesIreallywanttoERASEALLmydatausingthePSID` — reverts the TPer using the PSID password.
  - `revertNoErase` — reverts only the Locking SP, keeping the global range data.

- Enterprise-specific functionality:
  - `setBandsEnabled` —
  - `setBandEnabled` —
  - `eraseLockingRange` —

4.3 hdparm

The *hdparm*[4] tool provides a command line interface to control parameters of ATA disks. Among others this tools gives access to control ATA Security Feature Set. The examples of SED relevant commands for functionality described in 1 are:

- Enable the ATA Security Feature Set:
  `hdparm --security-set-pass pwd /dev/sda`
- Disable ATA Security Feature Set:
  `hdparm --security-disable pwd /dev/sda`
- Unlock the disk:
  `hdparm --security-unlock pwd /dev/sda`

4. `https://sourceforge.net/projects/hdparm`

- Erase the disk:
  `hdparm --security-erase pwd /dev/sda`

Implicitly the user password is referred to in the commands, but `--user-master` can be specified to select master or user password explicitly. While enabling the ATA Security Feature Set, it is also possible to use the `--security-mode` option to choose between high and maximum security mode. A command to lock the drive is missing because a drive with enabled ATA Security Feature Set is locked when powered off.

4.4 Proprietary solutions

Some disks might provide their own software for their own proprietary hardware encryption. These are usually found separately on website or something, but some are provided as by the disk on an unecrypted partition.

4.5 go-tcg-storage

The *go-tcg-storage*[5] is a Go library that does everything, and is actually supported. It has a few tools that let you do some things. It also supports more SSCs than others. Just found out about it, will need to look into it deeper and figure out why is it worse, so we won't look as bad.

5. `https://github.com/open-source-firmware/go-tcg-storage`

5 Security of hardware encryption

Self-encrypting disks usually do not ... For example, the Opal standard aims to only "Protect the confidentiality of stored user data against unauthorized access once it leaves the owner's control (following a power cycle and subsequent deauthentication)" [3]. Even if the device is successfully protected against such a threat model, it still leaves many possible attack vectors. To extend the threat model into a more realistic scenario, we consider an attacker that has physical access to the disk installed in a "locked" computer or something.

It should be noted that some systems

In order to limit the scope of the analysis, we will focus primarily on the hardware... The threat model we consider in this analysis is primarily going to be the same ...

5.1 Attacks on hardware encryption

In this chapter, we will provide an overview of state-of-the-art attacks relevant to hardware disk encryption. For each attack, we will provide our theoretical analysis of the protection offered by Opal, if properly implemented in a disk, against such an attack. For a selection of the attacks, we will also provide our practical insight.

TODO: clean up with citations from the vulnerabilities papers[12, 13, 14, 15, 16, 17].

5.1.1 Evil maid attack

An evil maid attack consists of a situation where the device is left unattended, and the attacker has full physical access. Such attacks usually have two phases. In the first phase, the disk is modified, e.g. by installing custom firmware or inserting a probe to eavesdrop on the communication. Then, the victim uses the disk, providing the password to unlock it. In the second phase, the attacker returns to collect the obtained information and undo their changes. In some cases, the attacker might not require physical access to the device and instead use a network to receive information and have, e.g. firmware reverse itself (or leave it be if the discovery of the attack is not a problem).

Hmmm, maybe something like: Opal does not specify anything about firmware update, and shadow MBR update requires authentication. Still, one could still listen to the cables, I suppose because Opal does not mandate secure messaging and so the communication can be simply sniffed out on the ATA/NVMe/whatever cables?

5.1.2 Attack on bad RNG

Attack on bad RNG is such an attack which abuses RNG generating data with low entropy or even entirely predictable.

Since Opal specification does not actually specify any implementation for the RNG, we cannot perform any fruitful analysis of the standard itself. However, we can still perform practical analysis.

