HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Embr Holdings Limited
Date: November 9th, 2021

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 Embr Holdings Limited.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	CrowdSale
Platform	Binance Smart Chain / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/teamembr/smart-contracts
Commit	bd830b5747421178227df0159fc5327b62f38c14
Technical Documentation	YES
JS tests	YES
Website	joinembr.com
Timeline	12 OCTOBER 2021 - 18 OCTOBER 2021
Changelog	18 OCTOBER 2021 - INITIAL AUDIT 01 NOVEMBER 2021 - SECOND REVIEW 09 NOVEMBER 2021 - THIRD REVIEW

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Embr Holdings Limited (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 contract and its code review conducted between October 12th, 2021 - October 18th, 2021.

Second review conducted on November 1st, 2021.

Third review conducted on November 9th, 2021.

Scope

The scope of the project is smart contracts in the repository:
Repository:
    https://github.com/teamembr/smart-contracts
Commit:
    bd830b5747421178227df0159fc5327b62f38c14
Technical Documentation: Yes (in repository readme.md)
JS tests: Yes (in repository test/)
Contracts:
    crowdsale.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 ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are well-secured.

Insecure	Poor secured	Secured	Well-secured

You are here

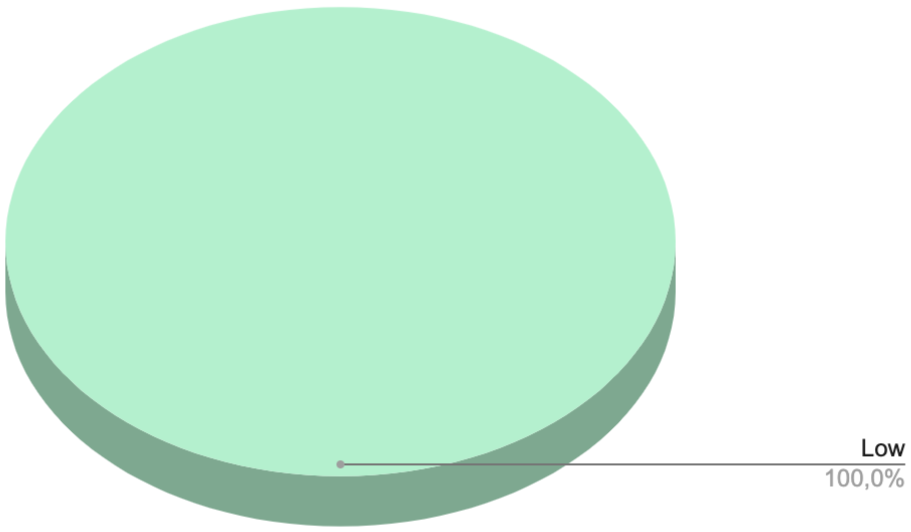
Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **3** medium and **9** low severity issues.

After the second review and also considering comments added by the customer security engineers found that there are still unresolved **2** medium and **3** low severity issues.

After the third review security engineers found **1** low severity issues.

Graph 1. The distribution of vulnerabilities after the audit.



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 can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

Audit overview

■ ■ ■ ■ Critical

No critical issues were found.

■ ■ ■ High

No high severity issues were found.

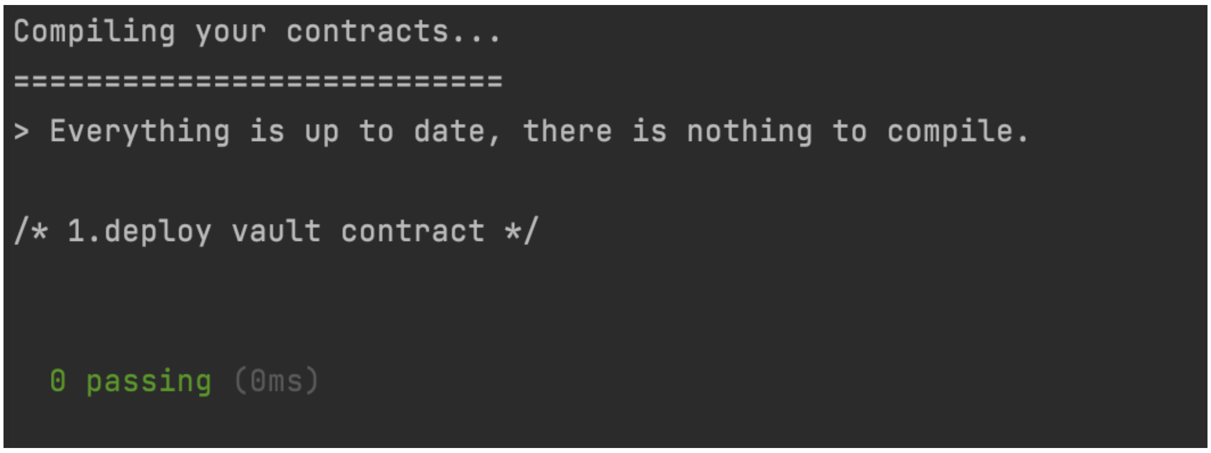
■ ■ Medium

1. Tests could not be run

While the documentation doesn't include an explanation on how to execute the tests, we've gone this way:

- truffle init
- npm i ethers
- truffle test test/test_script.js

But, unfortunately, even when we match the solidity versions, no tests could be executed. Below is the only output of the script:



```
Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.

/* 1.deploy vault contract */



  0 passing (0ms)
```

Recommendation: Please make sure all tests could be executed and there is a script or description of how to run them. Also, please make sure your tests are cover at least 95% of code branches.

