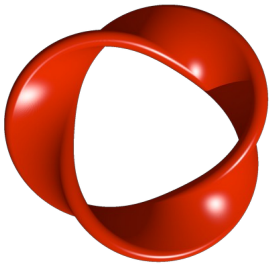

Secfault Security

Review Password Manager

Security Assessment

Report

for

Tether Holdings S.A.

Condominio Presidente Plaza, Level 12, Office 02
San Salvador

- hereafter called "Tether" -

Document History

Version	Author	Date	Comment
0.1	Maik Münch	2025-06-26	First Draft
0.2	Gregor Kopf	2025-07-01	Internal Review
0.3	Gregor Kopf	2025-07-02	Adjustments after Customer Feedback
0.4	Gregor Kopf	2025-07-10	Additional Changes after Customer Review
0.5	Maik Münch	2025-07-28	Retest
0.6	Maik Münch	2025-08-20	Customer Feedback
1.0	Maik Münch	2025-08-21	Final Version

Table of Contents

1 Executive Summary

Tether is building a decentralized, P2P-based software stack suitable for various applications. One of such applications is a password manager solution, which aims to provide secure means of storing, managing and sharing passwords in a decentralized environment.

Tether requested a review of this solution, in order to strengthen its overall security and to identify possible weaknesses in the implementation itself or in the used libraries.

The review has been performed in the time frame from 2025-06-23 to 2025-06-27 in ten person days including documentation. This document describes the results of the project.

The scope of the project is described in detail in section 2.1.1 of this document. Section 2.1.2 provides a description of the test procedures.

During the review three issues, which are described in detail in section 4 of this document, were identified. The most severe issue allowed to store partially controlled data in an arbitrary directory on Windows systems. Further issues included incomplete data validation during import of external data that rendered the solution unusable or a weak origin check in the `autopassHttpServer` that potentially allowed arbitrary browser extensions to communicate with it. All identified issues were resolved promptly by Tether and the respective fixes were confirmed to be effective in an additional review.

In general the solution was designed in a way that mitigated common vulnerability classes and the general design left a positive impression on the testers. The application was designed with security in mind and focused on robustness. This was achieved by strictly adhering to best practices and security principles.

Additionally, it should be mentioned that this review could not cover all of the underlying Pear-runtime given its complexity and the amount of source code. Therefore, this test was performed following a best effort approach incorporating the testers' prior experiences and domain knowledge.

In the time frame from 2025-07-25 to 2025-07-28 a retest of the identified issues was conducted and all identified issues were evaluated to be fixed successfully by Tether. Additionally, a cursory inspection of the custom browser extension was performed that lead to the identification of a minor observation in the IPC handling and is documented in section 5.

2 Overview

The following sections provide an overview of the project execution, the scope of the assessment as well as a brief summary of the test procedures applied during the engagement.

2.1 Project Description

Tether tasked Secfault Security to perform an security review of a custom password manager called PearPass. PearPass is a distributed password manager build on top of the Pear-runtime and allows secure storage of passwords and other sensitive data with the ability to distribute its data across multiple devices.

The source code of PearPass and related dependencies were provided by Tether as PearPass is planned to be developed as open source software.

2.1.1 Target Scope

For testing, Tether provided source code for a number of software components. The entire codebase (including the P2P aspects) is rather large and could not be fully reviewed within the project's timeframe. The analysis therefore focused on the Pearpass application itself, as well as on the code parts that are used by the Pearpass application, in a top-down fashion.

In particular following packages were subject to a manual source code audit:

- `autobase-main`
- `autopass`
- `blind-pairing-main`
- `blind-pairing-core-main`
- `corestore`
- `pear-apps-utils-validator-main`
- `pearpass-app-desktop-main`
- `pearpass-lib-data-import-main`
- `pearpass-lib-vault-desktop-main`
- `pearpass-lib-vault-main`

The following applications were in scope of this project:

- Linux/Windows/MacOS Apps
- Android/iOS Apps
- Chrome Browser Extension

Additionally, Tether provided a key to a deployed PearPass instance.

