HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Gains Associates
Date: April 18th, 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 Gains Associates.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC20 token whitelist pool; Lock wallet;
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.gains-associates.com/#/home
Timeline	16.03.2022 - 18.04.2021
Changelog	22.03.2022 - Initial Review 18.04.2022 - Revising

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 Gains Associates (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:

Repository:
https://github.com/lotfiZouad/gains-s
Commit:
73de1056315504d07192b9aaae0555dcdee0c3db
Technical Documentation: No
JS tests: Yes
Contracts:
./contracts/ClaimToken.sol
./contracts/ClaimTokenFactory.sol
./contracts/GainsLockWallet.sol
./contracts/GainsLockWalletFactory.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

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 no technical requirements. The total Documentation Quality score is **0** out of **10**.

Code quality

The total Code Quality score is **7** out of **10**. The code follows official language style guides and has low unit tests coverage.

Architecture quality

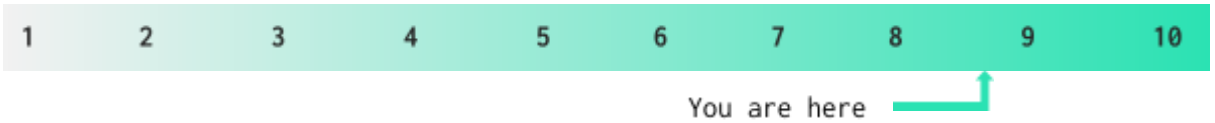
The architecture quality score is **10** out of **10**. The project has clean and clear architecture and follows the best practices.

Security score

As a result of the audit, the code contains **2** medium and **1** low severity issue. The security score is **10** out of **10**.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.7**

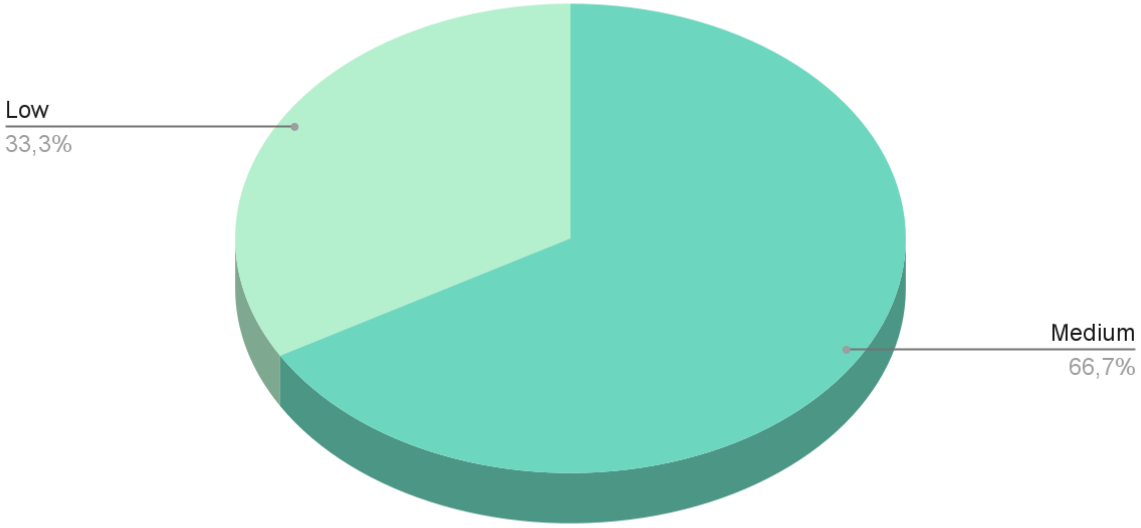


Notices

1. The ClaimToken.sol contract owner may arbitrarily update the token distribution list.
2. Some contracts are upgradable. Our report covers only those versions of contracts in the Scope section.

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

Graph 1. The distribution of vulnerabilities after the second audit.



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

Findings

Critical

No critical severity issues were found.

High

1. **Double spending.**

   The function `updateAddress` is available for calling by the owner. It updates the state of the `tokenClaimed` mapping. The mapping stores the amount of claimed tokens by address. The function `updateAddress` removes the previous beneficiary address from the mapping and assigns the claimed amount to the new address. Tokens are not redistributed, only the mapping state is updated. Then the previous address may call the `claim` function and get the whitelisted tokens again. It may happen only if the `merkleRoot` hash was not updated properly before calling `updateAddress`.

   This can lead to double spending if the contract owner calls `updateAddress` before the proper `merkleRoot` state update.

