HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Arkania
Date: May 13th, 2022

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 Arkania.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC20 token; Staking; Crowdfunding
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://arkania.io/
Timeline	21.04.2022 - 10.05.2022
Changelog	29.04.2022 - Initial Review 13.05.2022 - Second Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by Arkania (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.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/ArkaniaProtocol/Code
Commit:
7723ba8
Technical Documentation: Yes
JS tests: Yes
Deployed Contracts Addresses: No

Contracts:
File: ./ANIA.sol
SHA3: ed22f13654aa687e37d81adaee3939398b77871ba7757395bc736da6b7ec8433

File: ./Crowdfunding.sol
SHA3: 6ec53371f8a4b6016d617e421ed7a5d36ea917ea4c8024b44603b742a16a78cc

File: ./TokenStaking.sol
SHA3: 94b752e99447c4db63ddd7ca39accd61d347d7b2c6088ff1a7523157eb809690

Second review scope

Repository:
https://github.com/ArkaniaProtocol/Code
Commit:
4fe28ae
Technical Documentation: Yes
JS tests: Yes
Deployed Contracts Addresses: No
Contracts:
File: ./Ania.sol
SHA3: b3ea9a1d1f3eb22d9eaef41cc1139cb8a1e1cfb41fd49e2c71388df430b36666

File: ./AniaLottery.sol
SHA3: d0f2fb3d89cecc16be5fb5223ccf9732bbb320c7450e4fc805bbff534edab81b

File: ./AniaStake.sol
SHA3: c3f307e8e0965543a0c04caeeb6b798ab4983a9bfd13fe877753ead8f2c2d8b2

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

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

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

Executive Summary

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

Documentation quality

The Customer provided whitepaper and technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **5** out of **10**. The code does not follow the official style guide.Tests are provided but it’s impossible to run and check them due to the lack of a properly configured environment.

Architecture quality

The architecture quality score is **5** out of **10**. No development environment was provided.

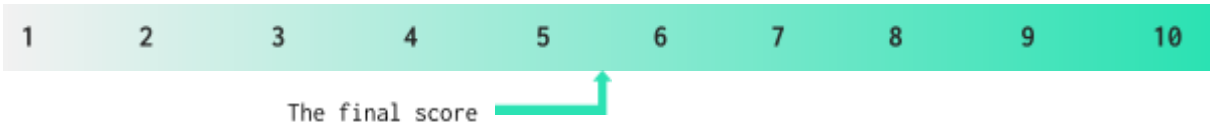
Security score

As a result of the audit, the code contains **1** high and **3** low issues. The security score is **5** 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: **5.5**.



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

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Failed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be destroyed until it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Passed
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed

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

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Failed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed

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

System Overview

ArkaniaProtocol is a mixed-purpose system with the following contracts:

- Ania — simple ERC-20 token that mints all initial supply to the deployer. Additional minting is not allowed.
  It has the following attributes:
  - Name: Arkania
  - Symbol: ANIA
  - Decimals: 18
  - Total supply: 100m tokens.
- AniaStake — a contract that rewards users for staking their Ania tokens. APY is established by the owner.
  It has the following attributes:
  - Ania: Ania token address
- AniaLottery - a contract that allows users to participate in the lottery. Users will be considered lottery winners if their staking amount matches contract specified conditions.
  It has the following attributes:
  - AniaStake: AniaStake address

Privileged roles

- The owner contract can add, delete admins.
- The owner contract can add, delete stable coins.
- The owner can withdraw all tokens from the contract.
- The admin can remove admins.
- The admin can withdraw tokens from the specified project.
- The admin can add specified user addresses to the whitelist for participation in the lottery.
- The admin can remove specified user addresses from the whitelist.
- The admin can start lottery
- The admin can arbitrarily create, update or delete projects.
- The admin can change tier values
- The admin can change tier ticket values

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

Findings

Critical

1. **Syntax error.**

   Syntax error on line 278

   Contracts:Ania.sol

   Function: approve

   Recommendation: Remove symbol “<”

   Status: Fixed (Revised commit: 4fe28ae)

2. **Wrong file name.**

   Wrong import file name, line 4

   Contracts:AniaStake.sol

   Function: -

   Recommendation: Change the import name or rename the file.

