Abax DAO 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_DAO_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

Abax Finance ("the Client") engaged Kudelski Security ("Kudelski", "we") to perform a Abax DAO Smart Contracts Security Code Review through the security partnership with Aleph Zero foundation to perform a Secure Code Review of DAO 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.

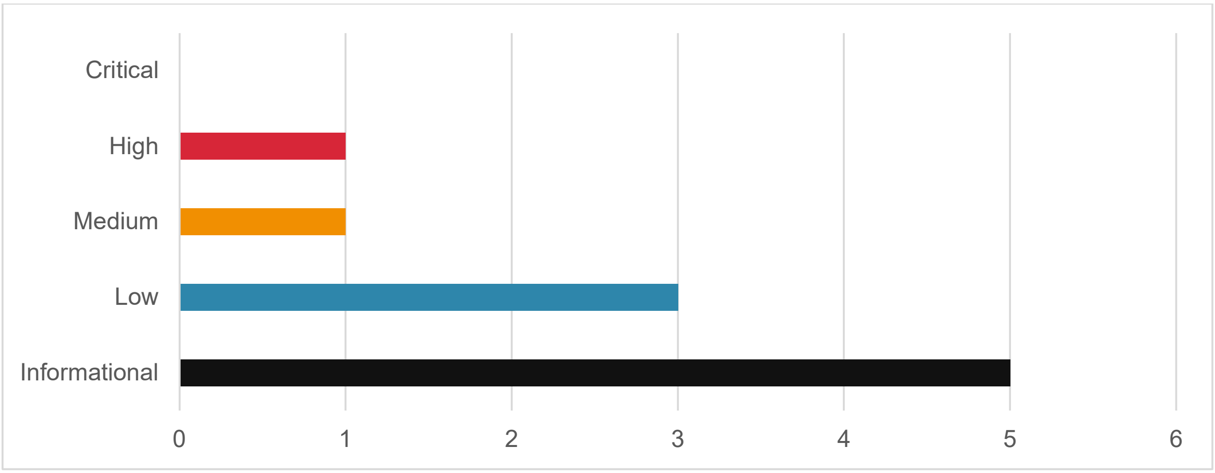
A second review was conducted between 12 July 2024 and 18 July 2024 after the modifications performed by the Client.

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.

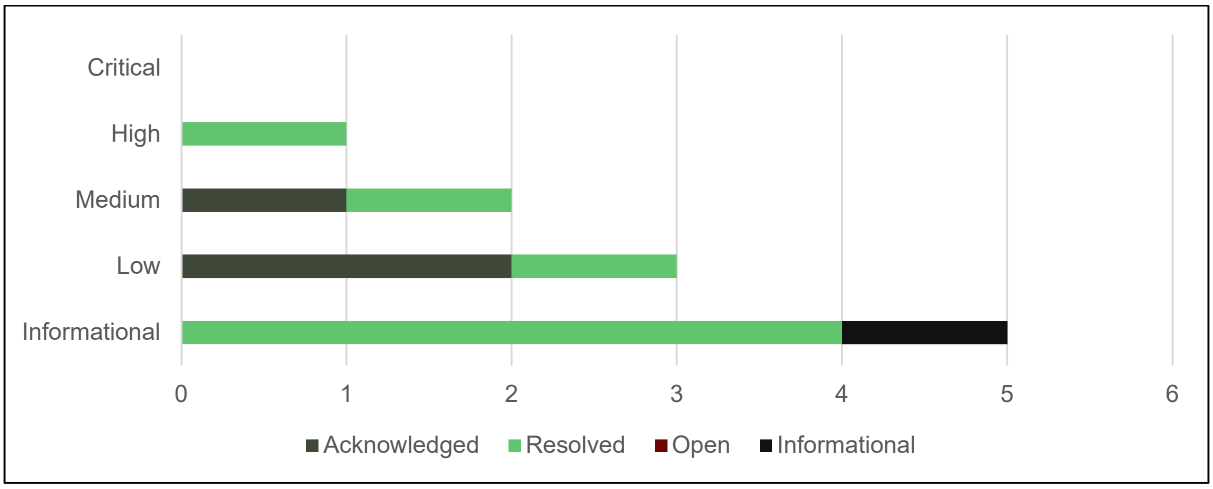
- Double voting
- Risk of centralization



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

The reviewed application is a Decentralized Autonomous Organization (DAO). It is a way of structuring organizations without a traditional management hierarchy, and instead rules are enforced digitally through smart contracts. In theory, this allows for more direct governance by stakeholders and more efficient decision-making processes. Users deposit tokens in exchange for voting rights to decide on the outcomes of proposals.

