Pendzl Smart Contracts Security Code Review

Technical Report

Abax Finance

19 July 2024
Version: 2.0

Kudelski Security – Nagravision Sàrl

Corporate Headquarters
Kudelski Security – Nagravision Sàrl
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

For Public Release

DOCUMENT PROPERTIES

Version:	2.0
File Name:	Kudelski_Security_Abax_Finance_Pendzl_Secure_Code_Review_2.0.pdf
Publication Date:	19 July 2024
Confidentiality Level:	For Public Release
Document Status:	Approved

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY 4
2. PROJECT SUMMARY 6
    2.1 Context 6
    2.2 Scope 6
    2.3 Remarks 6
    2.4 Additional Note 7
3. TECHNICAL DETAILS OF SECURITY FINDINGS 8
    3.1 KS–PNZ–F–1 Potential Underflow in Balance Update 9
    3.2 KS–PNZ–F–2 Potential Integer Overflow in `_decimals_offset` Usage 10
4. OBSERVATIONS 11
    4.1 KS–PNZ–O–1 Zero-Address Not Checked 12
    4.2 KS–PNZ–O–2 TODO Still Present in the Code 12
    4.3 KS–PNZ–O–3 Subtractions Not Performed With the Function `checked_sub` 12
    4.4 KS–PNZ–O–4 Lack of Access Control in Pausable System 13
    4.5 KS–PNZ–O–5 `ink::env::debug_println!` Still Present in The Code 13
    4.6 KS–PNZ–O–6 Potential Reentrancy Vulnerability 13
    4.7 KS–PNZ–O–7 Lack of Functionality to Revoke Allowances in PSP22 Contract 14
    4.8 KS–PNZ–O–8 Lack of Input Validation in Ownable Library 15
    4.9 KS–PNZ–O–9 Lack of Input Validation in `PSP22` Library 15
5. METHODOLOGY 16
    5.1 Kickoff 16
    5.2 Ramp-up 16
    5.3 Review 16
    5.4 Smart Contracts 17
    5.5 Reporting 17
    5.6 Verify 17
6. VULNERABILITY SCORING SYSTEM 18
7. CONCLUSION 20
DOCUMENT RECIPIENTS 21
KUDELSKI SECURITY CONTACTS 21
DOCUMENT HISTORY 21

1. EXECUTIVE SUMMARY

Abax Finance (“the Client”) engaged Kudelski Security (“Kudelski”, “we”) to perform to perform a Secure Code Review of library Pendzl which aims to provide standard for ink! smart contract implementation.

The assessment was conducted remotely by the Kudelski Security Team.

The review took place between 08 April 2024 and 21 June 2024, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

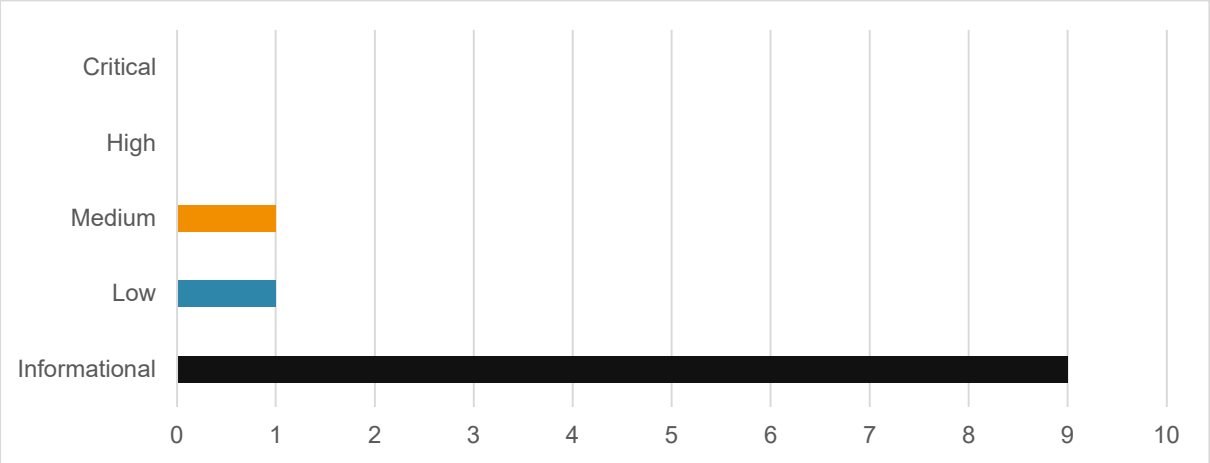
A second review was conducted between 12 July 2024 and 18 July 2024 after the modifications performed by the Client.

