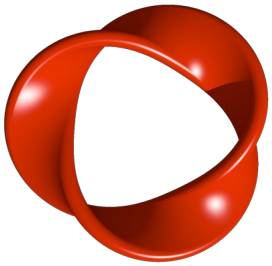

Secfault Security

NetHSM

Security Assessment

Report
Final

for

Nitrokey GmbH

Jan Suhr
Rheinstr. 10 C
14513 Teltow

- hereafter called "Nitrokey" -

Document History

Version	Author	Date	Comment
0.1	Maik Münch	2020-12-17	First Draft
0.2	Gregor Kopf	2020-12-18	Additions and Corrections
0.3	Dirk Breiden	2020-12-18	Internal Review
0.4	Gregor Kopf	2021-03-13	Additions after Retest
1.0	Gregor Kopf	2021-06-24	Final Version

Table of Contents

1 Executive Summary

Secfault Security was tasked by Nitrokey to perform a review of the "NetHSM" solution, which aims at providing an Open Source, extensible hardware security module. The focus of the review was the solution's software, which was subject to a manual, static analysis. Manual dynamic tests have been performed against a Docker image of the solution. Details about the test methodology and the scope are provided in section 2.1 and 2.2 of this document.

During the review, a number of vulnerabilities and weaknesses have been identified, which are described in detail in section 4 of this document.

One of the more severe issues was that setting the device to unattended mode and then back to attended mode currently (i.e., without hardware-level measures such as a TPM in place) implies the ability to bring the device into unattended boot mode again if physical access to the device is given.

Other findings of the security review included a persistent Cross Site Scripting issue, bypassing of key usage restrictions and a Denial of Service condition.

Most issues have been addressed by Nitrokey during project execution and the findings' retest status is also provided in the corresponding finding's description in section 4 of this document.

It should be noted that a number of hardware-related aspects have not yet been fully implemented (e.g., a TPM or secure element) and hence the review did not focus on threats emerging from this. A first step for introducing a TPM has been made while the fix verification phase of the project was in progress, namely the introduction of a device ID stored in the TPM NVRAM. However, the TPM integration is not fully finished at the time of writing this report.

The overall design of the solution (e.g., using the formally verified Muen separation kernel, as well as implementing the business logic in a high-level language) left a positive impression. These design choices rule out a number of possible implementation issues, such as memory corruption problems, upfront. The code was found to be well-structured and readable. The identified issues have been addressed in a timely manner.

2 Overview

Nitrokey is a company specialized in open source security hardware. The product line covers various aspects of IT security including hardware for data encryption, key management and user authentication. Nitrokey 's current goal is to get an overview of unknown security-related vulnerabilities and design issues present in the new HSM implementation named NetHSM.

2.1 Target System and Scope

Nitrokey tasked Secfault Security to review the NetHSM solution, which aims at providing a modern, Open Source HSM. The overall solution consists of several individual components. The core business logic is implemented in OCaml, running as a Mirage Unikernel[1] on top of the Muen[2] separation kernel. Network connectivity is provided by a separate Linux instance, which is running as a Muen subject and bridges the system's network interface towards the HSM Unikernel. Furthermore, the storage system is also provided by a separate Linux system running as a Muen subject, which provides a git[3] interface towards the HSM Unikernel.

The HSM is able to operate in two modes, namely attended and unattended mode. In the unattended mode, the device derives a Unlock Key from the unique Device ID, while in attended mode the HSM waits for the user to provide an Unlock Passphrase. When the unattended boot fails, the HSM automatically falls back to the attended boot mode.

Further, a user role model is implemented. Each user account is configured with exactly one of the following roles:

- **R-Administrator** - Access to all operations, exception for key usage operations, i.e. message signing and decryption.
- **R-Operator** - Access to all key usage operations, a read-only subset of key management operations and user management operations allowing changes to their own account only.
- **R-Metrics** - Access to read-only metrics operations only.
- **R-Backup** - Access to the backup-related operations only.

The review has been performed as a static manual code review with additional manual dynamic testing. The tests have been performed against a Docker-based[4] version of the system, not running on the actual target hardware. The aim of the review was to uncover software-related vulnerabilities and weaknesses.