2.2 Scope

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

- Commit hash: `220d7af92df3b09c545bf89d2e91d62c7de0c7bd`
- Source code repository: https://github.com/AbaxFinance/dao-contracts

The files and folders in scope are the ink! (`.rs`) files located in the `src/` folder of the above repository.

Follow-up

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

- Commit hash: `bcb82e0ae2e8c0fbfbf6f762ab4efd39939337d9`
- Source code repository : https://github.com/AbaxFinance/dao-contracts/tree/audit_prep

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.
- Security seemed to be one of the main focuses of Abax Finance developing team while implementing the 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.

3. TECHNICAL DETAILS OF SECURITY FINDINGS

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

The following table provides an overview of the findings.

#	SEVERITY	TITLE	STATUS
KS–ABXDAO–F–1	High	Potential for Multiple Votes through Withdraw and Deposit	Resolved
KS–ABXDAO–F–2	Medium	Risk of centralization	Acknowledged
KS–ABXDAO–F–3	Low	Lack of Input Validation in `create_order` Function	Acknowledged
KS–ABXDAO–F–4	Low	Risk of underflow	Resolved
KS–ABXDAO–F–5	Low	Lack of Input Validation in deposit & withdraw functions	Acknowledged

Findings overview.

3.1 KS–ABXDAO–F–1 Potential for Multiple Votes through Withdraw and Deposit

Severity	Impact	Likelihood	Status
High	High	High	Resolved

Description

The current implementation of the voting system in the smart contract allows a user to vote multiple times using the same token. Even though the transferring token is not implemented, there is still a way for an attacker to cast more votes than he is entitled to. An attacker could cast a vote with his tokens, then he withdraw them from the Governor smart contract. The next step for the attacker would be to deposit the same token back with another account and cast a new vote using the same token.

The attacker needs to transfer this token to another account and not the same one, the function `update_vote_of_for` remove previous votes before adding new ones.

Impact

This vulnerability allows an attacker to cast more votes than voting power. This could lead to manipulation of the outcome of a vote, undermining the fairness and integrity of the voting system.

Evidence

```
pub fn update_vote_of_for(
        &mut self,
        account: &AccountId,
        proposal_id: &ProposalId,
        vote: &Vote,
        amount: &Balance,
    ) -> Result<(), GovernError> {
…
let existing_user_vote = self.vote_of_for(account, proposal_id);
match existing_user_vote {
…
            Some(old_vote) => match old_vote.vote {
                Vote::Agreed => match vote {
                    Vote::Agreed => {
                        state.votes_for = state
                            .votes_for
```

```
                                    .checked_sub(old_vote.amount)
                                    .ok_or(MathError::Underflow)?;
                                state.votes_for = state
                                    .votes_for
                                    .checked_add(*amount)
                                    .ok_or(MathError::Overflow)?;
                        }
```

`src/contracts/governor/modules/govern/storage/govern_storage_item.rs`. Votes are associated with account and not with token which means that the same token can be used twice to vote with different accounts.

Affected Resources

- `src/contracts/governor/modules/govern/storage/govern_storage_item.rs, lines 273-411`

Recommendation

To prevent this kind of issue, we recommend making withdrawing or depositing not possible during a voting period. This would avoid having the same token being using multiple times to vote on the same proposal.

References

N/A

3.2 KS–ABXDAO–F–2 Risk of Centralization

Severity	Impact	Likelihood	Status
Medium	High	Low	Acknowledged

Description

The DAO implementation from Abax Finance implements different roles that have different type of power. Having a role-based approach in smart contract management is recommended to perform sensitive operations such as modifying contract parameters, pausing the contract, or even upgrading the contract. Nevertheless, having an admin role can pose several risks. For example, the project has a role named `PARAMETERS_ADMIN`, who has the responsibility to change voting rules, which are important in the DAO implementation. This means that a compromised would cause severe damage to the project.

Impact

This centralization of power can lead to several potential security issues.

1. Single Point of Failure If the admin's private key is lost or compromised, it could lead to a loss of control over the contract, potentially causing irreversible damage.
2. Misuse of Power: The admin could potentially act maliciously or make mistakes, leading to actions that could harm users or the system as a whole.
3. Centralization: One of the key benefits of blockchain and smart contracts is decentralization. Having an admin role can contradict this principle, as it centralizes control in the hands of a single entity.