Key Findings

The following are the major themes and issues identified during the testing period.

These, along with other items, within the findings section, should be prioritized for remediation to reduce to the risk they pose.

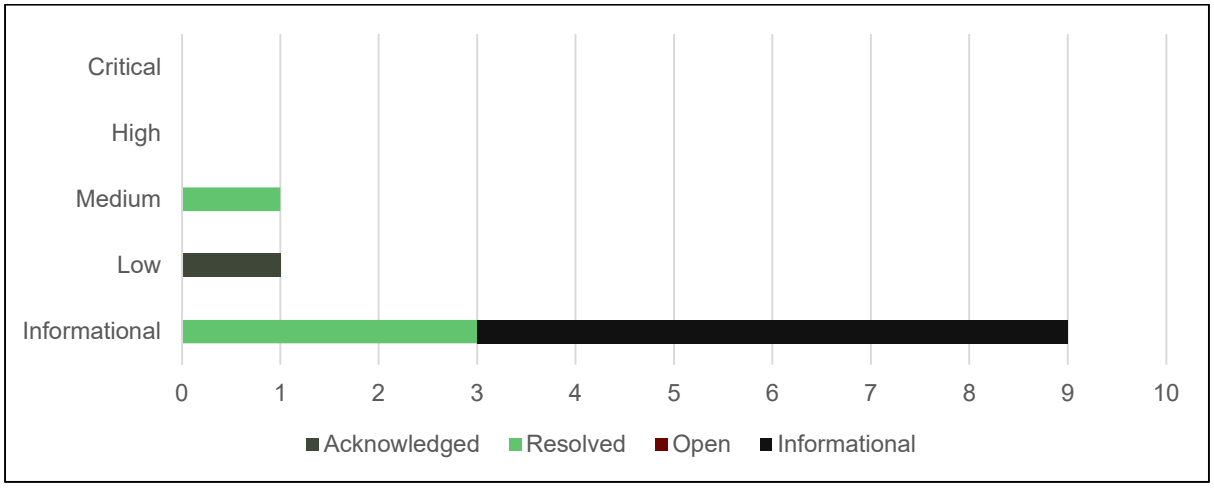
- Potential Underflow in Balance Update
- Lack of Input validation in ownable library
- Lack of Input validation in PSP22 Library



Findings ranked by severity.

Status of Findings after Re-review

A second review was conducted after the modifications performed by the Client. A finding is set as resolved if the Client did modify the implementation and we considered the fix to be correct. A finding is set to acknowledged if either the Client decided to accept the risk, or the mitigation is difficult to impossible to implement with the existing tools and resources. Finally, a finding is set as open if nothing was implemented or communicated by the client between the two code reviews.



Overview of findings and their status

2. PROJECT SUMMARY

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

2.1 Context

Pendzl is a library for smart contract development. It provides standard contracts for Polkadot eco-system. Importantly, this standard library includes the PSP22 standard for non-fungible token and the PSP34 standard for fungible token.

2.2 Scope

The scope consisted in specific ink! files and folders located at:

- Commit hash: `b9333cbb1bc03dff3433ec20e36570b72449c024`
- Source code repository : https://github.com/Nradko/Pendzl.git

The files in scope are all ink! files (.rs) in the following folders:

- `/contracts`
- `/lang`

Follow-up

After the initial report, Abax Finance addressed or acknowledged the vulnerabilities and weaknesses in the following codebase revision:

- Commit hash: `a26a38774f7517df012e944b85744f948e41aaa8`
- Source code repository : https://github.com/Pendzl/pendzl/tree/main

We reviewed the changes between the two commits and updated the status of the findings.

2.3 Remarks

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The developers have made a careful and in-depth analysis of their project.
- Tests were also provided as part of the project, which is convenient for better understanding how the library works and useful for elaborating scenarios and validating findings.
- Finally, we had regular and very enriching technical exchanges on various topics.

2.4 Additional Note

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, along with the scope of the agreement.

In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination. Information about the severity ratings can be found in Chapter Vulnerability Scoring System of this document.

Additionally, as Pendzl is a library aiming to improve standardization of ink! smart contract implementation, we also highlighted some observations of what the library provides or not to the users in terms of security.