Hardware-related attacks have not been considered during the project. Furthermore, the following aspects have been defined as out of scope for the assessment:

1 https://mirage.io/
2 https://muen.codelabs.ch/
3 https://git-scm.com/
4 https://docker.io

- TPM, secure element or other hardware-based security devices such as random number generators; these are planned to be added at a later stage of development.
- Physical attacks against the system while running in unattended mode

Physical attacks against the hardware platform itself (e.g., attaching probes to measure timings or power consumption, writing to flash chips, attaching to serial or debug ports etc.) have not been considered during this software-focused review.

The following commit hashes have been considered:

- nitrohsm - `c85e4656b7b00e3cfbbf66fc3f803806eaf82626`
- nitrohsm-grub - `fc2a26b5703adbf282497df9a664a37eea573168`
- muen - `53d03f6097df78a50bfe7a448ba85887bc9fcf9e`
- nitrohsm-coreboot - `de66d04f8da415ce088d794c8a1419ac5d5dfdf5`
- nitrohsm-muen-linux - `549738f15da0e5a00275977623be199fbbf7df50`

The fix verification was performed based on commit hash `73ec09896e40a64455341fb6f9b39ef9c839cef7`.

2.2 Test Procedures

For an optimal project execution, Nitrokey provided full access to the solution's source code, including documentation.

The following subsections will detail on the test procedures, which included a manual code review and targeted dynamic tests.

No third-party tools like code analyzers or existing fuzzers have been used during the project.

2.2.1 Manual Code Review

The main activity of the project was the source code review of the provided solution. The code review took place in two phases, as described below.

- Information Gathering
- Code Audit

2.2.1.1 Information Gathering

As an initial step prior to performing the actual review, an information gathering phase that collects information about the target application and its assisting technologies was performed.

Several aspects of the software are examined in this stage, including:

- Used technology stack

- General software architecture
- Components comprising the solution

Additionally, a conference call was held to further elaborate on the solution's intended security promises and it's general threat model.

For performing the subsequent code review step, there are generally three main strategies that can be used during the code review of a project. These are: the "top-down" approach, the "bottom-up" approach and the "hybrid" approach.

2.2.1.2 Top-down

This approach follows the "Waterfall" software development paradigm. It starts off with the high-level understanding of the application code via the data collected in the information gathering stage. It then divides the software into logical and functional units and these are reviewed for possible connections to unsafe input handling methods. From an attacker's perspective, controlling the input data is the main way of influencing the execution flow of an application, bringing the execution to invalid/unauthorized states.

The top-down approach first discovers vulnerabilities in the architecture and logic of the application and then moves on to evaluate bugs in the implementation details. The approach is well-suited for large scale solutions, in particular if the work of multiple review teams has to be performed and coordinated.

2.2.1.3 Bottom-up

The "bottom-up" approach starts with the immediate review of the source code of the application, without having a high-level view of the software architecture, or the relations between the various software modules. The reviewer essentially builds a picture of the application's operation from the application's implementation details. This provides the reviewer with an early view of issues and threats that are caused by implementation bugs.

For small projects, this approach can be beneficial as it provides a direct insight into the inner workings of the target solution, without time-consuming orientation phases.

The "bottom-up" approach however also introduces a considerable delay to the review process for larger solutions, as it reevaluates which parts of the project are considered critical, each time a new segment of code has been reviewed.

2.2.1.4 Hybrid

The hybrid approach is a mixture of the previous two approaches. It starts off with a high-level evaluation of the application and its subsystems. Each subsystem is then reviewed either via the top-down approach, or the bottom-up approach, depending on a judgment made by the reviewer for that particular software module. The main advantage of this methods is that the reviewer builds a model

of possible attacks, based both on the general architecture of the application and its implementation details.

For executing this project, the Hybrid approach was chosen to be the most efficient way of approaching the codebase. After identifying and obtaining a basic understanding of the individual components, their implementations have been reviewed in a bottom-up fashion. This approach was significantly eased by the good readability and structure of the provided source code.

2.2.2 Vulnerability Scope

The main focus of the review was the identification of potential and actual vulnerabilities in the provided source code. The possible vulnerability types in scope typically depend on the particular solution, used languages, frameworks etc.