   Status: Fixed (Revised commit: 4fe28ae)

3. **Assets integrity.**

   It is possible to buy or claim tokens without sending proper amount of coins. The only validation is that a msg.value should be at least 1 wei.

   This can lead to the depletion of token balances.

   Contracts:AniaLottery.sol

   Function: claim, buy, checkBuy

   Recommendation: use the exchange rate to calculate the amount of eth required to purchase tokens.

   Status: Fixed (Revised commit: 4fe28ae)

High

1. **The owner can withdraw all tokens.**

   The owner can withdraw all tokens from the contract.

   This may affect users' funds.

   Contracts: AniaLottery.sol

   Function: withdraw

   Recommendation: remove the ability of the owner to withdraw user tokens or update the documentation accordingly.

   Status: New

1. **Potential DoS.**

   If a ‘projectBilling’ address is a contract address and this contract has a fallback or receive function, the transfer will fail due to 2300 gas limit.

   Possible DoS.

   Contracts:AniaLottery.sol

   Function: buy

   Recommendation: Ensure that the Billing address is an externally-owned address or use the call method instead of transfer.

   Status: Fixed (Revised commit: 4fe28ae)

2. **Potential DoS.**

   The function iterates over all stakers and changes the state every iteration.

   Gas consumption can differ a lot between different transactions. Possible DoS if the number of stakers is large enough.

   Contracts:AniaStake.sol

   Function: redistributeRewards

   Recommendation: do not iterate over all stakers.

   Status: Fixed (Revised commit: 4fe28ae)

3. **Potential DoS.**

   The function iterates over all stakers.

   Gas consumption can differ a lot between different transactions. Possible DoS if the number of stakers is large enough.

   Contracts:AniaStake.sol

   Function: totalStakesWithRewards

   Recommendation: do not iterate over all stakers.

   Status: Fixed (Revised commit: 4fe28ae)

4. **Potential DoS.**

   The function iterates over all whitelisted project users.

   Gas consumption can differ a lot between different transactions. Possible DoS if the number of stakers is large enough.

   Contracts:AniaLottery.sol

   Function: removeUsersFromWhitelist

   Recommendation: do not iterate over all users.

   Status: Fixed (Revised commit: 4fe28ae)

5. **Potential DoS.**

   The function iterates over all whitelisted project users.

   Gas consumption can differ a lot between different transactions. Possible DoS if the number of stakers is large enough.

   Contracts:AniaLottery.sol

   Function: getProjectStakeCap, getProjectUsers, logoutFromWhitelist, _checkUserExistInProject

   Recommendation: do not iterate over all users.

   Status: Fixed (Revised commit: 4fe28ae)

6. **Potential DoS.**

   The function iterates over all project lottery winners users.

   Gas consumption can differ a lot between different transactions. Possible DoS if the number of stakers is large enough.

   Contracts:AniaLottery.sol

   Function: getProjectRaisedAmount, getLotteryWinners,getLotteryWinner, setLotteryWinnerClaimedStatus, _checkUserisProjectWinner

   Recommendation: do not iterate over all users.

   Status: Fixed (Revised commit: 4fe28ae)

7. **User reward is never verified.**

   User reward (tierTicketValue) is never considered in calculations.

   It is not clear what reason to have tier ticket value assigned to the users.

   Contracts:AniaLottery.sol

   Function: -

   Recommendation: Review and fix the logic

   Status: Fixed (Revised commit: 4fe28ae)

Medium

No medium severity issues were found.

Low

1. **Floating pragma**

   The contracts use floating pragma ^0.8.2.

   Contract: Ania.sol, AniaStake.sol, AniaLottery.sol

   Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Reported

2. **Style guide violation.**

   Contracts do not follow the Solidity code style guide.

   Contracts: AniaStake.sol, AniaLottery.sol

   Recommendation: Follow the official Solidity code style guide.

   Status: Reported

3. **Outdated version of Solidity used.**

   The contracts use 0.8.2 Solidity version. The latest Solidity version is 0.8.13. It is recommended to use it.

   Contracts: AniaStake.sol, AniaLottery.sol, Ania.sol

   Function: -

   Recommendation: change the Solidity version where it is possible.

   Status: New

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

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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 audit cannot guarantee the explicit security of the audited smart contracts.