HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Ainomo
Date: 26 January, 2024

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Ainomo
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Tags	AI/Blockchain
Platform	EVM
Language	Solidity
Methodology	Hacken Methodology
Website	https://ainomo.com/
Changelog	25.12.2023 – Initial Review 26.01.2024 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Ainomo(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

Ainomo Protocol ("Ainomo") thrives on AI-driven directives, laying the groundwork for decision-making with unparalleled precision and reliability. With a diverse suite of AI services at its disposal, covering machine learning, natural language processing, computer vision, and robotic process automation, the company stands out in its adept utilization of artificial intelligence. AINOMO harnesses the capabilities of distributed data storage systems like Hadoop Distributed File System (HDFS) and Amazon S3, ensuring robust data warehousing that boasts high availability and scalability.
The files in the scope:

- **RouterHelper.sol** - The Abstract contract facilitates various swaps, and provides library functions to determine transaction specifics..
- **IRouterHelper.sol** - The interface of RouterHelper.
- **SwapRouter .sol** - The contract contains various functions for swaps, including supporting multiple swap routes.
- **CustomErrors.sol** - The error handling contract.
- **Library.sol** - The library contract provides the necessary functionality for the swap operation.
- **TransferHelper.sol** - The library contract provides methods for interacting with ERC20 and ETH tokens, as well as for safely sending them.
- **InterfaceComptroller.sol** - The interface of Comptroller contract.
- **IPair.sol** - The interface of Pair contract.

Privileged roles

- SwapRouter.sol:
    - onlyOwner privilege roles:
        - sweepToken() method caller: can withdraw ERC20 token from the contract.
        - setBNBAddress() : set the BNB token address.

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 provided.
- Technical description is provided.
- NatSpecs are very good.

Code quality

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

- The development environment is configured.
- The order of the function does not follow the style guide perfectly, but the exceptions make sense.

Test coverage

Code coverage of the project is **97.08%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Some functions are not tested.
- Interactions by several users are tested thoroughly.

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: **9.7**.

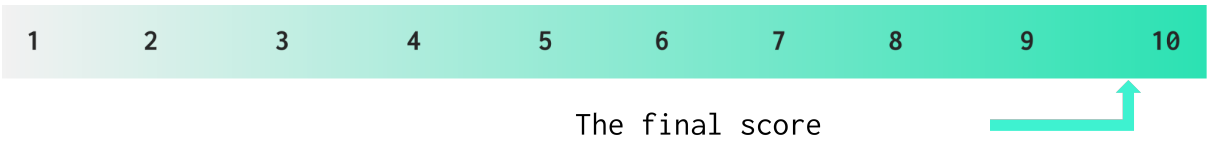

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
25 Dec 2023	3	1	1	0
26 Jan 2024	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	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	The contract should not be self-destructible while it has funds belonging to users.	Passed	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Passed	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	

Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Passed	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	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	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	
Flashloan Attack	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. Contracts shouldn't rely on values that can be changed in the same transaction.	Passed	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	

Gas Limit and Loops	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	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

Critical

No critical severity issues were found.

High

H01. Missing Swap Path Validation

Impact	High
Likelihood	Medium

All the functions that would execute a swap take as argument the path that will be used to swap between the two tokens concerned. However, no check is done to verify that the path matches with the two tokens that are supposed to be swapped.

It is possible to perform a swap with a path if the user has the relative tokens , even if the TokenAddress used as parameter is the address of another token.

Paths: ./contracts/Swap/SwapRouter.sol

./contracts/Swap/RouterHelper.sol

Recommendation: Check that the tokens at the beginning and the end of the path are the ones that are supposed to be swapped.

Found in: e35d0ac

Status: Fixed (cc6b8cb)

Medium

M01. Mishandled Edge Case

Impact	Medium
Likelihood	Medium

In the presented code, if-else control statements are structured in a manner that can lead to unintended execution under certain conditions. Specifically, when the contract has insufficient liquidity, statement 'A' or 'B' could become zero. Despite this, due to the use of the logical "AND" operator (&&), the function will still execute successfully.

Path: ./contracts/Swap/lib/Library.sol:quote(), getAmountOut(), getAmountIn()

Recommendation: Change the “&&” operator with “||” operator.

Found in: e35d0ac

Status: Fixed (cc6b8cb)

M02. Unverifiable Logic

Impact	Low
Likelihood	Medium

The SwapRouter contract externally calls *VBep20.sol* inside the *_supply()* function. The contract uses or interacts with code that is out of audit scope.

Path: ./contracts/Swap/SwapRouter.sol

Recommendation: Add the code that cannot be verified in the scope or document it properly if it cannot be added to the scope.