As the main parts of the solution were written in Ocaml, a language that provides memory-safety guarantees, the focus was put on the identification of general logic issues and implementation issues in the HSM implementation itself. Vulnerability classes in this area include for instance privilege elevations or missing authorization checks in the implemented user role model, cryptographic issues, insecure storage of sensitive material such as cryptographic keys, or web-related issues inherent to the used web-based API.

2.2.3 Targeted Dynamic Tests

In order to verify assumptions and to strengthen the general understanding of the reviewed solution, targeted dynamic testing has been performed. This includes for instance the web-based API implementation, which provides the main interaction facility of the HSM.

2.3 Project Execution

The project has been executed in the time frame from 2020-12-07 to 2020-12-18.

The consultants assigned to this projects were:

- Maik Münch
- Jennifer Gehrke
- Gregor Kopf

3 Result Overview

An overview of the project results is provided in the following table.

Description	Chapter	Type	Exploitability	Attack Impact	Status
Key Usage Restriction Bypass via Raw RSA Decryption	4.1	Code	Medium	Low	Closed
Use of PBKDF2	4.2	Code	Low	High	Closed
Missing Input Sanity Checks for Several REST Operations	4.3	Code	Medium	Medium	Closed
Persistent XSS Via Certificate Upload	4.4	Code	Low	High	Closed
No Secure Wiping	4.5	Design	Low	High	To be re-evaluated
Changing the Unlock Passphrase without Knowing It	4.6	Code	n/a	n/a	To be re-evaluated
Rate Limiting Restrictions	4.7	Code	Low	Medium	Closed
Use of PKCS#1v1.5 Decryption	4.8	Design	Low	Low	Closed
Missing Range Checks in ECDSA Implementation	4.9	Design	Low	Low	Closed

Each identified issue is briefly described by its title, its type, its exploitability and by the impact of a successful exploitation. Technical details for the individual issues are provided in the respective sections of chapter 4 of this document. Details regarding the vulnerability rating scheme used in this document are provided in section 6.

4 Results

The issues identified during the project are described in detail in the following sections. For each finding, there is a technical description, recommended actions and - if necessary and possible - reproduction steps. For details regarding the used vulnerability rating scheme, please refer to section 6 of this document.

4.1 Key Usage Restriction Bypass via Raw RSA Decryption

Summary

Type	Location	Exploitability	Attack Impact	Status
Code	Keyfender	Medium	Low	Closed

Technical Description

An analysis of the HSM's RSA usage revealed that RSA keys stored on the HSM can be configured to be usable for encrypting and/or signing data. However, the decrypt function of the HSM module appears to include support for raw RSA decryption[5].

Being able to perform a raw RSA private key operation appears to imply the ability to build signatures as well (simply by "decrypting" a properly-crafted bit string, in the same way as this is done in the actual signature generation). This means that an attacker could be able to use an RSA key that has the purpose `Decrypt` to also compute signatures, hence bypassing key usage restrictions.

Enforcing key usage restrictions is not one of NetHSM's primary security goals. Hence, the impact of this issue is rated low.

Recommended Action

To address this issue it is advised to restrict the raw RSA decryption to keys that have specified `SignAndDecrypt` as purpose.

Retest Status

The original implementation of the key usage restriction focused on the idea of allowing or denying certain operations like signing or decrypting. This concept has been reworked. The key usage restriction mechanism now focuses on allowing or denying particular cryptographic operations, such as PKCS#1v1.5 signatures or PKCS#1v1.5 decryption. This allows users to make a more conscious decision about what a key may be used for, including possible risks of allowing keys to be used for legacy schemes. A related issue is documented in section 4.8.

5 Key.decrypt function in /nethsm/src/keyfender/hsm.ml

4.2 Use of PBKDF2

Summary

Type	Location	Exploitability	Attack Impact	Status
Code	Keyfender	Low	High	Closed

Technical Description

While reviewing the `keyfender` source code, it was found that the implementation of `key_of_passphrase` in `crypto.ml` makes use of PBKDF2 for key derivation. While PBKDF2 is hardened against a number of attacks like rainbow tables, it is not hardened against parallelizing brute-forcing by building custom ASICs. An attacker with access to such ASICs might therefore be able to make offline attacks more feasible, which might for example result in the disclosure of the key material stored on the HSM or the corresponding backup files.