3. TECHNICAL DETAILS OF SECURITY FINDINGS

This chapter provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

#	SEVERITY	TITLE	STATUS
KS–PNZ–F–1	Medium	Potential Underflow in Balance Update	Resolved
KS–PNZ–F–2	Low	Potential Integer Overflow in `_decimals_offset` Usage	Acknowledged

Findings overview.

3.1 KS–PNZ–F–1 Potential Underflow in Balance Update

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

The Pendzl library implements the PSP34 standard for non-fungible tokens. In this standard implementation, the owner's balance of a token is decremented without checking if the balance is zero. This operation could potentially lead to an underflow if the balance is already zero. An underflow in this context would mean that the balance becomes a very large number, which could lead to incorrect behavior in the token contract.

Impact

If an underflow occurs, it could lead to serious issues such as incorrect token balances being reported, tokens being minted incorrectly, or other parts of the contract behaving unexpectedly. This could potentially be exploited by an attacker to gain an unfair advantage or disrupt the operation of the contract.

Evidence

```
let balance = self.owned_tokens_count.get(from).unwrap_or(0);
self.owned_tokens_count.insert(from, &(balance - 1));
let total_suply = self.total_supply.get().unwrap();
self.total_supply.set(&(total_suply - 1));
```

`implementation.rs`. Uncheked substaction could result in underflow.

Affected Resources

- `Pendzl/contracts/src/psp34/implementation.rs, lines 104-08`

Recommendation

We recommend using checked arithmetic operations to prevent any risks of underflows. In ink!, you can use the `checked_sub` method, which returns None if the operation would cause an underflow. Additionally, in ink! smart contract development, the overflow/underflow protection is enabled by default, we recommend ink! builder to keep this enable and disable it only if there is not only other option.

References

- [1] `checked_sub function`:

3.2 KS–PNZ–F–2 Potential Integer Overflow in `_decimals_offset` Usage

Severity	Impact	Likelihood	Status
Low	Medium	Low	Acknowledged

Description

The `_decimals_offset` function is used to calculate an offset that is then used as an exponent in a power of 10 operation. If the value returned by `_decimals_offset` is greater than 39, this can lead to an integer overflow. This is because the `u128` type in ink!, which is used to store the result of the power operation, can hold a maximum value of $\mathbf{2^{128} - 1}$, which is approximately $\mathbf{3.4 * 10^{38}}$. This means that If `_decimals_offset` is equal to 40 or more, the result of the power operation will be $\mathbf{10^{40}}$ or greater, which exceeds the maximum value that a `u128` can hold.

Impact

An integer overflow can lead to unexpected behavior, in this case if the value wraps around and start from zero again. This can lead to incorrect token balance calculations, which is especially problematic in a financial or blockchain context where accurate calculations are crucial.

Evidence

```
let decimals_offset = 10_u128.pow(self._decimals_offset() as u32);
…
let decimals_offset = 10_u128.pow(self._decimals_offset() as u32);
```

`implementation.rs`. Unchecked power could result in Overflow.

Affected Resources

- `Pendzl/contracts/src/token/psp22/extensions/vault/implementation.rs, lines 117 and 135`

Recommendation

To mitigate this risk, it is recommended to add a check before the power operation to ensure that the value returned by `_decimals_offset` is not greater than 39. If it is, an error should be returned to prevent the overflow. In ink!, we advise developers to maintain the overflow/underflow protection enable.

It is important to highlight that, the developers added after the initial report a usage comment for the Pendzl library users to prevent this risk.

References

N/A

4. OBSERVATIONS

This chapter contains additional observations that are not directly related to the security of the code, and as such have no severity rating or remediation status summary. These observations are either minor remarks regarding good practice or design choices or related to implementation and performance. These items do not need to be remediated for what concerns security, but where applicable we include recommendations.

#	SEVERITY	TITLE	STATUS
KS–PNZ–O–1	Informational	Zero-Address not Checked	Informational
KS–PNZ–O–2	Informational	TODO Still Present in the Code	Resolved
KS–PNZ–O–3	Informational	Subtractions Not Performed With the Function `checked_sub`	Resolved
KS–PNZ–O–4	Informational	Lack of Access Control in Pausable System	Informational
KS–PNZ–O–5	Informational	`ink::env::debug_println!` Still Present in the Code	Resolved
KS–PNZ–O–6	Informational	Potential Reentrancy Vulnerability	Informational
KS–PNZ–O–7	Informational	Absence of Functionality to Revoke Allowances in PSP22 Token Contract	Informational
KS–PNZ–O–8	Informational	Lack of Input validation in ownable library	Informational
KS–PNZ–O–9	Informational	Lack of Input Validation in `PSP22` Library	Informational

Observations overview.

