HACKEN

SMART CONTRACT SECONDARY CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Bartertrade
Date: July 27, 2020
Platform: Ethereum
Language: Solidity

This document may contain confidential information about IT systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.
The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities fixed - upon decision of customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Bartertrade
Platform	Ethereum / Solidity
Initial Audit	
File	BART Audit.zip
File SHA-256	803B7E46DE6297B63CFBC67A337015A23ED745C026B2AD892BF064AE32CD1B9C
Date	23.07.2020
Secondary Audit	
File	BART Fixed Contracts.zip
File SHA-256	DE263C1E5B53FF7C0992E98CD3032A893512CAFFA597249A0B2AA770BF363D14
Date	27.07.2020

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Bartertrade (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between July 21st, 2020 - July 23rd, 2020. Secondary audit was conducted on 27th of July.

Scope

The scope of the project are smart contracts within the archive:
Initial Audit File - BART Audit.zip
Secondary Audit File - BART Fixed Contracts.zip
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

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

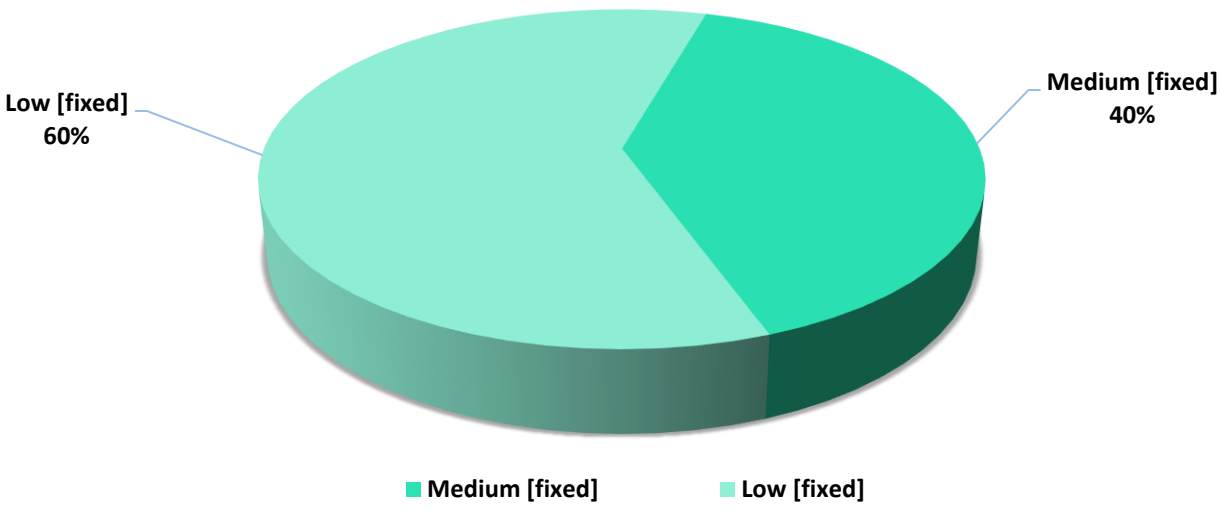
Insecure	Poor secured	Secured	Well-secured

You are here

Our team performed analysis of code functionality, manual audit and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in Audit overview section. General overview is presented in AS-IS section and all found issues can be found in Audit overview section.

We found 2 medium and 3 low issues in smart contract code during initial audit. All issues were fixed before secondary audit.

Graph 1. The distribution of vulnerabilities.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens lose etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have 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 tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

BARTToken

BARTToken imports AccessControl.sol, Context.sol, ERC20Burnable.sol from OpenZeppelin library.

Contract **BARTToken** is **Context**, **AccessControl**, **ERC20Burnable**.

Contract **BARTToken** defines following parameters:

- public constant bytes32 **MINTER_ROLE** set to **keccak256("MINTER_ROLE")**
- public constant bytes32 **PAUSER_ROLE** set to **keccak256("PAUSER_ROLE")**

Contract **BARTToken** has 2 functions:

- **constructor** - calls **ERC20** constructor with name and symbol parameters and sets msg.sender roles to DEFAULT_ADMIN_ROLE, MINTER_ROLE and PAUSER_ROLE
- **mint** - public virtual function that mints tokens to account

ERC20Migrator

ERC20Migrator imports IERC20.sol, Math.sol, Ownable.sol and Pausable.sol from OpenZeppelin library; it also imports BARTToken.sol.

Contract **ERC20Migrator** is **Pausable**, **Ownable**.

Contract **ERC20Migrator** defines following parameters:

- private BARTToken **_legacyToken**
- private BARTToken **_newToken**

Contract **BARTToken** has 8 functions:

- **constructor** - sets **_legacyToken** address
- **legacyToken** - public view function that returns **_legacyToken** address
- **newToken** - public view function that returns **_newToken** address
- **beginMigration** - public function that starts token migration
- **migrate** - public function that swaps specified amount of old token to new one. Has **whenNotPaused** modifier

- **migrateAll** - public function that swaps all available amount of old token to new one
- **pause** - public function that pauses migration. Has **onlyOwner** modifier
- **unpause** - public function that resumes migration. Has **onlyOwner** modifier

TokenVesting

TokenVesting imports Context.sol, IERC20.sol and SafeMath.sol from OpenZeppelin library.

Contract **TokenVesting** is **Context**.

Contract **TokenVesting** defines following parameters:

- private IERC20 **_token**
- struct **Vesting** that stores uint256 **start**, uint256 **interval**, uint256 **duration**, uint256 **balance** and uint256 **released**
- private mapping (**address** => **Vesting**) **_vestings**

Contract **BARTToken** has 7 functions:

- **constructor** - sets **_token** address
- **getVesting** - public view function that returns beneficiary's vesting
- **createVesting** - public function that creates new vesting
- **postponeVesting** - external function that postpones vesting by specified time

- **release** - public function releases vesting for beneficiary
- **releasableAmount** - public view function that returns releasable amount for beneficiary
- **vestedAmount** - public function that returns vested amount for beneficiary

Audit overview

Critical

No critical issues were found.

High

No high issues were found.

Medium

1. TokenVesting contract uses **pragma experimental ABIEncoderV2**, which is considered insecure. It's still considered experimental and may contain security flaws. It also may cause higher gas usage.

   Fixed before secondary audit

2. beginMigration function should have onlyOwner modifier. Without it, it's theoretically possible to set new token contract to any value.

   Fixed before secondary audit

Low

3. BARTToken contract defines PAUSER_ROLE, however, it doesn't have any functionality for it. It's recommended to remove PAUSER_ROLE from BARTToken contract.

   Fixed before secondary audit

4. Solidity pragma is not locked. It's recommended to lock pragma to latest stable version.

   Fixed before secondary audit

5. ERC20Migrator doesn’t use IERC20.sol and import can be removed.

   Fixed before secondary audit

Lowest / Code style / Best Practice

No best practice issues were found.

Conclusion

Smart contracts within the scope was manually reviewed and analyzed with static analysis tools. For the contract high level description of functionality was presented in As-is overview section of the report.

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

Overall quality of reviewed contracts is very good. Security engineers found several medium and low issues during initial audit. All issues were fixed before secondary audit.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to: 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 security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. 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 security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.