SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Avatea
Date: October 26th, 2022

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

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 Avatea
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	MarketMaker
Platform	EVM
Network	Ethereum Mainnet
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://avatea.io
Timeline	30.08.2022 - 26.10.2022
Changelog	06.09.2022 - Initial Review 27.09.2022 - Second Review 26.10.2022 - Third Review

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

Table of contents

Introduction 4
Scope 4
Severity Definitions 6
Executive Summary 7
Checked Items 8
System Overview 11
Findings 12
Disclaimers 14

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

Introduction

Hacken OÜ (Consultant) was contracted by Avatea (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/Uri-Avatea/Avatea-Smart-Contracts
Commit:
83c322f474a84ee055c5137bd014005912034a2e
Documentation:
Functional requirements

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/MarketMaker.sol
SHA3: d4d25377be64297e87fbe88a3f3b3613369728d82d191d5e3976ed6eeea326a7

File: ./contracts/MarketMakerDeployer.sol
SHA3: 14dd24ff72f37136f3fc74d7ea9ee3cb62b37be9baa9e7594d49a2580a3c6631

Second review scope

Repository:
https://github.com/Uri-Avatea/Avatea-Smart-Contracts
Commit:
24b59590d5b94d45c3cc32373cad60b234aa6cce
Documentation:
Functional requirements

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/MarketMaker.sol
SHA3: 5514c9c39f3809d013073b2b41526a8f338532b4b6b216d192b5533a47af3e9c

File: ./contracts/MarketMakerDeployer.sol
SHA3: 6d7fe0111fe9dce7f3ab7adbf9f318c893910694fc219823d6332f51ba8ed06f

Third review scope

Repository:
https://github.com/Uri-Avatea/Avatea-Smart-Contracts
Commit:
fb390dafebd183d1fc84162be2a7cdf47446cf88
Documentation:
Functional requirements

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/MarketMaker.sol
SHA3: 7237cb287ab0f8116c321e91bb8603062b74c32439c7b5b3ade825d00697e8ed

File: ./contracts/MarketMakerDeployer.sol

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

SHA3: 827267631ca787451ac46edc7afd27ce3065d4c24ced3f74e5f6fbc77484c6cf

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 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**. Instructions on how to run tests and build the project are missing.

Code quality

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

Test coverage

Deployment and basic user interactions are covered with tests. Negative cases covered.
Test coverage is 100.00%. (branch coverage)

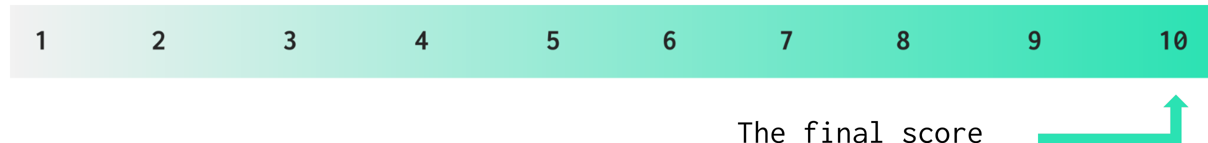
Security score

As a result of the second audit, the code does not contain any security 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: **10.00**.



The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
1 September 2022	4	2	1	0
23 September	0	0	0	0

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.	Passed
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.	Passed
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 self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
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.	Not Relevant

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

Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
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 or be predictable.	Passed
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.	Passed
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant
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.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed

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

Environment Consistency	**Custom**	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	**Custom**	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.	Passed
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
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

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

System Overview

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

- *MarketMaker* – A smart contract that offers market making for projects. This contract can be used by both project owners as well as holders of the project's token.
- *MarketMakerDeployer* – A smart contract that deploys Market Maker contracts for projects.

Privileged roles

- The owner is assigned to the wallet that has deployed the contract and corresponds to the project owner.
- The controller is assigned by the deployer and corresponds to the Market Making Algorithm.
- The moderator is a wallet in control of the Avatea team.
- The paused functionality applies to all functionalities that move funds away from the Market Maker (eg: withdrawals, market making, or liquidity providing).

Risks

- **The repository contains contracts that are out of the audit scope. Secureness and correctness of those contracts are not guaranteed.**

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

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

1. Funds Lock

Native coins can be accepted by the contract but do not have any ways to withdraw these funds.

This can lead to funds lock.

Path: ./contracts/MarketMaker.sol : receive()

Recommendation: Remove the *receive* function.

Status: Fixed (Revised commit: 24b5959)

■■ Medium

1. Hardcoded Value

The *MIN_GAS_FOR_CONTROLLER* value is hardcoded. In case of Gas price growth, *controllerWallet* account will not be able to send transactions.

Path: ./contracts/MarketMakerDeployer.sol

Recommendation: Allow an owner to change the value.

Status: Fixed (Revised commit: 24b5959)

2. Tautology or Contradiction

Detects expressions that are tautologies or contradictions.

Path: ./contracts/MarketMaker.sol: buy(), sell(), stakeInLiquidityMaker()

Recommendation: Remove duplicate statements and create a local variable for repeated statements.

Status: Fixed (Revised commit: 24b5959)

■ Low

1. Missing Zero Address Validation

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

This can lead to unwanted external calls to 0x0.

Path: ./contracts/MarketMaker.sol : constructor()

Parameters: baseTokenAddress, pairedTokenAddress, controller, moderator

Recommendation: Implement zero address checks.

Status: Fixed (Revised commit: 24b5959)

2. **Writing State Variables in a Loop**

Writing a state variable or an attribute of it in a loop may be costly, in terms of Gas fees. The variables totalVested, availableBaseFee, availablePairedFee are inside the loops during 'sell' function execution call.

Path: ./contracts/MarketMaker.sol :sell()

State variables: totalVested, availableBaseFee, availablePairedFee

Recommendation: Save the state variable or its attribute into a local variable and update only once instead of updating for each loop step.

Status: Fixed (Revised commit: 24b5959)

3. **Floating Pragma**

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

The project uses floating pragmas 0.8.9.

Paths: ./contracts/MarketMaker.sol

./contracts/MarketDeployer.sol

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

Status: Fixed (Revised commit: 24b5959)

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.