Recommended Action

As memory-hard schemes like scrypt or Argon2 do include protection against this ASIC-based attack vector, replacing the use of PBKDF2 by one of these schemes is recommended.

Retest Status

For deriving cryptographic keys from the unlock passphrase and from the backup passphrase, the adjusted HSM implementation makes use of the scrypt key derivation function that provides better protection against parallelization and special purpose hardware in offline guessing attacks. The currently defined parameters determining the execution costs cannot be evaluated without access to the final hardware. Once the hardware is settled, they need to be adapted to the maximum values acceptable from the usability perspective.

For storing user authentication passphrases, NetHSM now relies on HMAC-SHA256, i.e., does not make use of a memory-hard function. This is due to the fact that protecting user passphrases from attackers who can read out the HSM's storage is not one of NetHSM's security goals. Brute-force attacks against the unlock passphrase - which is required to access key material - are addressed by using scrypt. User passphrases should therefore be chosen uniquely for each NetHSM appliance, in order to prevent attackers from mounting brute-force attacks and subsequently using the obtained user passphrases on other devices or services.

4.3 Missing Input Sanity Checks for Several REST Operations

Summary

Type	Location	Exploitability	Attack Impact	Status
Code	API	Medium	Medium	Closed

Technical Description

A review of the exposed REST endpoints revealed that the HSM allows to assign IDs to keys and to users on the system. Such IDs can be provided either by the device administrator or are automatically generated by the HSM. The implementation uses these IDs as keys to store the respective items in their key-value (KV) stores. There is a check in place to ensure that user-provided IDs are alpha-numeric (`valid_id` function). However, this check is typically only performed during a POST or a PUT request; for GET or DELETE requests, the validity check has been found not to be in place.

This can result in the deletion of items from the respective KV store, which in turn can at least be used to mount a DoS attack against the system.

Recommended Action

To address this issue it is advised to implement the validity check on all REST endpoints.

Reproduction Steps

The following command illustrates the issue:

```
curl -v --path-as-is -s -k
https://admin:Administrator@10.0.0.1:8443/api/v1/users/.version.
```

This request results in the error `"message":"Could not write to disk. Check hardware."`. When performing a DELETE request instead of a GET request, the `.version` file from the user store will be removed, which will result in future errors when locking/unlocking or rebooting the system, rendering its services unavailable.

Retest Status

Overall checks, rejecting the string `.version` as an input that will be subsequently processed as part of a KV store key, were found to be applied. While this addresses the documented issue in the current situation, where `.version` is the only meta data entry in the stores, a more general approach mitigating this risk should preferred. A holistic utilization of the validation function `valid_id` would also address the issue. It further has the advantage, that it would protect any future meta data entries, whose key involves a special character.

4.4 Persistent XSS Via Certificate Upload

Summary

Type	Location	Exploitability	Attack Impact	Status
Code	API	Low	High	Closed

Technical Description

During the analysis of the web-based API it was found that, when adding a certificate to a key, the user can directly specify the Content-Type of the data that is to be stored. While this is convenient for storing different types of certificates, it also means that an attacker could set a content type like `text/html` or similar. This would allow them to upload HTML that will be rendered in a user's browser after navigating to the URI path `/keys/<ID>/cert`.

It should be noted that only administrative users can add certificates, which makes a successful exploitation of this issue more difficult. However, the impact of such an attack would be problematic as the injected content could include scripts that would trigger further actions to be performed on the HSM on behalf of the logged-in user, as browsers automatically send along the Basic HTTP authentication credentials. Furthermore, an attacker might make the web UI display messages asking the user to "change your unlock passphrase for security reasons" or similar, tricking users into providing the system's currently valid unlock passphrase to an attacker-provided piece of JavaScript code.

In order to trick the user into navigating to the attacker-provided contents, typical attack vectors include sending fake emails, including a direct link to the NetHSM web UI.

Please be aware that such an attack can not be performed without user interaction. As furthermore an administrative user account is required, the exploitability rating of this finding has been rated low.

Recommended Action

To address this issue it should be evaluated, if a restriction of the allowed content types is possible.