4.1 KS–PNZ–O–1 Zero-Address Not Checked

Description

We observed that the zero-address verifications were not done, this means that a role can be assigned to the zero-address or token could be transferred to the zero-address, which has a secret key publicly known.

This is a choice was made by the developers who believe that it is responsibility stands to the user to know assigned a role or send token to the zero-address. It is important to highlight that in OpenZeppelin, the standard library for Ethereum smart contracts, the zero-address verification is enabled by default and users can choose to disable it, while in the case of Pendzl the opposite choice was made.

Affected Resources

- This is a general observation for the overall Pendzl library

Recommendation

As this a choice of the Pendzl developers, the only recommendation is to explicitly mention this in the guideline of the standard library.

4.2 KS–PNZ–O–2 TODO Still Present in the Code

Description

The codebase contains several `TODO` comments indicating unfinished tasks or features that need to be implemented. While `TODO` comments can be useful for marking areas of the code that need further work, leaving them unresolved in the production code result in to unexpected behavior or incomplete functionality.

Affected Resources

- `Pendzl/contracts/src/finance/general_vest/general_vest_types.rs` lines 115

Recommendation

We recommend reviewing all TODO comments in the codebase and resolving them as soon as possible. If the tasks they represent are not immediately actionable, consider tracking them in a project management tool or issue tracker instead of leaving them in the code. This will help ensure that all tasks are accounted for and can be properly prioritized and tracked. If a `TODO` is no longer relevant, remove the comment to avoid confusion.

4.3 KS–PNZ–O–3 Subtractions Not Performed With the Function `checked_sub`

Description

There are subtraction operations that are performed without using the `checked_sub` method.

Affected Resources

- `Pendzl/contracts/src/finance/general_vest/implementation.rs, lines 93 and 95`

Recommendation

Even though, there are no risk of underflow in these specific cases, it is still recommended to use the `checked_sub` method for subtraction operations.

4.4 KS–PNZ–O–4 Lack of Access Control in Pausable System

Description

The Pendzl library proposes standard and default methods to pause and unpause a system, specifically `_pause_default_impl` and `_unpause_default_impl`. However, there doesn't appear to be any access control mechanisms in place to restrict who can call these methods. Depending on the context and use case of this system, this could potentially be a security issue. If any account can pause or unpause the system, it could lead to misuse or disruption of the system's intended operation.

Affected Resources

- `Pendzl/contracts/src/security/pausable/implementation.rs`

Recommendation

Implement access control mechanisms to ensure that only authorized accounts can pause or unpause the system. This could be done by adding checks in the `_pause_default_impl` and `_unpause_default_impl` methods to verify the caller's permissions before proceeding with the operation. The specifics of this implementation would depend on the broader context of your application, but it could involve checking if the caller's account is in a list of authorized accounts, or if the caller has a certain role or privilege level. This would help to prevent unauthorized use of these critical operations and enhance the security of your system.

4.5 KS–PNZ–O–5 `ink::env::debug_println!` Still Present in The Code

Description

The Pendzl library implementation code still contains debugging print messages.

Affected Resources

- `src/token/psp22/extensions/vault/implementation.rs`

Recommendation

Debugging messages needs to be suppressed before releasing the code for production.

4.6 KS–PNZ–O–6 Potential Reentrancy Vulnerability

Description

Reentrancy is a vulnerability that occurs when a function can be interrupted during execution and called again before the first call is finished. This can lead to unexpected behavior, such as funds being withdrawn multiple times in a single transaction. In the context of this code, the

`_withdraw_default_impl` and `_deposit_default_impl` functions are potentially vulnerable to reentrancy attacks. Both functions call `self._asset().transfer` or `self._asset().transfer_from` (which are external calls) and then change the state of the contract with `self._mint_to(receiver, shares)` or `self._burn_from(owner, shares)`. If the `transfer` or `transfer_from` functions are compromised, they could call back into `_withdraw_default_impl` or `_deposit_default_impl` and reenter the function before the state changes have been committed.

