HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ArdCoin
Date: 15 Nov, 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 ArdCoin
Auditor	Viktor Raboshchuk | Solidity SC Auditor at Hacken OU
Approved By	Yves Toiser | Solidity SC Auditor at Hacken OU
Tags	ERC20 token;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://ardcoin.com/

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 ArdCoin (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

PROJECT_NAME is a staking protocol with the following contracts:

- *Token* — The ERC-20 token has functionality to blacklist users, a snapshot mechanism, an emergency stop mechanism through pause/unpause functions, and the ability to mint/burn tokens. It also inherits ERC20Permit contract.
  It has the following attributes:
  - Name: ArdCoin
  - Symbol: ARDX
  - Decimals: 18
  - Total supply: not set.

Privileged roles

- The contract has 5 privileged roles that are assigned to the deployer of the contract in the constructor.:
  - DEFAULT_ADMIN_ROLE - By default, the admin role for all roles is `DEFAULT_ADMIN_ROLE`, which means that only accounts with this role will be able to grant or revoke other roles.
  - SNAPSHOT_ROLE - Can use the snapshot features.
  - PAUSER_ROLE - Can pause transfer features.
  - MINTER_ROLE - Can use mint features of the smart contract.
  - BLACKLIST_ROLE- Can update blacklist.

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

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 and technical requirements are provided.

Code quality

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

- The development environment and deployment instructions are sufficient.
- NatSpec is sufficient.

Test coverage

Code coverage of the project is **100.00%** (branch coverage)

- For a project with less than 250 LOC (Lines of Code) the test coverage is not mandatory, and it is not accounted for in the final score.

Security score

As a result of the audit, the code does not contain 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: **10**. The system users should acknowledge all the risks summed up in the risks section of the report.

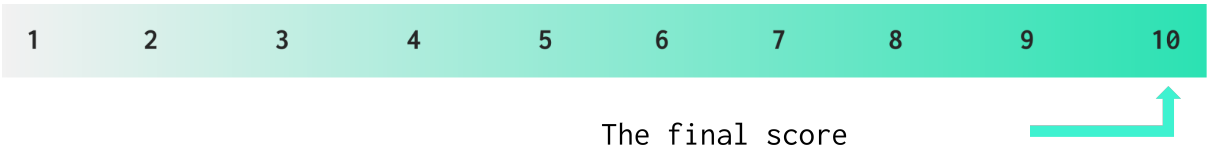

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
07 November 2023	3	0	0	0
15 November 2023	0	0	0	0

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

Risks

- The whitepaper states that a total supply of 5,158,308,000 ARDX tokens was issued in January 2019. Addresses with the MINTER_ROLE can mint additional tokens. **The code currently lacks an upper limit for minting**.
- The BLACKLIST_ROLE has the capability to add any address to the blacklist, preventing users from transferring their ARDX tokens.
- The PAUSER_ROLE has the capability to pause token operations for all users.

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

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

No medium severity issues were found.

Low

L01. Redundant Import of ERC20.sol and ERC20Permit.sol

ArdCoin inherits ERC20 that is part of ERC20Burnable, and inherits ERC20Permit that is part of ERC20Votes contract.

Impact

Using redundant import increases deployment Gas price and decreases code quality.

Path: ./contracts/ArdCoin.sol

Recommendation: Remove redundant imports, and ensure that the contract is imported only in the required locations, avoiding unnecessary duplications.

Found in: bcceaf2

Status: Fixed (Revised commit: 54384cf)

Remediation: Unneeded imports have been removed from the smart contract.

L02. Missing Event Emitting for blackListUpdate Function

There is no event emitted in the blackListUpdate function. The function stops blacklisted addresses from transferring their ARDX tokens, this process is not being tracked.

Impact

The users will not be able to subscribe to events and check what is going on with the project. This can lead to wrong assumptions about the current contract state.

Path: ./contracts/ArdCoin.sol

Recommendation: Emit the event whenever corresponding action happens.

Found in: bcceaf2

Status: Fixed (Revised commit: 54384cf)

Remediation: The event has been added to the function blackListUpdate and will be emitted in case of blacklisting a user.

L03. Incorrect Inheritance Order of OpenZeppelin Contracts

ArdCoin inherits multiple Opezeppeline contracts, with no designated inheritance order.

Solidity supports multiple inheritance, meaning that one contract can inherit several contracts. Multiple inheritance introduces an ambiguity called the Diamond Problem: if two or more base contracts define the same function, which one should be called in the child contract? Solidity deals with this ambiguity by using reverse C3 Linearization, which sets a priority between base contracts.

Impact

That way, base contracts have different priorities, so the order of inheritance matters.

Neglecting inheritance order can lead to unexpected behavior.

Path: ./contracts/ArdCoin.sol

Recommendation: When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/.

The correct order is AccessControl, Pausable, ERC20Burnable, ERC20Snapshot, ERC20Votes.

Found in: bcceaf2

Status: Fixed (Revised commit: 54384cf)

Remediation: The order of inheritance has been changed from "most base-like" to "most derived."

Informational

I01. Use uint256(1)/uint256(2) instead of Boolean States

Boolean variables in Solidity are more expensive than uint256 or any type that takes up a full word, due to additional Gas costs associated with write operations. When using boolean variables, each write operation emits an extra SLOAD to read the slot's contents, replace the bits taken up by the boolean, and then write back. This process cannot be disabled and leads to extra Gas consumption.

```
mapping(address => bool) private _blacklist;
```

Impact

By using uint256(1) and uint256(2) for representing true and false states, it is possible to avoid a Gwarmaccess (100 Gas) cost and also avoid a Gsset (20000 Gas) cost when changing from false to true, after having been true in the past. This approach helps in optimizing Gas usage, making contract more cost-effective.

Path: ./contracts/ArdCoin.sol

Recommendation: For optimal storage of true/false values, it is recommended to utilize 1 as false and 2 as true.

References

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27

Found in: bcceaf2

Status: Acknowledged (Revised commit: 54384cf)

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: ./contracts/ArdCoin.sol : _beforeTokenTransfer()

Recommendation

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

References

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

Found in: bcceaf2

Status: Acknowledged (Revised commit: 54384cf)

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 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.

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

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.

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

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.

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

Appendix 2. Scope

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

Initial review scope

Repository	https://github.com/ArdCoin/ArdCoin-Token-Smart-Contract
Commit	bcceaf2a6
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/ArdCoin.sol SHA3: 7d406aff0f5f7fe4149f53b5e8d4889007918e299dfd850f072e425777de87e4

Second review scope

Repository	https://github.com/ArdCoin/ArdCoin-Token-Smart-Contract
Commit	54384cf2e
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/ArdCoin.sol SHA3: e41507a8689a979b58357850f7afaa1b971834d02a4cad49dd08a7fea1c22711