2.1.2 Test Procedures

The engagement was performed following a white-box methodology. This means that Tether provided full details about the target solution to the consultants beforehand. This methodology generally yields higher-quality results, as it significantly reduces the amount of uncertainty and guesswork about the target system.

The source code review has been performed in a manual fashion, i.e., without relying on automated vulnerability scanners or similar tools. Besides identifying possible classical implementation weaknesses, one main focus of the review was the identification of potential logic problems. This requires an in-depth understanding of the solution's inner workings, which is best achieved by a manual process.

The dynamic tests have been performed in a targeted fashion. On the one hand, this served the purpose of validating issues identified during the source code review. On the other hand, dynamic tests were also performed to obtain a better understanding of the overall solution and the interplay of its individual components.

In a first step, the solution was dynamically tested in order to gain a better overview of its features and to identify first interesting areas.

Following this the source code was inspected briefly to map interesting areas to certain software components and packages.

As one of major focus areas, the cryptographic primitives were subject to a thorough review. This included encryption best practices as well as potential logical issues.

Additionally, the file-related facilities were reviewed for common issues such as directory traversals or insufficient validation of imported data.

Frontend-related source code was analysed with common vulnerability classes such as Cross-Site-Scripting (XSS) in mind.

Additionally, the authentication and authorization mechanisms used by the `autopassHttpServer` was reviewed as well as the implemented desktop application vault client.

Other interesting areas that were briefly inspected included for example:

- Identifier generation
- Sharing/Invitation using the Pear-runtime's blind-pairing implementation

2.2 Project Execution

The project has been executed in the time frame from 2025-06-23 to 2025-06-27 in ten person days.

The consultants assigned to this projects were:

- Maik Münch

- Gregor Kopf

2.2.1 Retest

A retest of the identified issues has been executed in the time frame from 2025-07-25 to 2025-07-28 in two person days.

The consultants assigned to the retest were:

- Maik Münch

3 Result Overview

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

Description	Chapter	Type	Exploitability	Attack Impact
Directory Traversal in Invite Handling	4.1	Code	Medium	High
DoS in Import Functionality	4.2	Code	Medium	Medium
Weak CORS-Origin Check	4.3	Observation	Low	N/A

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.

All of the identified issues have been mitigated by Tether by the time of this documentation.

4.1 Directory Traversal in Invite Handling

Summary

Type	Location	Exploitability	Attack Impact
Code	pearpass-lib-vault-desktop/src/vaultClient/client.js	Medium	High

Technical Description

During static analysis it was identified that during the handling of invite codes only the character `/` was considered as a potential path delimiter.

However, on systems such as Windows different delimiters, e.g., the `\` character can be used as path delimiters.

The following excerpt shows the function responsible for pairing using invite codes:

```
  async pair(inviteCode) {
    this._logger.log('Pairing with invite code:', inviteCode)

    const [vaultId, inviteKey] = inviteCode.split('/')

    const encryptionKey = await this._pairActiveVaultInstance(
      vaultId,
      inviteKey
    )

    this._logger.log('Paired with invite code:', { inviteCode, encryptionKey })

    return { vaultId, encryptionKey }
  }
```

The function splits the passed `inviteCode` into a `vaultId` and an `inviteKey` using `/` as a delimiter. Finally, these two variables are passed into the function `_pairActiveVaultInstance()`, which is

shown in the following excerpt:

```
  async _pairActiveVaultInstance(vaultId, inviteKey) {
  ...

    const encryptionKey = await this._pairInstance(
      `vault/${vaultId}`,
      inviteKey
    )

    return encryptionKey
  }
```

This combines the hard-coded path `vault/` with the `vaultId` using string interpolation and passes it together with the `inviteKey` to `_pairInstance()` trying to limit the given vault path to be in a directory called `vault` in the current working directory. `_pairInstance()` then builds a `fullPath` and uses it to initialize a `Corestore` as shown in the following code excerpt:

```
  _buildPath(path) {
    if (!this.storagePath) {
      throw new Error('Storage path not set')
    }

    return this.storagePath + '/' + path
  }

...

  async _pairInstance(path, invite) {
    const fullPath = this._buildPath(path)

    const store = new Corestore(fullPath)

    ...
  }
```

It can be observed that the resulting code path is not considering different path delimiters such as `\`.