The Opal specification offers a few ways how to generate random data. The most useful would be GenKey used for generating DEKs, but since the destination cells are protected from reading, we cannot get to them easily. Another method of generating random data is the method Random. This method allows us to get at least 32 bytes [3] every invocation. Using this method, we have decided to generate random bytes to inspect later. From the disks we have inspected we have found the following:

- SanDisk X300s — first Random invocation of session always return the same bytes. The value of the first bytes seems to change only after the new activation of Locking SP. Following invocations of the Random method in a session return different bytes.

- Kingston KC600 — for the minimum 32 bytes, the TPer closes the session on Random invocation. The same response was for most of the other tested values. However, if exactly $256^n - 1$ bytes are requested, $128 + n$ bytes are returned... lol, I see where the problem is now. The TPer doesn't parse it as a token but as a C unsigned integer. So, it was reading the header instead, and the `0xff` bytes were required for "NOPs" since they are empty atoms. Using the parsing now, we can get up to 255 bytes from this disk at once.

- Samsung disks — return seemingly random values.

The results of dieharder's Diehard test suite found no statistical something problem. Note that for the Kingston KC600, we did not include the first batch of random bytes.

SanDisk X300s — However, using GenKey on a locking range does change the data into random bytes, so it does not seem that this affects key generation.

5.1.3 Attack on bad interface?

An attack on the interface is such an attack where problems in the interface are taken advantage of. This might include attacks such as using vendor commands to read the disk directly, even if the disk is unlocked... In the past, many hardware encrypted disks were vulnerable to such attacks, requiring only the usage of undocumented vendor commands to read either the data of the encryption module (in some cases containing the unencrypted DEK) or even the content of the disk itself.

As can be seen from the previous subsection 5.1.2, the implementation of the interface is not perfect for some of the disks. Something like non-security issues might very likely be a sign of possible security issues, so don't take those problem that we found lightly! Maybe also something that more information can be found in the data chapter or something like that.

TODO: rethink the subsection name.

5.1.4 Cold boot attack

In this attack, the property of volatile memories is used. Even though volatile memories lose the data stored in them after powering off, they do not lose the data instantly. This introduces the cold boot attack, which takes advantage of this fact. One of the ways is to take a RAM stick from the victim's locked computer and install it into the attacker's computer, where it can be read. Another possible approach is the reboot the victim's computer into the attacker's system.

Even though this attack works quite well against software-based disk encryption and software encryption in general, against purely SED, it does not work that well. This is because, compared to software encryption, there is no need to store the encryption key in the

computer's memory. The key is instead stored in the memory of the disk controller, where it can (hopefully) be well protected against removal... (and also, it's not a generic RAM stick). However, this advantage holds only for the case where the key or the disk password is not kept in the computer's memory, and this may not always be the case with SEDs. In the Linux Opal ioctl commands introduced in an earlier chapter, there is introduced functionality to save the password for locking range in the memory. This functionality is introduced in order to allow the computer to wake from suspension automatically since the disk might need to be unlocked in such a case. But since the key is stored in the computer's memory, it is possible to acquire it using the cold boot attack.

5.1.5 Hot plug attack

Hot-plug attacks [18] are attacks similar to cold boot attacks. However, compared to the cold boot attacks, where the RAM is inserted into the attacker system before the data in it is naturally destroyed, hot plug attacks instead move the disk into the attacker's system. This attack abuses the fact that SATA disks have a data cable and a separate power cable. Because of this, it is possible to switch the data cable into a different device without powering off and therefore locking the disk.

In the Core standard, there exists a reset type called HotPlug. Although this reset type is not described closer than the name in the standards, it most likely refers to the ATA/NVMe? hot plug TODO[**TODO**]... Using this reset type, it could possible to set an LR to lock when a hot plug is detected. However, as this reset type is not mandatory (nor optional) for Opal devices, and none of our Opal devices implemented it, we were not able to examine this solution further. However, even if the device would implement the HotPlug reset type, using it to prevent the hot plug attack would not be advisable as the hot-plugging process could be disabled in the attacker's system.