The impact of centralization can be severe, but it requires to break an Aleph Zero account which cannot be done on the smart contract side. This risk can be mitigated as described below.

Evidence

```
pub const EXECUTOR: RoleType = ink::selector_id!("EXECUTOR");
pub const PARAMETERS_ADMIN: RoleType = ink::selector_id!("PARAMETERS_ADMIN");
// 368_001_360_u32
```

`src/contracts/governor.rs`. Example of role used in the DAO implementation

Affected Resources

- This is a general finding valid for the entire project.

Recommendation

To mitigate the risks of role-based smart contract administration, we recommend using a multisig (or multi-signature) account [1] for all accounts that play a critical role in the proper functioning of the smart contract.

Instead of a single admin, use a multisig account where multiple parties must sign off on changes. This would prevent any abuse of power and reduce the risk of a single point of failure.

It is important to know that we had a discussion with Abax Finance development team about multisig accounts and they mentioned that they plan to use these types of accounts for key admin accounts.

References

- [1] Multisig Polkadot

3.3 KS–ABXDAO–F–3 Lack of Input Validation in create_order Function

Severity	Impact	Likelihood	Status
Low	Low	Low	Acknowledged

Description

The `create_order` function in the `abax_treasury` contract does not perform any validation on its inputs. This function takes in parameters `for earliest_execution, latest_execution`, and operations, and uses them directly to create a new order. Without validation, this could lead to issues if invalid data is passed.

Impact

The lack of input validation could lead to the creation of orders that are inexecutable or that behave unexpectedly. For example, if `earliest_execution` is later than `latest_execution`, the order cannot be executed. If an operation is empty or contains invalid operations, the execution of the order could fail or have unintended effects.

Evidence

```
fn create_order(
// ...
            let order_id =
                self.orders
                    .insert_order(earliest_execution, latest_execution,
&operations)?;
            self.env().emit_event::<OrderCreated>(OrderCreated {
                id: order_id,
                earliest_execution,
                latest_execution,
                operations,
            });

            Ok(order_id)
        }

…
```

`lib.rs`. No input sanitization is done for the parameters `earliest_execution` are performed `latest_execution.`

Affected Resources

- `src/contracts/abax_treasury/lib.rs, lines 69-89`

Recommendation

For such important function such as `create_order`, we recommend adding input `sanitzation`. For example, check that `earliest_execution` is earlier than `latest_execution`, that operations are not empty and contains valid operations, and that `earliest_execution` and `latest_execution` are in the future. Consider using ink!'s Result type to return an error if the validation fails.

References

N/A

3.4 KS–ABXDAO–F–4 Risk of Underflow

Severity	Impact	Likelihood	Status
Low	Medium	Low	Resolved

Description

The `increase_counter` function in the `vault_counter_storage_item.rs` uses the function `overflowing_add` to perform addition over `u128` which allows overflow. The developers of Abax Finance safely take care of the edge case when an overflow happen. However, there would be an error if a double overflow happens, and the counter would not be valid anymore.

Impact

The incorrect counter would represent wrongly the shares of different users, and therefore break the voting logic of the DAO.

Evidence

```
pub fn increase_counter(&mut self, amount: u128) {
        let mut counter = self.counter();
        counter = counter.overflowing_add(amount).0;
        self.counter.set(&counter);
    }
```

`vault_counter_storage_item.rs`. The increase of the counter could theoretically overflows twice

Affected Resources

- `src/contracts/governor/modules/govern/storage/vault_counter_storage_item.rs lines 13-17-89`

Recommendation

Even though the impact of this finding could be medium or even high, the probability of it occurring is extremely low. Therefore, we conclude that this finding has a severity level set to Low. We recommend changing the type of the variable `counter` to `u256`, which would reduce the probability of encountering this issue to 0.

References

N/A

3.5 KS–ABXDAO–F–5 Lack of Input Validation in _deposit & _withdraw Functions

Severity	Impact	Likelihood	Status
Low	Medium	Low	Acknowledged

Description

The `_deposit` and `_withdraw` functions implemented in the Governor part of the DAO do not perform any checks to validate their inputs. Specifically, they do not check whether the provided account addresses are non-zero and whether the provided asset and share amounts are greater than zero. This could potentially lead to unexpected behavior or vulnerabilities.

Impact

If invalid or unnecessary inputs are allowed, the contract could consume more resources than necessary. This could lead to increased costs for the users of the contract. This can also lead to emission of unnecessary events and incorrect state changes, leading to loss of funds and poor user experience.