   Contracts: ClaimToken.sol

   Function: updateAddress, claim

   Recommendation: Remove `updateAddress` function or add the `merkleRoot` state check to the `updateAddres` to verify that the old beneficiary address may not claim the same amount of tokens from the whitelist anymore.

   Status: Fixed

2. **Inability to get whitelisted funds.**

   After calling the `updateAddress` function, a new beneficiary address is added to the `tokenClaimed` mapping, but if this address is added to the `tokenClaimed` mapping, it may not get all the whitelisted funds according to the logic of the `claim` function, also `updateAddress` function does not update `claimedAddresses` mapping state.

   The address added to the `tokenClaimed` mapping by the `updateAddress` function may not get all the whitelisted funds from the tokens pool.

   Contracts: ClaimToken.sol

   Function: updateAddress, claim

   Recommendation: Remove `updateAddress` function or rework `claim` function logic to check if the address exists in `claimedAddresses` mapping.

   Status: Fixed

Medium

1. **Inability to withdraw non-locked tokens.**

   The GainsLockWallet.sol contract has a function that allows withdrawal of any token from the contract when the release time is up, but the expected behavior is to store the one kind of token. A user may accidentally send a wrong token to the lock wallet, and there is no opportunity to withdraw it before the lock period ends.

   Contracts: GainsLockWallet.sol

   Function: withdrawGains

   Recommendation: Allow the beneficiary to withdraw any type of tokens except the token, which should be locked.

   Status: Fixed

2. **Unexpected msg.sender.**

   The GainsLockWallet.sol contract has the `onlyOwner` modifier, which allows calling functions only if the `msg.sender` is `beneficiary`. Calling the lock wallet functions from the GainsLockWalletFactory.sol contract may not be possible because the factory contract will be the `msg.sender`, but the factory address is not a `beneficiary` address.

   It may not be possible to call lock wallet functions from the factory contract.

   Contracts: GainsLockWalletFactory.sol, GainsLockWallet.sol

   Function: withdrawGains, relockGains, relockGainsBySubscription, withdrawGains

   Recommendation: Rework logic to allow calling throw the factory contract or remove part of some logic including such functions as `relockGainsBySubscription`, `withdrawGains`.

   Status: Fixed

3. **Checks-Effects-Interactions pattern violation.**

   The untrusted token may have arbitrary implementation. The reentrancy attack is possible during the external call to an untrusted token contract implementation.

   Any state variable modifications that happen after an external call is executed can lead to unexpected behaviors in the function execution.

   Contracts: ClaimToken.sol

   Function: claim

   Recommendation: Interact only with trusted contracts, check external calls for reentrancy, follow check effect interaction pattern.

   Status: New

4. **The signed amount of tokens has no deadline.**

   If a specific amount of tokens is assigned to a user, it may be claimed at any time. The Merkle tree may be updated by the contract admin anytime. If the Merkle tree is updated and the new state does not have some of the previous leaves with an encoded amount of tokens the inability to claim tokens may be unexpected to the user.

   The lack of the deadline field obliges the admin to track the assigned amount for the whole time and not to remove the leaf from the Merkle tree after the updates.

   Contracts: ClaimToken.sol

   Function: claim

   Recommendation: It is recommended to include the deadline timestamp to the Merkle tree leaf structure. This gives a flexible way to manage the deadlines of the token assignments.

   Status: New

Low

1. **Zero value token transaction.**

   The contract adds the amount of claimed tokens to the `tokenClaimed` mapping. If the user calls the `claim` function twice the execution goes to the conditional block where the claimed amount is subtracted from the requested amount `_amount - tokenClaimed[msg.sender]`, if the user claims tokens for the second time the result will be equal to 0. This value will be passed to the `safeTransfer` call and zero tokens will be transferred to the user.

   Zero value transactions will happen without any errors if the user requests the claimed amount twice or more.

   Contracts: ClaimToken.sol

   Function: claim

   Recommendation: It is recommended to rework the condition block, and add a `required` statement to check if the amount to send is more than 0.

   Status: New

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

Recommendations

1. There is no need to use the SafeMath library for Solidity version greater than 0.8. Since Solidity 0.8, the overflow/underflow check is implemented on the language level.

   Contracts: ClaimToken.sol, GainsLockWallet.sol

2. It is recommended to update the `claim` function logic by adding a nonce value inside the Merkle tree leaf, this will give an opportunity to simplify the condition statements inside the `claim` function when a user has few signed leaves.

   Contracts: ClaimToken.sol

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.