6 Data (and our utility)

In our analysis we have tested several internal SSD devices, by multiple manufacturers and with different interfaces.

6.0.1 Program structure

Is there something to really talk about?

As the Core standard introduces an entirely? new protocol for the communication and the stream encoding? we had to implement that ourselves.

In order to

6.0.2 Disk capabilities discovery

In order to collect information about Opal capabilities of the disks, we have written an utility program. This program uses the direct communication described in section 1 to communicate with the disk. We have decided to use the direct communication instead of the primarily because the capability to perform the discovery process using the Linux Opal ioctl is currently not possible. Even though there are patches being suggested, even if they were to be accepted, only the newest version of Linux kernel would support this feature, limiting the sample size. However, because this approach requires separate implementations of the disk interface commands, something about how this limits us only to a few disk interfaces because of testing, but it's not such a problem because they have a majority share, so find some source that this is actually true. The utility program performs the level 0 and level 1 discovery and the identify command to gather information about the disk. The aggregated and formatted output can be seen in table **??**.

Compared to the two previous levels, the level 2 discovery is more complicated. The level 2 discovery is based on reading tables. Since the standard leaves space for vendor unique tables, we need to first discover these tables. This is done by first reading the SP table of the Admin SP, getting a list of all the SPs in the TPer. Afterwards, we read the Table table of each of the found SPs, getting a list of all tables in each of the SPs. Since each table used in this table discovery process is

mandatory, we should be able to get a list of all the tables in the TPer this way. Finally, we can iterate through all the tables, read each entry, and save the contents.

Output of the level 2 discovery depends on several factors: the state of the TPer, the authority used during the discovery... The state of the TPer defines which tables and SPs are available..., It is not desirable to change this state by activating the Locking SP, since that could have unwanted consequences for the future use of Opal on this device by the owner. The admin authority on the other hand could be optionally provided by the user. However, none of the authorities can access all the tables[1], and the information accessible by higher authorities is only potentially sensitive information. So this is a non-issue.

The first part of the table contains values acquired through the level 0 discovery, the second part (starting with `MaxComPacketSize`) contains values obtained through the level 2 discovery. In case the disk did not report a value, empty cell is used. There are some noticeable differences: since the first part is reported through firmly established C headers with static form, the numbers are parsed directly as C integers... the second part uses the TCG Storage protocol, with tokens that can be of different sizes — notably in `/dev/sdb` which returns numbers encoded with 4 bytes, even when not necessary. Values of some variables are dependent on each other. For example, because `MaxPackets` is always the minimum (1), then `MaxPacketSize` = `MaxComPacketSize` − (fixed size of `MaxComPacketSize` header).

Although not a part of the discovery process, the utility can also access other functions that tell us more about the device, such as the Random method to generate random data.

6.1 Disk management

Other than the Discovery process described earlier, the utility can also be used for managing a SED device. For this application it offers several commands. The commands allow basic interaction with a SED disk, such as taking ownership of the disk, reverting the disk, creating

1. Counterintuitively, access protected by `ACE_ANYBODY`, can be only be accessed by the Anybody authority and not any of the higher authority authorities, such as Admin authorities.

locking ranges, or locking the disk. However, it is possible to easily extend the tool to support also more advanced use cases, something about how it's thanks to its good quality ;).

TODO: something we can do better than sedutil????