Evidence

```
 #[overrider(PSP22VaultInternal)]
    fn _deposit(
        &mut self,
        caller: &AccountId,
        receiver: &AccountId,
        assets: &Balance,
        shares: &Balance,
    ) -> Result<(), PSP22Error> {
        ink::env::debug_println!("total supply pre {:?}",
self._total_supply());
        self.counter.increase_counter(*shares);
        self.govern.set_last_stake_timestamp(receiver);
        self._deposit_default_impl(caller, receiver, assets, shares)?;
        ink::env::debug_println!("total supply post {:?}",
self._total_supply());
        Ok(())
    }
```

`lib.rs`. Lack of input validation

Affected Resources

- `dao-contracts-main/src/contracts/governor/lib.rs, lines 82-147`

Recommendation

We recommend adding input validation checks at the start of the `_deposit` and `_withdraw` functions. For example, check that the account addresses are non-zero and that the asset and share amounts are positive. This will help to ensure that the functions only continue execution if the inputs are valid, helping to prevent unexpected behavior or vulnerabilities.

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–ABXDAO–O–1	Informational	Inconsistencies between the Code and Comments	Resolved
KS–ABXDAO–O–2	Informational	TODO still present in the Code	Resolved
KS–ABXDAO–O–3	Informational	Incorrect Error Message String	Resolved
KS–ABXDAO–O–4	Informational	Use of a Yanked Dependency	Informational
KS–ABXDAO–O–5	Informational	ink::env::debug_println! still present in the Code	Resolved

Observations overview.

4.1 KS–ABXDAO–O–1 Inconsistencies between the Code and Comments

Description

There are inconsistencies between some comments and the implemented code. An example is for the computation the cost of phase 1 by multiplying `amount_phase1` with `self.tge.cost_to_mint_milion_tokens` and dividing by 10^6. However, the comment associated with this line of code does not accurately describe what the code is doing as it claims that denominator should be 10^8. This inconsistency can lead to confusion for anyone reading or maintaining the code.

Another inconsistency, is the fact that the constant `PART_OD_FOUNDATION_E3: u16` is set to 20 while the comment indicates that it should be set to 200.

Affected Resources

- `src/contracts/abax_tge/lib.rs/ lines, lines 493-494`
- `src/contracts/abax_tge/constant.rs, lines 26-27`

Recommendation

After a discussion with the Abax Finance team, it was agreed that the comments were incorrect and therefore there is no security issue. We recommend updating the comment to accurately reflect what the code is performing.

4.2 KS–ABXDAO–O–2 TODO still present in the Code

Description

The DAO codebase contains a TODO comment 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 can lead to unexpected behavior or incomplete functionality.

Affected Resources

- `src/contracts/abax_tge/constant.rs, lines 135`

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, rather than 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–ABXDAO–O–3 Incorrect Error Message String

Description

The file `capped_inflation_storage_field.rs` performs a subtraction operation using the `checked_sub` method and throwing an error when an underflow happens. However, the error message that is returned when an underflow is encountered is `MathError::Overflow` while it should be `MathError::Underflow`.

Affected Resources

- `src/contracts/abax_token/modules/capped_inflation/capped_inflation_storage_field.rs, line 60`

Recommendation

The error message should accurately reflect the error that occurred. In this case, the error message should be `MathError::Underflow` instead of `MathError::Overflow`.

4.4 KS–ABXDAO–O–4 Use of a Yanked Dependency

Description

The project is using the `pest` crate, which flagged to be yanked by `cargo audit`

Affected Resources

Valid for the entire project.

Recommendation

The crate is outdated or not maintained, it's recommended to look for alternatives that are actively maintained and have good community support.

4.5 KS–ABXDAO–O–5 ink::env::debug_println! still present in the Code

Description

The provided code of the DAO implementation is still having the code debugging printed message.

Affected Resources

- `src/contracts/abax_tge/lib.rs`
- `src/contracts/governor/lib.rs`
- `src/contracts/governor/module/govern/helpers/finalization.rs`
- `src/contracts/governor/module/govern/storage/govern_storage_item.rs`
- `src/contracts/governor/module/govern/traits/stucts/transaction.rs`

Recommendation

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

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.

Kickoff
Ramp-up
Review
Report
Verify

5.1 Kickoff

The Kudelski Security Team set up a kickoff 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 Abax Finance or its customers at risk.

The Kudelski Security Team identified 5 security issues: 1 high risk, 1 medium risks and 3 lower risk. 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:

- Ban withdraw or deposit during the voting period
- Multisig accounts for key roles
- Update of documentation

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