An attacker can now trick a user into using an invite code such as `..\..\..\..\..\..\..\..\..\..\..\..\JRN/LOL` to traversal the file system and create arbitrary directories on a Windows system as the resulting path to store the `Corestore` would be `this.storagePath + '/' + ..\..\..\..\..\..\..\..\..\JRN`, which results in a `Corestore` created at the path `C:\JRN`.

If the attacker is able to control the data partially, such file write issues often lead to a compromise of the host system.

Recommended Action

In order to address this issue it is advised to use library functionality that is platform-agnostic to build file system paths and to ensure that the resulting path is located within the designated `storagePath`. Using custom string interpolation and concatenation often results in edge cases being missed and should be avoided.

Reproduction Steps

To reproduce this issue please start PearPass and enter the Master Password. Then click on "Load Vault" and enter the following invite code `..\..\..\..\..\..\..\..\..\..\..\JRN/LOL` as shown in image Figure 1.

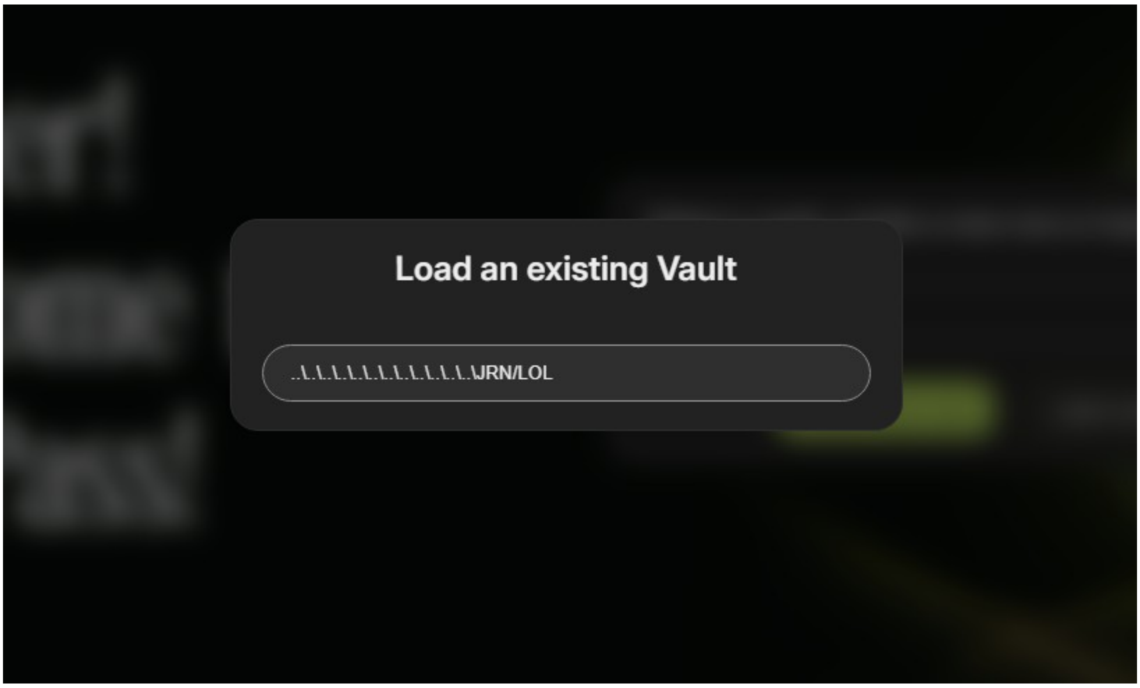

Figure 1 - Loading a vault using an invite code

Then please check using the File Explorer that a directory called `C:\JRN` was indeed created as depicted in the image Figure 2.