6.2 TODOs

TODO: describe how different disks accept "bad" input: some disks don't mind empty list in parameter, other disks hate it (even close the session, some just return empty list as response but success); or alignment of packets:

TODO: using Opal before taking ownership — what happens if I just use default password for everything — works just fine :)

TODO: try to lock only reading, how to write/can I write? – can do just fine :)

TODO: why do some disks have maxinstances of base template one??? TODO: missing DataRemovalMechanism table

TODO: activate sets from manufactured-inactive to manufactured, admin sp starts in manufactured, how to use datastore extension of actiavte then? SPObjectUID.Activate[DataStoreTableSizes = list [uintegers]] => [ ]

7 Conclusion

In the thesis we have introduced several approaches to using hardware-encrypted block devices in Linux. Then, we have focused on the Opal standard more deeply, ... We have then compared capabilities of several Opal-compliant disks. And finally we have provided an overview on attacks on such devices and their feasibility.

As can be easily seen from ...

8 TODO

Holding onto papers here: [16], [15], [17], [13], and [12].

Bibliography

1. *Verizon 2022 Data Breach Investigations Report*. 2022. Tech. rep.
2. *ATA/ATAPI Command Set - 3*. Tech. rep. Available also from: ...d2161r5-ATAATAPI_Command_Set_-_3.pdf.
3. *TCG Storage Security Subsystem Class: Opal*. 2022-01. Version 2.02. Standard. Trusted Computing Group. Available also from: https://trustedcomputinggroup.org/resource/storage-work-group-storage-security-subsystem-class-opal.
4. *TCG Storage Architecture Core Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-architecture-core-specification.
5. *TCG Storage Security Subsystem Class: Opalite Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-security-subsystem-class-opalite.
6. *TCG Storage Security Subsystem Class: Pyrite Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-security-subsystem-class-pyrite.
7. *TCG Storage Security Subsystem Class: Ruby Specification*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-security-subsystem-class-ruby-specification/.
8. *TCG Storage Opal SSC Feature Set: Single User Mode*. Tech. rep. Available also from: https://trustedcomputinggroup.org/resource/tcg-storage-opal-ssc-feature-set-single-user-mode/.
9. *SED OPAL Library*. Tech. rep. Available also from: https://lore.kernel.org/lkml/1482176149-2257-1-git-send-email-scott.bauer@intel.com/.
10. *NVM Express® Base Specification*. Tech. rep. Available also from: https://nvmexpress.org/developers/nvme-specification/.
11. *SCSI Primary Commands - 4*. Tech. rep. Available also from: https://www.t10.org/members/w_spc4.htm.

12. BOTEANU, Daniel; FOWLER, Kevvie. Bypassing Self-Encrypting Drives (SED) in Enterprise Environments. In: 2015.

13. ALENDAL, Gunnar; KISON, Christian; MODG. got HW crypto? On the (in)security of a Self-Encrypting Drive series. *IACR Cryptol. ePrint Arch.* 2015, vol. 2015, p. 1002.

14. MÜLLER, Tilo; LATZO, Tobias; FREILING, Felix C. *Self-Encrypting Disks pose Self-Decrypting Risks: How to break Hardware-based Full Disk Encryption*. Available also from: `https://faui1-files.cs.fau.de/filepool/projects/sed/seds-at-risks.pdf`.

15. MÜLLER, Tilo; LATZO, Tobias; FREILING, Felix C.; FRIEDRICH-ALEXANDER. Self-Encrypting Disks pose Self-Decrypting Risks How to break Hardware-based Full Disk Encryption. In: 2013.

16. MEIJER, Carlo; GASTEL, Bernard van. Self-Encrypting Deception: Weaknesses in the Encryption of Solid State Drives. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 72–87. Available from DOI: 10.1109/SP.2019.00088.

17. MÜLLER, Tilo; FREILING, Felix C. A Systematic Assessment of the Security of Full Disk Encryption. *IEEE Transactions on Dependable and Secure Computing*. 2015, vol. 12, no. 5, pp. 491–503. Available from DOI: 10.1109/TDSC.2014.2369041.

18. MÜLLER, Tilo; LATZO, Tobias; FREILING, Felix C.; FRIEDRICH-ALEXANDER. Self-Encrypting Disks pose Self-Decrypting Risks How to break Hardware-based Full Disk Encryption. In: 2013.