Retest Status

The implemented content type restriction addresses the described issue, as the allowed options should not lead to a parsing of the file contents as HTML or JavaScript data anymore. However, as the behavior depends on the utilized browser and is not standardized, it is advisable to make use of the `X-Content-Type-Options: nosniff` HTTP response header to mitigate the risk of MIME/Content sniffing.

4.5 No Secure Wiping

Summary

Type	Location	Exploitability	Attack Impact	Status
Design	Git storage backend	Low	High	To be re-evaluated

Technical Description

During the review it was found that the HSM implementation makes use of Git as a backend data store. All sensitive information in the store is encrypted using a symmetric key (please refer to section 5.2 for exceptions to this rule). This encryption key in turn is encrypted with the unlock passphrase or the device ID in case of using the unattended boot mode. When switching from unattended to attended mode, the key slot encrypted with the device ID is "removed", so that only the unlock passphrase can be used to unlock the HSM.

However, as git is used in the backend, restoring the key slot for unattended boot is possible via the git history. This means that an HSM that once was in unattended state and has not been fully re-provisioned afterwards, can be brought to unattended state again by an attacker with access to the git backend (e.g., an attacker who has physical access to the device). It should be noted that while the use of git offers a simple way for restoring an old device state, it might not be the only option for an attacker. As SSDs make use of wear-leveling, attackers might be able to restore old data even after deleting them from disk.

It should be further noted that this issue does not only affect the general storage protection, but can also be used to recover old keys or user accounts that are considered deleted from the HSM in the current state.

Recommended Action

To address this issue, one potential option to consider could be to re-encrypt sensitive information after a mode switch, but this could be a major change. Furthermore, attackers could still restore an old device state (with all "old" keys still in place). It should be evaluated if the platform offers facilities that allow to securely erase sensitive data.

During discussion with Nitrokey, it was found that this issue should be re-evaluated once hardware-based security measures are in place. It has been added to this report in order to keep track of the potential problem in the future.

4.6 Changing the Unlock Passphrase without Knowing It

Summary

Type	Location	Exploitability	Attack Impact	Status
Code	Keyfender	n/a	n/a	To be re-evaluated

Technical Description

While reviewing the mechanism for changing the unlock passphrase, it was found that the unlock passphrase can be changed by an administrator without proving knowledge about the current unlock passphrase. This can be problematic in situations where an administrative account is compromised, but the attacker does not directly have access to the unlock passphrase (which might for example only be known to a limited set of persons who memorize it). By changing the unlock passphrase to a known value, the attacker might be able to obtain access to the stored key material if they have physical access to the system.

The same weakness affects the backup passphrase. While this secret is at first sight considered less critical than the unlock passphrase, knowledge of it, in combination with access to a backup file, places an attacker in the same, or even better, situation compared to having physical access to the HSM's storage medium. Obviously, both preconditions can be achieved when control over an administrative account was gained. An administrator always has access to the backup endpoint and, again, can change the relevant passphrase without knowledge of the previous value.

A similar reasoning applies to setting the device to unattended mode without knowledge of the unlock passphrase. This as well considerably reduces the general security of the HSM and allows an attacker to circumvent the storage protection at least partially.

Recommended Action

To address this issue it is advised to require the current unlock passphrase for both enabling the unattended boot mode and for setting a new one. The same procedure should be applied for the backup passphrase.

While discussing this issue with Nitrokey, it has been found that under the current security assumption that an administrative user can fully compromise the HSM, this issue is not relevant. As described in the NetHSM documentation, the administrative user is deliberately the super user. This is required to ensure the availability of the NetHSM system and the stored keys. The implemented role and access model allows organisations the separation of duties by assigning other users restricted access rights.

However, it has been added to this report in order to be able to re-evaluate possible issues once hardware-based measures have been added to the solution.

4.7 Rate Limiting Restrictions

Summary

Type	Location	Exploitability	Attack Impact	Status
Code	Rate Limiting	Low	Medium	Closed

Technical Description

The review of the rate limiting implementation revealed that the login rate limit implementation is essentially based on the remote host, which performs the respective login attempts. Whenever a login succeeds, the counter is reset. This means that an attacker who already has access to one user account (e.g., an operator account or a metrics account) could use this account to log in between brute-force attempts in order to reset the counter.