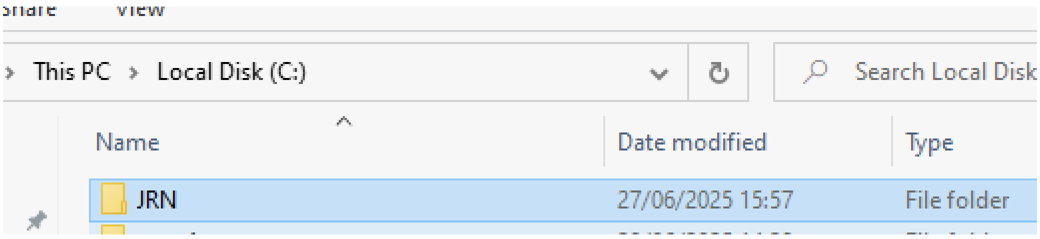

Figure 2 - Resulting directory C:\JRN

Retest Status

This issue could not be reproduced dynamically during the retest. Static analysis showed, that Tether implemented a validation of the invite code that ensures that no meta characters are included. Further, platform-agnostic library functionality was utilized to construct file system paths. Secfault Security, therefore, considers this issue fixed.

4.2 DoS in Import Functionality

Summary

Type	Location	Exploitability	Attack Impact
Code	pearpass-lib-vault/src/utils/validateAndPrepareRecord.js	Medium	Medium

Technical Description

During dynamic analysis of the import functionality it was revealed, that the validation of the imported data is incomplete and allows for example the `type` field to contain arbitrary values such as `foobar` as shown in the following record:

```
[
  {
    "id": "foo",
    "version": 1,
    "type": "foobar",
    "vaultId": "bar",
    "data": {
      "title": "bla",
      "note": "foo",
      "customFields": []
    },
    "folder": null,
    "isFavorite": false,
    "createdAt": 1750166018706,
    "updatedAt": 1750166018706,
    "vaultName": "test"
  }
```

When importing such data the PearPass instance is rendered unusable. This might be exploited by attackers by tricking users into importing attacker controlled data and thus compromising the solutions availability potentially resulting in a loss of trust by users if data cannot be recovered.

The validation function `validateAndPrepareRecord()` in the file `pearpass-lib-vault-main/src/utils/validateAndPrepareRecord.js` only validates well known record types and simply ignores if the `type` is unknown, assumingly leading to a confusion when internally working with a record that uses an unknown record type.

Recommended Action

In order to address this issue, it should be ensured that the imported records are only allowed to be

one of the well-known types. Records using unknown types should be discarded or handled using a catch-all type.

Reproduction Steps

For the reproduction of this issue please use the following steps. Please be aware that this might corrupt your PearPass instance potentially leading to a loss of data:

- First, please start PearPass, enter the Master Password and open or create a vault.
- Please create a Login record and save it.
- Now, under settings, please export this vault to a JSON file.
- Edit the exported JSON file and change the "type" field of the created record to e.g., "UNKNOWN" and save the file.
- Now, please import the altered JSON file.
- The application is now not responsible and only shows a black screen as depicted in image Figure 3.

Please note, that a restart of the application does not resolve this issue and the application is still unavailable.