Status: Fixed.

2. No emitting events

There are no emitting events neither in the CrowdSale nor the Vault contracts.

Recommendation: Please emit events on changing critical parameters as well as when performing contract actions.

Status: Responded.

Customer comment: *We have decided not to emit events as we don't require off-chain data in this case and to also reduce the gas used (up to 4x as advised by our Solidity consultant)*

3. No check for transfer result

While there could be any BEP20 token contract address set to the CrowdSale, not every contract will throw on error. BEP20 defines that a contract should return a boolean result of the transfer function, therefore there should be checking for the result.

Contract: crowdsale.sol

Functions: buy, getMyTokens

Recommendation: Please check the result of the transfer function call.

Status: Fixed.

■ Low

1. Tests configured incorrectly

We were able to run tests by the given instructions, but there are also some changes that should be made to accomplish that:

- rename “abi” => “abi-interfaces”
- line 17 of “test/test.js” change “abi-interfaces.vault.abi” => “abi-interfaces/vault.abi”

Recommendation: Please fix the test scripts.

Status: Fixed.

2. Tests running slow

As the docs stated: “The test may take over 45 minutes to run, due to dependency both on the public BSC testnet”. But why not to fork the testnet and run tests in the local ganache environment with the ability to manually “mine” any number of blocks you need.

Recommendation: Please try to re-work tests to run them locally not remotely.

Status: Fixed.

3. Different solidity pragma versions in one codebase.

Using different solidity versions in one codebase make it harder to compile, deploy and test contracts.

Recommendation: Please use one Solidity version.

Status: Acknowledged.

Customer Comment: Leaving unchanged due to development constraints

4. Conformance to Solidity naming conventions

Solidity defines a naming convention that should be followed.

Contract: crowdsale.sol

State Variables: ADJ_CONSTANT, _own_address, token_address, wl_script_wallet, vault_contract_address, approved_founders, approved_founders_length, sale_stages, sale_stages_length, acive_sale_stage, withdraw_schedules, withdraw_schedules_length, user_schedule_withdraw_status, white_list, user_crowdsale_balance, func_sign, investors_balances, init_investors_balances

Function Parameters: isApprovedFounder, setVaultContract, setActiveSaleStage, registerFounder, removeFounderByAddress, addSaleStage, addWithdrawSchedule, signFounder, resetSignatures, isAllFoundersSigned, emergencyWithdrawTokens, withdraw, hash, hash5uint256bool, getUserCrowdsaleBalance

Recommendation: Follow the Solidity naming convention.

Status: Fixed.

5. No return statement

While the function declared to return a boolean value it will always return false without an explicit return statement.

Contract: crowdsale.sol

Function: resetSignatures

Recommendation: Please either use a return statement or remove the return type declaration.

Status: Fixed.

6. Dead-code

SafeMath library defines a method mod(uint256,uint256) that is never used in the code.

Contract: crowdsale.sol

Functions: SafeMath.mod

Recommendation: Remove unused functions.

Status: Fixed.

7. Too many digits

Literals with many digits are difficult to read and review.

Contract: crowdsale.sol

Functions: buy

Constants: ADJ_CONSTANT

Recommendation: Please use scientific notation and/or ether units to simplify the numeric values (ie.: *100e3* instead of *100000*; *1 ether* instead of *1000000000000000000*)

Status: Partly Fixed. Constant is still contains many digits.

8. State variables that could be declared constant

Constant state variables should be declared constant to save gas.

Contract: crowdsale.sol

State Variable: ADJ_CONSTANT

Recommendation: Add the **constant** attributes to state variables that never change.

Status: Fixed.

9. A public function that could be declared external

public functions that are never called by the contract should be declared **external** to save gas

Contract: crowdsale.sol

Functions: getActiveSaleStage, getActiveScheduleSlot

Recommendation: Use the **external** attribute for functions never called from the contract.

Status: Fixed.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **3** medium and **9** low severity issues.

After the second review and also considering comments added by the customer security engineers found that there are still unresolved **2** medium and **3** low severity issues.

After the third review security engineers found **1** low severity issue.

Disclaimers

Hacken Disclaimer

Technical Disclaimer