Recommended Action

To address this issue the rate limiting could be applied on a per account basis. While this opens up the ability of Denial-of-Service attacks against selected accounts, it should be considered whether disconnecting the HSM from the network is a reasonable measure in a scenario were it is considered under attack, anyway. Successful authentications to the corresponding account should not reset the rate limit counter.

Retest Status

This issue has been addressed by re-working the rate limiting scheme. The new implementation considers both, the remote host and the account name. Hence, attackers with access to one user account can no longer reset their host-based login counter.

4.8 Use of PKCS#1v1.5 Decryption

Summary

Type	Location	Exploitability	Attack Impact	Status
Design	Keyfender	Low	Low	Closed

Technical Description

The PKCS#1v1.5 decryption code of the HSM contains an error handling path, which will be triggered if a cipher text has been submitted for decryption, which results in an invalid PKCS#1v1.5 padding. In this case, the user will be informed about the error and the process is aborted without actually providing data to the user. This however directly implies the ability to distinguish between valid and invalid paddings when submitting tampered cipher texts. This directly leads to a possible Bleichenbacher attack, which an attacker could leverage in order to perform arbitrary operations with the respective private key.

In other settings (such as TLS, where the user is not actually supposed to be able to decrypt data with the private key), this would be a severe problem. In the analyzed HSM case, this is different insofar as the user has the legitimate ability to decrypt data. However, it should be pointed out that an attacker might abuse this issue in order to bypass the key usage restrictions of the HSM: instead of requesting a signature operation (which would be forbidden), they instead exploit the padding oracle in order to "decrypt" an arbitrary input (which is carefully crafted, so that the decryption operation actually is a signature computation).

Furthermore, it should be noted that the use of PKCS#1v1.5 padding in general is not advisable, as similar attack scenarios appear to be likely. The combination of allowing PKCS#1v1.5 padding and enforcing key usage restrictions is not trivial to achieve.

Enforcing key usage restrictions is not one of NetHSM's primary security goals. Hence, the impact of this issue is rated low.

Recommended Action

There are a number of possible solution approaches, ranging from removing support for PKCS#1v1.5 over restricting PKCS#1v1.5 to be usable only for keys that have the usage flag `SignAndDecrypt` over adjusting the key usage model entirely.

Retest Status

The original implementation of the key usage restriction focused on the idea of allowing or denying certain operations like signing or decrypting. This concept has been reworked. The key usage restriction mechanism now focuses on allowing or denying particular cryptographic operations, such as PKCS#1v1.5 signatures or PKCS#1v1.5 decryption. This allows users to make a more

conscious decision about what a key may be used for, including possible risks of allowing keys to be used for legacy schemes.

4.9 Missing Range Checks in ECDSA Implementation

Summary

Type	Location	Exploitability	Attack Impact	Status
Design	Mirage-Crypto	Low	Low	Closed

Technical Description

While reviewing the ECDSA support added to the solution, it was found that one of the used libraries, mirage-crypto[6], did not properly check the length of messages to be signed when using ECDSA. This can enable an attacker to provide two different messages `X` and `Y`, which both produce the same signature value. Please be aware that in practical cases where a hash value is provided as "message", this situation is likely not trivial to exploit.

Recommended Action

In order to reduce the attack surface of the implementation, adding a range check for the message to be signed is recommended, in order to make sure that the message length does not exceed the length of the curve's group order.

Retest Status

The issue has been addressed in commit `3e470d0d7e3870769af9b7ce0544e4a13d53c619` of mirage-crypto.

6 https://github.com/mirage/mirage-crypto

5 Additional Observations

Secfault Security would like to point out a number of general observations and recommendations regarding the analyzed system in the following subsections.

5.1 Password Policies

The validation routines for setting passphrases involved in the different HSM use-cases were found to enforce a length of at least 10 characters, without further restrictions. While brute-forcing passphrases of user accounts is primarily of interest in an online scenario attacking the REST API, which can be mitigated by the implementation of brute-force protection measures, the backup and unlock passphrases should be able to withstand offline guessing attacks. In the first case, the passphrase should protect exported backup files at rest, whereas the unlock passphrase generally provides confidentiality to both the HSM contents and the backup files, assuming attended boot mode to be present. Based on the final context the HSM is commonly used in, it should be considered how the offline guessing attacks can be impeded.

