HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Ease
Date: Aug 15th, 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 Ease
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token; Staking; DAO
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://ease.org/
Timeline	26.07.2022 - 15.08.2022
Changelog	02.08.2022 - Initial Review 15.08.2022 - 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 Ease (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/EaseDeFi/gvToken
Commit:
9379907aaea908c0cc762310f90e16b95f2ab1ab
Technical Documentation:

Type: Technical description
Link

Type: Functional requirements
Link

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/core/GvToken.sol
SHA3: 564d0d7cc60d8dce88d53920ad5e3f2504d49adbefb833c612f9f695547057ad

File: ./contracts/core/EaseToken.sol
SHA3: f7fdbcec4e761a2f9022f20ca8c365727c4560298008cd98f29803b8da81be23

File: ./contracts/core/BribePot.sol
SHA3: 1e1ea47b54ca3f4d03b830666c41001032312dcd8a8d628a7c383f1fe9f7b0e2

File: ./contracts/core/TokenSwap.sol
SHA3: bc9620499aed8e639478eef436022923033968b0d2b832f4536a114015bd6ead

File: ./contracts/library/MerkleProof.sol
SHA3: 694defddbeb583a5653725bb0b75244720e4f8952c30e056dc61f463dba32065

File: ./contracts/external/SolmateERC20.sol
SHA3: 2fd4ef4bf5f6e604c941c3e72a30551c0a4e193fc96be3f4a4e780354c694fa3

Second review scope

Repository:
https://github.com/EaseDeFi/gvToken
Commit:
1920de6bcc6e6dcfbd54ead4569a4810caec08f8
Technical Documentation:

Type: Technical description
Link

Type: Functional requirements
Link

Integration and Unit Tests: Yes
Contracts:

File: ./contracts/core/GvToken.sol
SHA3: c5cb8b18b11b199516ac00281ee3aa39f11f5dccf3d095679df975a157a58ed8

File: ./contracts/core/EaseToken.sol
SHA3: fa86486799ddda10839c23d90a7b5288011f36d54e3087db51e4fe853184f24a

File: ./contracts/core/BribePot.sol
SHA3: a6ece6f75b24c7ea8fa3abf301cf279a71bd2f7ac0f547f3f36a6b380d0eead0

File: ./contracts/core/TokenSwap.sol
SHA3: 2f2ace6d0a151d592faccc936d4603744325541b48fbc5a919fc3fcb9c9f6974

File: ./contracts/library/MerkleProof.sol
SHA3: 694defddbeb583a5653725bb0b75244720e4f8952c30e056dc61f463dba32065

File: ./contracts/external/SolmateERC20.sol
SHA3: feeb8282577465207d8b90cddf12285fd43829a4116d8310ea4e1463512d3e60

File: ./contracts/external/Delegatable.sol
SHA3: 9204cf5de8cec3fe3c405e9b1ea4c3de28ee42996d952e6b67e2a4c798d7b02d

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 methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**. Provided documentation covers all contract functionality. Deposits migration is not covered in the documents but exists in the contract.

Code quality

The total CodeQuality score is **9** out of **10**. Code is well-formatted and commented. Code contains unresolved TODOs (*TokenSwap* contract).

Architecture quality

The architecture quality score is **10** out of **10**. Code follows a single responsibility principle and is well-organized.

Security score

As a result of the audit, the code contains **2** medium, and **2** low severity issues. The security score is **8** 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: **8.3**.

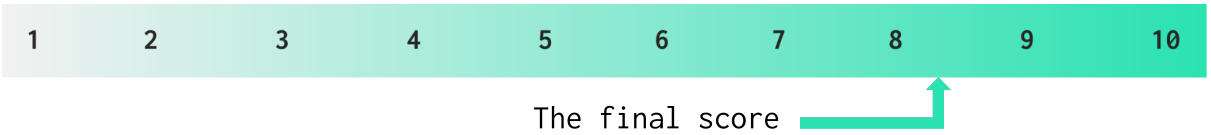

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
2 August 2022	8	3	0	1
15 August 2022	2	2	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.	Not Relevant
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.	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 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	SWC-115	tx.origin should not be used for	Not Relevant

through tx.origin		authorization.	
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 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	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 or be predictable.	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.	Passed
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.	Failed
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
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of	Passed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
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.	Failed
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.	Not Relevant
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.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

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

- *EaseToken* – simple ERC-20 token with unlimited minting.
  It has the following attributes:
  - Name: Ease Token
  - Symbol: EASE
  - Decimals: 18
  - Total supply: unlimited
- *TokenSwap* – swapping contract that accepts Armor or vArmor tokens and exchanges them for Ease tokens.
- *MerkleProof* - library contract that deals with verification of Merkle trees (hash trees).
- *SolmateERC20* - open-source ERC20 + EIR-2612 contract implementation
- *GvToken* - voting token for governance, has staked for off-chain loss calculation, and a place where users interact with the bribe pot contract.
- *BribePot* - allows users to sell their gvToken power to the highest bidders. A user may add a stake to the bribe pot, then receive rewards based on how many people are paying for bribes each week.

Privileged roles

- The governance role of the *GvToken* contract can change the cryptography signature.
- The governance role of the *GvToken* contract can change time delay for withdrawals. Delay could not be less than 2 weeks.
- The governance role of *EaseToken* contract can mint tokens without any limit.
- The *GvToken* can call *deposit*, *withdraw,* and *getReward* functions of BribePot contract.

