SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Arakis
Date: 19 Oct, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Arakis
Auditor	Viktor Raboshchuk | Solidity SC Auditor at Hacken OU
Approved By	Yves Toiser | Solidity SC Auditor at Hacken OU
Tags	ERC20 token; AirDrop;
Platform	EVM
Language	Solidity
Methodology	Link
Website	-
Changelog	12.10.2023 - Initial Review 19.10.2023 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Arakis (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Arakis is a token with the following contracts:

- *ArakisToken* – is a simple ERC-20 token that mints the entire initial supply to a deployer. Additional minting is not allowed. The contract has functionality to distribute airdrops to an array of addresses. The amount and addresses are chosen by the owner of the contract.
  It has the following attributes:
  - Name: ARAKIS
  - Symbol: AURICS
  - Decimals: 18
  - Total supply: 10 billion tokens.

Privileged roles

- Only the owner of the Arakis.sol contract is able to execute the token airdrop.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are present:
  - Tokenomic description is provided.
- Technical description is provided:
  - NatSpec is sufficient.

Code quality

The total Code Quality score is **8** out of **10**.

- The development environment is not configured.
- The code follows all best practices and style guides.

Test coverage

Code coverage of the project is **00.00%**.

- Tests are not provided (according to our methodology, tests are not mandatory for projects smaller than 250 lines of codes).

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**.The system users should acknowledge all the risks summed up in the risks section of the report.

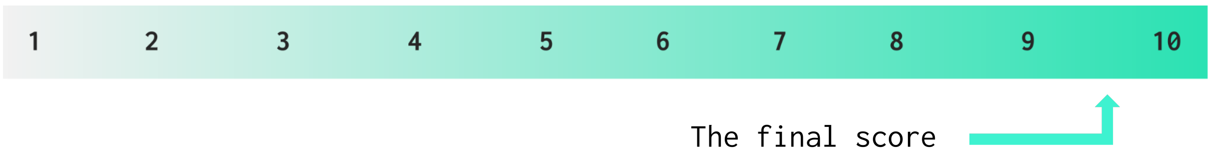

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
12 October 2023	3	0	0	0
19 October 2023	0	0	0	0

Risks

- All tokens are minted to a single address. The secureness of the supply depends on the secureness of key storage.
- The AirdropTokens function has an unbounded array of addresses. In case the array is excessively large, this can lead to a Gas limit exception and the transaction will fail. To avoid this, the owner can invoke the function multiple times with smaller arrays.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

No high severity issues were found.

■■ Medium

No medium severity issues were found.

■ Low

L01. Public Function That Should Be External

Description

Functions that are meant to be exclusively invoked from external sources should be designated as "external" rather than "public".

Path: ./Arakis.sol : AirdropTokens()

Impact

This is essential to enhance both the gas efficiency and the overall security of the contract.

Recommendation

- Transition the relevant functions, which are exclusively utilized by external entities, from their current "public" visibility setting to the "external" visibility setting.

References

- https://docs.soliditylang.org/en/latest/cheatsheet.html#function-visibility-specifiers

Found in: 0xa6ed00dc459b6f4a4737ec5edda0b33b6d1e0ef8

Status:

Fixed (Revised contract: 0x3f14d0f3f2321abeef4d306913b598f412db61c2)

L02. Floating Pragma

Description

The project uses floating pragmas ^0.8.17.

Path: ./Arakis.sol : AirdropTokens()

Impact

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version, which may include bugs that affect the system negatively.

Recommendation

- Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

References

- https://docs.soliditylang.org/en/latest/layout-of-source-files.html

Found in: 0xa6ed00dc459b6f4a4737ec5edda0b33b6d1e0ef8

Status:

Fixed (Revised contract: 0x3f14d0f3f2321abeef4d306913b598f412db61c2)

L03. Redundant Zero Address Check Before Transfer

Description

AirdropTokens() verifies that the recipient's address is not the zero address before initiating the token transfer, yet this validation is already conducted within the _transfer() function.

Redundant conditional checks within Solidity smart contracts refer to situations where multiple conditional statements exist to validate the same condition.

Path: ./Arakis.sol : AirdropTokens()

Impact

Redundant conditional checks resulting in unnecessary complexity and Gas waste.

Recommendation

- Remove redundant checks that duplicate the validation efforts of other existing checks.

Found in: 0xa6ed00dc459b6f4a4737ec5edda0b33b6d1e0ef8

Status:

Fixed (Revised contract: 0x3f14d0f3f2321abeef4d306913b598f412db61c2)

Informational

I01. Cache Array Length Outside of Loop In AirdropTokens Function

Description

The Solidity compiler will always read the length of the array during each iteration.

Path: ./Arakis.sol : AirdropTokens()

Impact

As it is a memory array, this is an extra mload operation (3 additional Gas for each iteration except for the first).

Recommendation

- This extra cost can be avoided by caching the array length before the loop.

References

- https://soliditylang.org/blog/2020/11/04/solidity-ama-1-recap/

Found in: 0xa6ed00dc459b6f4a4737ec5edda0b33b6d1e0ef8v

Status:

Fixed (Revised contract: 0x3f14d0f3f2321abeef4d306913b598f412db61c2)

I02. Use Custom Errors Instead of Require Statement String

Description

Custom errors introduced by Solidity make error reporting cheaper as well as programmatically dynamic, making Solidity code more efficient and structured.

Path: ./Arakis.sol : AirdropTokens()

Recommendation

- Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

References

- https://soliditylang.org/blog/2021/04/21/custom-errors/

Found in: 0xa6ed00dc459b6f4a4737ec5edda0b33b6d1e0ef8v

Status:

Fixed (Revised contract: 0x3f14d0f3f2321abeef4d306913b598f412db61c2)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	Medium
Medium Likelihood	**High**	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Deployed Contract	https://mumbai.polygonscan.com/address/0xa6ed00dc459b6f4a4737ec5edda0b33b6d1e0ef8#code
Commit	-
Whitepaper	-
Requirements	Link
Technical Requirements	-
Contracts	File: ./ArakisToken.sol SHA3: 0f8b43ef16a00fa5c587aec97a39f2af5e7351d18b9657f4d60ebbf95c127f98

Second review scope

Deployed Contract	https://polygonscan.com/address/0x3f14d0f3f2321abeef4d306913b598f412db61c2#code
Commit	-
Whitepaper	-
Requirements	Link
Technical Requirements	-
Contracts	File: Arakis.sol SHA3: 21d06c7bf9c6852cf75b1ed5101ed88e369933fec9cb375c2c9188b1f02ab5ae