Affected Resources

- `Pendzl-main/contracts/src/token/psp22/extensions/vault/implementation.rs`

Recommendation

We did not consider this to be a direct security threat as ink! developers need to set the `CallFlags::ALLOW_REENTRY` flag to allow reentry in a smart contract. This flag is not set in the Pendzl library. If a user sets the reentry flag, they need to mitigate the risk of reentry, consider using the Checks-Effects-Interactions pattern, where you perform any external calls or transfers last, after all internal state has been updated.

4.7 KS–PNZ–O–7 Lack of Functionality to Revoke Allowances in PSP22 Contract

Description

The PSP22 token contract includes functions to approve (`_approve_default_impl`), decrease (`_decrease_allowance_from_to_default_impl`), and increase (`_increase_allowance_from_to_default_impl`) allowances. However, it lacks a function to revoke the allowances. User can use `_approve_default_impl` or `_decrease_allowance_from_to_default_impl` to set new amount to 0. However, there should be an option for the developers to set one or all previously approved allowances to zero through a `_revoke_Allowance` or `_revoke_All_Allowances` function.

If a user sets an allowance for a malicious spender and later receives funds into their account, the spender could potentially drain these funds based on the previously set allowance. This could occur even if the user did not intend for the spender to have access to these new funds.

Affected Resources

- `Pendzl-main/contracts/src/token/psp22/implementation.rs, lines 256 and 302`

Recommendation

Add a function to the contract to allow users to revoke allowances. This could be a `_revoke_Allowance` function that sets the allowance for a specific spender to zero. This would give users more control over their allowances and help to prevent unexpected loss of funds. Additionally, consider implementing a `_revoke_All_Allowances` function that sets all allowances to zero for further control and security. Interesting literature related to this subject can be found with the following links:

- Token Allowance
- Revoking Token Allowance.

4.8 KS–PNZ–O–8 Lack of Input Validation in Ownable Library

Description

In the provided Pendzl library, there is folder named ownable, which is a library providing standard approach for ownership operation of a contract, there are several functions that could benefit from input validation to ensure the integrity and security of the contract.

Affected Resources

- `Pendzl-main/contracts/src/access/ownable/implementation.rs, lines 1 and 81`

Recommendation

We recommended to implement input sanitization in these functions by default. For example, OpenZeppelin standard library validates that the provided address is not a zero-address and is not the same as the current owner's address. This would prevent the contract from being initialized with no owner, prevent the contract owner from being set to an invalid address, and prevent unnecessary ownership transfers.

Mature standard libraries for smart contracts standard application, such as OpenZeppelin for solidity, includes some input validation by default, that can disable by users.

4.9 KS–PNZ–O–9 Lack of Input Validation in PSP22 Library

Description

In the provided ink! implementation of the `PSP22` token standard, there are several functions that could benefit from inputs validation to ensure the integrity and security of the contract.

Affected Resources

- `Pendzl-main/contracts/src/token/psp22/implementation.rs, lines 1 and 302`

Recommendation

We recommend adding standard input validation to these functions. For example, this could be done with simple conditional checks at the beginning of each function. If the "to" address is zero or the transfer amount is zero, the function should fail and return an error.

More mature standard smart contract libraries, such as OpenZeppelin for Solidity, include some input validation by default, which can be disabled by the user.

References

- [1] OpenZeppelin ERC20 smart contract

5. METHODOLOGY

For this engagement, Kudelski Security used a methodology that is described at a high level in this chapter. This is broken up into the following phases.

5.1 Kickoff

The Kudelski Security Team set up a kick-off meeting where project stakeholders were gathered to discuss the project as well as the responsibilities of participants. During this meeting, we verified the scope of the engagement and discussed the project activities.

5.2 Ramp-up

Ramp-up consisted of the activities necessary to gain proficiency on the particular project. This included the steps required for gaining familiarity with the codebase and technological innovations utilized.

5.3 Review

The review phase is where most of the work on the engagement was performed. In this phase we have analyzed the project for flaws and issues that could impact the security posture. The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools was used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

Code Review

Kudelski Security Team reviewed the code within the project utilizing an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content, to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- authentication (*e.g.* A07:2021, CWE-306)
- authorization and access control (*e.g.* A01:2021, CWE-862)
- auditing and logging (*e.g.* A09:2021)
- injection and tampering (*e.g.* A03:2021, CWE-20)
- configuration issues (*e.g.* A05:2021, CWE-798)
- logic flaws (*e.g.* A04:2021, CWE-190)
- cryptography (*e.g.* A02:2021)

These categories incorporate common weaknesses and vulnerabilities such as the OWASP Top 10 and MITRE Top 25.