Risks

- Ease tokens can be minted unlimitedly, and no burning mechanisms are provided.
- IVArmor, IRcaController contracts are out of audit scope and could not be verified.
- *stakingToken* in the GvToken contract could not be verified before deployment to be sure that it is an EaseToken contract.
- Deposits migration is not documented and could be used to create deposits at any time after contract deployment (to manipulate voting power).

Findings

■■■■ Critical

1. Race condition

During funds withdrawal, a user could lose access to newly added deposits. *withdrawFinalize* always removes deposits from the end of the array, so deposited funds between *withdrawRequest* and *withdrawFinalize* functions calls could be lost.

File: ./contracts/core/GvToken.sol

Contract: GvToken

Function: withdrawFinalize

Recommendation: Fix the logic to prevent the race conditions.

Status: Fixed (1920de6bcc6e6dcfbd54ead4569a4810caec08f8)

■■■ High

No high severity issues were found.

■■ Medium

1. Requirements incompliance

Provided documentation declares that voting power will reach maximum in a year after deposit, but *_balanceOf* function ignores all deposits made more than a year during voting power calculation.

File: ./contracts/core/GvToken.sol

Contract: GvToken

Function: _balanceOf

Recommendation: Check if the existing logic is correct.

Status: Reported

2. Data consistency

Any user can modify *periodFinish* variable to an unlimited value using an external function *bribe*. This variable is shared for all users and vaults.

File: ./contracts/core/BribePot.sol

Contract: BribePot

Function: bribe

Recommendation: Check if the existing logic is correct.

Status: Reported

3. **Overcomplicated logic**

   _updateDepositsAndGetPopCount is overcomplicated and not Gas efficient. During *withdrawRequest* it is enough to store the pending withdraw amount and *withdrawFinalize* remove all deposits (looping from 0) to satisfy the withdrawal request.

   File: ./contracts/core/GvToken.sol

   Contract: GvToken

   Function: _updateDepositsAndGetPopCount

   Recommendation: Simplify the logic of withdrawals.

   Status: Fixed (1920de6bcc6e6dcfbd54ead4569a4810caec08f8)

Low

1. **Redundant import**

   hardhat/console.sol must not be in the deployment version of the contract.

   Files: ./contracts/core/GvToken.sol, ./contracts/core/BribePot.sol,

   Contracts: GvToken.sol, BribePot.sol,

   Function: -

   Recommendation: Remove *console.sol* imports and usages before the deployment.

   Status: Fixed (1920de6bcc6e6dcfbd54ead4569a4810caec08f8)

2. **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.

   Files: all

   Contracts: all

   Function: -

   Recommendation: Consider locking the pragma version to the latest one and avoid using a floating pragma in the final deployment.

   Status: Fixed (1920de6bcc6e6dcfbd54ead4569a4810caec08f8)

3. **Unchecked transfer**

   The return value of an external *transfer*/*transferFrom* call is not checked. Several tokens do not revert in case of failure and return false. If one of these tokens is used in audited contracts, the

withdrawal will not revert if the transfer fails, and an attacker can call the withdrawal for free.

File: ./contracts/core/TokenSwap.sol

Contract: TokenSwap

Functions: swap, swapVArmor

Recommendation: Check the result of the transfer if it is true or not.

Status: Reported

4. **Unused function**

   The functions created but not used in the project should be deleted. This will make a more Gas efficient contract.

   File: ./contracts/external/SolmateERC20.sol

   Contract: SolmateERC20

   Function: _burn

   Recommendation: Remove unused function.

   Status: Fixed (1920de6bcc6e6dcfbd54ead4569a4810caec08f8)

5. **Public functions instead of external**

   Some functions are declared as public, although they are not called internally in the related contract.

   Public function visibility consumes more Gas than external visibility.

   File: ./contracts/core/BridgePot.sol

   Contract: BridgePot

   Functions: rewardPerToken, withdraw, getReward

   Recommendation: Change public visibility with external.

   Status: Partially fixed (1920de6bcc6e6dcfbd54ead4569a4810caec08f8)

6. **Redundant operation**

   `_updateDepositsAndGetPopCount` function in the loop updates deposits map `_deposits[user][i - 1] = userDeposit`, which has no effect. This line could be deleted to simplify the code and optimize Gas usage.

   File: ./contracts/core/GvToken.sol

   Contract: GvToken

   Function: _updateDepositsAndGetPopCount

Recommendation: Remove redundant operation.

Status: Fixed (1920de6bcc6e6dcfbd54ead4569a4810caec08f8)

7. **Code organization**

   _deposit function could be split into two different functions: update balances and transfer *stakingToken*. The current implementation of the function is too generic, and for some cases, *PermitArgs* are redundant.

   Files: ./contracts/core/GvToken.sol

   Contract: GvToken

   Function: _deposit

   Recommendation: Split functionality to update balances and transfer tokens.

   Status: Fixed (1920de6bcc6e6dcfbd54ead4569a4810caec08f8)

8. **Code duplication**

   claimAndDepositReward functionality is duplicated in multiple functions: *unStake*, *stake*, *withdrawFromPot*.

   File: ./contracts/core/GvToken.sol

   Contract: GvToken

   Functions: unStake, stake, withdrawFromPot

   Recommendation: Consider reusing code instead of duplicating it.

   Status: Fixed (1920de6bcc6e6dcfbd54ead4569a4810caec08f8)

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.