Found in: e35d0ac

Status: Mitigated (The used contract, which is a wrapper contract, calls the main contract without modifying the function.)

Low

L01. Missing Zero Address Validation

Impact	Medium
Likelihood	Low

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Path: ./contracts/Swap/SwapRouter.sol: constructor(), sweepToken()

Recommendation: Implement zero address checks.

Found in: e35d0ac

Status: Fixed (cc6b8cb)

L02. Boolean Equality

Impact	Low
Likelihood	Medium

Boolean constants can be used directly and do not need to be compared to true or false.

Path: ./contracts/Swap/SwapRouter.sol : ensureTokenListed()

Recommendation: Remove boolean equality.

Found in: e35d0ac

Status: Fixed (cc6b8cb)

L03. Redundant Check

Impact	Low
Likelihood	Medium

In the Library.quote() function, the following checks are done:

```
if (reserveA == 0 && reserveB == 0) {revert InsufficientLiquidity();}
```

```
require(reserveA > 0 && reserveB > 0,
```

The first check is useless as it checks something that is also checked in the second one.

Path: ./contracts/Swap/lib/Library.sol : quote()

Recommendation: Remove redundant code.

Found in: e35d0ac

Status: Fixed (cc6b8cb)

Informational

I01. SPDX License Identifier Not Provided

Impact	Low
Likelihood	Medium

"SPDX-License-Identifier" is not provided in the source files.

Paths: ./contracts/Swap/interfaces/CustomErrors.sol,

./contracts/Swap/interfaces/IPair.sol,

./contracts/Swap/interfaces/IBNB.sol,

./contracts/Swap/interfaces/Itoken.sol,

./contracts/Swap/interfaces/InterfaceComptroller.sol,

./contracts/Swap/interfaces/Library.sol,

./contracts/Swap/interfaces/IRouterHelper.sol,

./contracts/Swap/interfaces/SwapRouter.sol,

Recommendation: "SPDX-License-Identifier" should be added to each source file.

Found in: e35d0ac

Status: Fixed (cc6b8cb)

I02. Floating Pragma

Impact	Low
Likelihood	Medium

The project uses floating pragmas `^0.8.13`.

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.

Path: ./contracts/Swap/interfaces/CustomErrors.sol :

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.

Found in: e35d0ac

Status: Fixed (cc6b8cb)

I03. Public Functions That Should Be External

Impact	Low
Likelihood	Medium

Functions that are only called from outside the contract should be defined as external.

Path: ./contracts/Swap/RouterHelper.sol : quote(), getAmountOut(), getAmountIn(), getAmountsOut(), getAmountsIn()

Recommendation: Make these functions external.

Found in: e35d0ac

Status: Fixed (cc6b8cb)

I04. Copy of Well-Known Contract

The system uses a copy of a well-known contract instead of reusing it.

Path: ./contracts/Swap/SwapRouter.sol : nonReentrant()

Recommendation: Use the OpenZeppelin non reentrancy guard instead of copying it.

Found in: cf6b8cb

Status:Fixed (c5c88cb)

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.

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

Repository	https://github.com/ainomodatalab/ainomoprotocol
Commit	e66d0ac9df768263dbc830546280b4d0f9385c4e
Website	https://ainomo.com
Requirements	-
Technical Requirements	-
Contracts	File: contracts/Swap/IRouterHelper.sol File: contracts/Swap/RouterHelper.sol File: contracts/Swap/SwapRouter.sol File: contracts/Swap/interfaces/CustomErrors.sol File: contracts/Swap/interfaces/InterfaceComptroller.sol File: contracts/Swap/interfaces/IPair.sol File: contracts/Swap/interfaces/IBNB.sol File: contracts/Swap/interfaces/Itoken.sol File: contracts/Swap/lib/Library.sol File: contracts/Swap/lib/TransferHelper.sol

Second review scope

Repository	https://github.com/ainomodatalab/ainomoprotocol
Commit	c66b8cb0735cf0ded3435161c6ea2e2d6c4b48e4
Website	https://ainomo.com
Requirements	-
Technical Requirements	-
Contracts	File: Swap/IRouterHelper.sol File: Swap/RouterHelper.sol File: Swap/SwapRouter.sol File: Swap/interfaces/CustomErrors.sol File: Swap/interfaces/InterfaceComptroller.sol File: Swap/interfaces/IPair.sol File: Swap/interfaces/IBNB.sol File: Swap/interfaces/Itoken.sol File: Swap/interfaces/IBNB.sol File: Swap/lib/Library.sol File: Swap/lib/TransferHelper.sol