5.4 Smart Contracts

We reviewed the smart contracts, checking for additional specific issues that can arise such as:

- assessment of smart contract admin centralization
- reentrancy attacks and external contracts interactions
- verification of compliance with existing standards such as ERC20 or PSP34
- unsafe arithmetic operations such as overflow and underflow verification dependance on timestamp
- access control verification to ensure that only authorized users can call sensitive functions.

5.5 Reporting

Kudelski Security delivered to Abax Finance a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project.

In the report we not only point out security issues identified but also observations for improvement. The findings are categorized into several buckets, according to their overall severity: **Critical**, **High**, **Medium**, **Low**.

Observations are considered to be **Informational**. Observations can also consist of code review, issues identified during the code review that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

5.6 Verify

After the preliminary findings have been delivered, we verify the fixes applied by Abax Finance. After these fixes were verified, we updated the status of the finding in the report.

The output of this phase is the final report with any mitigated findings noted.

6. VULNERABILITY SCORING SYSTEM

Kudelski Security utilizes a custom approach when computing the vulnerability score, based primarily on the **Impact** of the vulnerability and **Likelihood** of an attack.

Each metric is assigned a ranking of either low, medium or high, based on the criteria defined below. The overall severity score is then computed as described in the next section.

Severity

Severity is the overall score of the finding, weakness or vulnerability as computed from Impact and Likelihood. Other factors, such as availability of tools and exploits, number of instances of the vulnerability and ease of exploitation might also be taken into account when computing the final severity score.

IMPACT / LIKELIHOOD	LOW	MEDIUM	HIGH
HIGH	MEDIUM	HIGH	HIGH
MEDIUM	LOW	MEDIUM	HIGH
LOW	LOW	LOW	MEDIUM

Compute overall severity from Impact and Likelihood. The final severity factor might vary depending on a project's specific context and risk factors.

- **Critical** The identified issue may be immediately exploitable, causing a strong and major negative impact system-wide. They should be urgently remediated or mitigated.
- **High** The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
- **Medium** The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
- **Low** The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
- **Informational** findings are best practice steps that can be used to harden the application and improve processes. Informational findings are not assigned a severity score and are classified as Informational instead.

Impact

The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

- **High** The vulnerability has a severe effect on the company and systems or has an affect within one of the primary areas of concern noted by the client.
- **Medium** It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.
- **Low** There is little to no affect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, difficulty of exploitation and institutional knowledge.

- **High** It is extremely likely that this vulnerability will be discovered and abused.
- **Medium** It is likely that this vulnerability will be discovered and abused by a skilled attacker.
- **Low** It is unlikely that this vulnerability will be discovered or abused when discovered.

7. CONCLUSION

The objective of this Secure Code Review was to evaluate whether there were any vulnerabilities that would put the Pendzl library or its users at risk.

The Kudelski Security Team identified 2 security issues: 1 medium risk and 1 lower risks. On average, the effort needed to mitigate these risks is estimated as low.

In order to mitigate the risks posed by this engagement's findings, the Kudelski Security Team recommends applying the following best practices:

- Use `checked` (for example `checked_sub`) for all arithmetic operations
- Inputs sanitization
- Write explicit documentation about what the Pendzl library provide or not.

Kudelski Security remains at your disposal should you have any questions or need further assistance.

Kudelski Security would like to thank Abax Finance for their trust, help and support over the course of this engagement and is looking forward to cooperating in the future.

DOCUMENT RECIPIENTS

NAME	POSITION	CONTACT INFORMATION
Konrad Wierzbik	Co-founder	konrad.wierzbik@gmail.com
Łukasz Łakomy	Co-founder	lukasz.jan.lakomy@gmail.com

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Jean-Sebastien Nahon	Application and Blockchain Security Practice Manager	jean-sebastien.nahon@kudelskisecurity.com
Ana Acero	Project Manager/ Operations Coordinator	ana.acero@kudelskisecurity.com

DOCUMENT HISTORY

VERSION	DATE	STATUS/ COMMENTS
1.0	21 June 2024	Findings status to Open
1.1	18 July 2024	Re-review (Finding status update)
2.0	19 July 2024	Public release version