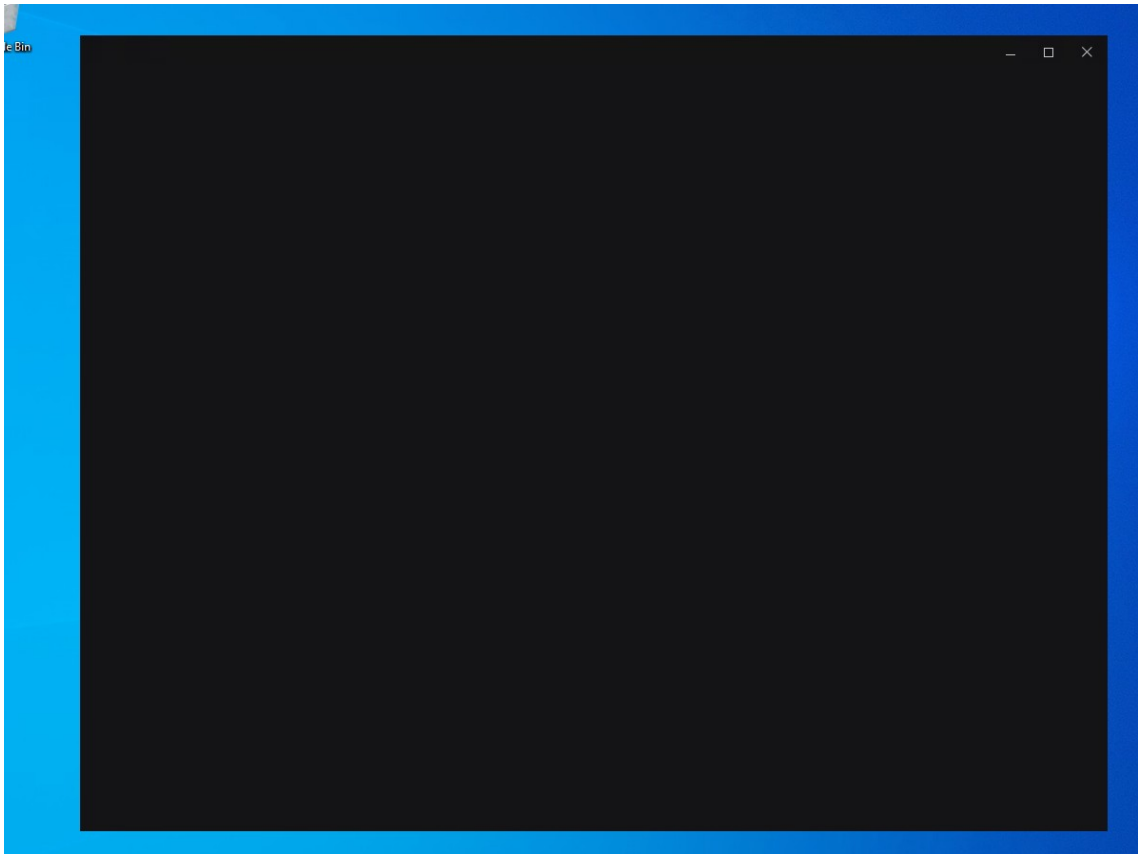

Figure 3 - Unresponsive Application

Retest Status

Tether addressed this finding by implementing further validation of the imported data in form of an allow list for record types. Additionally, this finding could not be reproduced dynamically. Therefore, this issue is considered to be fixed.

4.3 Weak CORS-Origin Check

Summary

Type	Location	Exploitability	Attack Impact
Observation	pearpass-lib-vault-desktop-main/src/server/autopassHttpServer.js	Low	N/A

Technical Description

The cursory review of the `autopassHttpServer` revealed, that the verification of the origin of a request might be incomplete. The following excerpt illustrates the concern:

```
  async _onRequest(req, res) {
    const { pathname, query } = url.parse(req.url, true)

    try {
      const origin = req.headers.origin || ''

      if (origin.startsWith('chrome-extension://')) {
        res.setHeader('Access-Control-Allow-Origin', origin)
        res.setHeader('Access-Control-Allow-Credentials', 'false')
        res.setHeader('Access-Control-Allow-Methods', 'GET,POST,DELETE')
        res.setHeader(
          'Access-Control-Allow-Headers',
          'Content-Type,Authorization'
        )
      } else {
        logger.log(`Blocked request from unauthorized origin: ${origin}`)
      }
  ...
  }
```

The request's `Origin` -Header is checked to block unauthorized origins and only allow requests that start with the schema `chrome-extension://`. However, this might allow arbitrary extensions to communicate with the `autopassHttpServer`.

A malicious extension might request unauthenticated endpoints such as `/vaults/init`, which might result into resource allocation impacting the operation of the host system.

Recommended Action

In order to address this issue it is advised to evaluate if this is wanted behavior and possibly to only allow a limited number of well known origins using an allow list approach.

Reproduction Steps

As this observation was identified during static code analysis, please refer to the details section for reproduce this issue.

Retest Status

This issue was addressed by replacing the former HTTP-based approach to Native Messaging and requiring the user to enable browser extension support and explicitly providing an extension identifier. Thus, Secfault Security considers this issue to be fixed.