Nitrokey stated that the choice of passphrases is not part of NetHSM's responsibilities. Enforcement of password policies and user guidance is planned to be realized as part of an associated UI to be implemented in future.

5.2 Plain Text Storage

The HSM's general concept is based on the encryption of sensitive data when stored in non volatile memory. The decryption uses a key that is preferably (as implemented in attended boot mode) derived from a user-provided passphrase. For completeness, two exceptions from this rule are listed here.

The backup key is stored in plain text as part of the HSM's configuration. The impact on the solution's security, however, is limited, as the key is only used to protect exported backup files. An attacker with access to the non volatile memory of the HSM already has access to the contents of the backup files after decrypting those by means of the backup key. Accordingly, an attacker could only gain advantage from this knowledge when old backup files contain data of interest that is no longer stored in the HSM. Please refer to issue 4.5 for the current feasibility of such a situation.

The same storage condition holds true for the TLS private key used to authenticate the REST API Web server. While this poses a considerable risk for the HSM security, it is not listed as a separate issue as Nitrokey announced to store this key on a connected smart card in the final product. Accordingly, the private key storage is not considered to be part of the scope for the current assessment.

6 Vulnerability Rating

This section provides a description of the vulnerability rating scheme used in this document. Each finding is rated by its type and its exploitability/impact of a successful exploitation. The meaning of the individual ratings are provided in the following sub-sections.

6.1 Vulnerability Types

Vulnerabilities are rated by the types described in the following table.

Type	Description
Configuration	The finding is a configuration issue
Design	The finding is the result of a design decision
Code	The finding is caused by a coding mistake
Observation	The finding is an observation, which does not necessarily have a direct impact

6.2 Exploitability and Impact

The exploitability of a vulnerability describes the required skill level of an attacker as well as the required resources. Therefore, it provides an indication of the likelihood of exploitation.

Exploitability Rating	Description
Not Exploitable	This finding can most likely not be exploited.
Minimal	Although an attack is theoretically possible, it is extremely unlikely that an attacker will exploit the identified vulnerability.
Low	Exploiting the vulnerability requires the skill-level of an expert. An attack is possible, but difficult pre-conditions (e.g., prior identification and exploitation of other vulnerabilities) exist or the attack requires resources not available to the general public (e.g., expensive equipment). Successful exploitation indicates a dedicated, targeted attack.
Medium	The vulnerability can be exploited under certain pre-conditions (e.g., user interaction or prior authentication). Non-targeted, random attacks are possible for attackers with a medium skill level who perform such attacks on a regular basis.
High	The vulnerability can be exploited immediately without special pre-conditions, by random attackers or in an automated fashion. Only general knowledge about vulnerability exploitation is required.

The following table describes the impact rating used in this document.

Impact Rating	Description
Critical	The vulnerability is a systematic error or it permits compromising the system completely and beyond the scope of the assessment.

High	The vulnerability permits compromising the systems within the scope completely.
Medium	The vulnerability exceeds certain security rules, but does not lead to a full compromise (e.g., Denial of Service attacks)
Low	The vulnerability has no direct security consequences but provides information which can be used for subsequent attacks.
Informational	The observed finding does not have any direct security consequence; however, addressing the finding can lead to an increase in security or quality of the system in scope.

When rating the impact of a vulnerability, the rating is always performed based on the scope of the analysis. For example, a vulnerability with high impact typically allows an attacker to fully compromise one or all of the core security guarantees of the components in scope. Identical vulnerabilities can therefore be rated differently in different projects.

7 Glossary

Term	Definition
API	Application Programming Interface
DoS	Denial Of Service
ECDSA	Elliptic Curve DSA
HMAC	Keyed-Hash Message Authentication Code
HSM	Hardware Security Module
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
ID	Identification
REST	Representational state transfer
RSA	Asymmetric Cryptosystem named after Rivest, Shamir and Adleman
TLS	Transport Layer Security
TPM	Trusted Platform Module
URI	Uniform Resource Identifier