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 Local Denial-of-Service via IPC

During static analysis of the browser extension and the implemented native messaging it was identified that a Unix domain socket is used to bridge the browser extension with the desktop application.

It appears that no form of authentication was implemented. This enables local attackers or malicious software to communicate with the socket and interact with PearPass and potentially exploit the exposed methods to gain sensitive information.

During brief dynamic testing it was identified for example that the following script can be used to communicate with the exposed Unix domain socket and after executing the script, the PearPass instance cannot be used anymore presumably due to locking issues of the underlying database:

```
import IPC from 'pear-ipc'
const COMMAND_DEFINITIONS = [
  // Encryption commands
  { id: 1001, name: 'encryptionInit' },
  { id: 1002, name: 'encryptionGetStatus' },
  { id: 1003, name: 'encryptionGet' },
  { id: 1004, name: 'encryptionAdd' },

  // Vaults commands
  { id: 1005, name: 'vaultsInit' },
  { id: 1006, name: 'vaultsGetStatus' },
  { id: 1007, name: 'vaultsGet' },
  { id: 1008, name: 'vaultsList' },
  { id: 1009, name: 'vaultsAdd' },
  { id: 1010, name: 'vaultsClose' },

  // Active vault commands
  { id: 1011, name: 'activeVaultInit' },
  { id: 1012, name: 'activeVaultGetStatus' },
  { id: 1013, name: 'activeVaultGet' },
  { id: 1014, name: 'activeVaultList' },
  { id: 1015, name: 'activeVaultAdd' },
  { id: 1016, name: 'activeVaultRemove' },
  { id: 1017, name: 'activeVaultClose' },
  { id: 1018, name: 'activeVaultCreateInvite' },
  { id: 1019, name: 'activeVaultDeleteInvite' },
```

```
  // Password and encryption key commands
  { id: 1020, name: 'hashPassword' },
  { id: 1021, name: 'encryptVaultKeyWithHashedPassword' },
  { id: 1022, name: 'encryptVaultWithKey' },
  { id: 1023, name: 'getDecryptionKey' },
  { id: 1024, name: 'decryptVaultKey' },

  // Pairing and misc commands
  { id: 1025, name: 'pair' },
  { id: 1026, name: 'initListener' },
  { id: 1027, name: 'closeVault' }
]

const c = new IPC.Client({
  socketPath: '/tmp/pearpass-native-messaging.sock',
  connect: true,
  methods: COMMAND_DEFINITIONS
})

const timeoutPromise = new Promise((_, reject) => {
setTimeout(() => {
  reject(new Error('Connection timed out'))
}, 1000)
})

await Promise.race([c.ready(), timeoutPromise])

console.log('IPC-BRIDGE', 'INFO', 'Successfully connected to IPC server')

async function cal(methodName, cleanParams) {
  let result = null
  try {
    result = await Promise.race([ c[methodName](cleanParams), timeoutPromise ])
  } catch (error) {
    console.log( 'IPC-BRIDGE', error)
  }
  console.log(result)
}

await cal('encryptionInit', {})
await cal('vaultsInit', {})
await cal('vaultsList', {})
await cal('vaultsAdd', {
    id: 'my-vault2',
    name: 'My Password Vault',
    hashedPassword:
'b623e2189004471791f5093c65fba661e9248d7ca12436ca2af88b1237ff65db',
    ciphertext: 'asease',
```

```
    nonce: '213',
    salt: 'H8rsFozmE7TZomM79C/VeQ==',
});
await cal('hashPassword', {password: 'AAA'})
```

A user has to restart PearPass to be able to successfully unlock their vaults again.

Given that the Unix domain socket is available even without enabling browser extension support, Secfault Security wanted to include this observation in this document although no sensitive information were leaked and the socket is only writable by the current user.

Further, it should be evaluated if the Unix domain socket should be available when browser extension support is disabled to minimize potential attack surface.

At the time of documenting this observation, Tether was already working on a mitigation.

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.

Impact Rating	Description
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
HTTP	Hyper Text Transfer Protocol
IPC	Inter-Process Communication
JSON	JavaScript Object Notation
P2P	Peer-To-Peer
XSS	Cross Site Scripting