SMART CONTRACT SECONDARY CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Equus
Date: August 3, 2020
Platform: Ethereum
Language: Solidity

Document

Name	Smart Contract Code Review and Security Analysis Report for Equus
Platform	Ethereum / Solidity
Initial Audit	
Repository	https://github.com/equusprotocol/equus
Commit	24feb5765faa1ea19f7556a6075da2f35ac759a9
Branch	master
Date	29.07.2020
Secondary Audit	
Repository	https://github.com/equusprotocol/equus
Commit	b20579a77b9f15262bdfabb02b7ebcd2ad84d4c2
Branch	master
Date	03.08.2020

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Equus (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 27th, 2020 - July 29th, 2020. Secondary audit was conducted between August 1st, 2020 - August 3rd, 2020

Scope

The scope of the project are smart contracts within the repository:

Repository - https://github.com/equusprotocol/equus

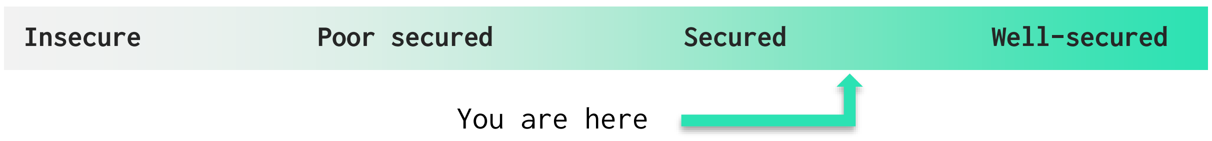
Branch - master

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

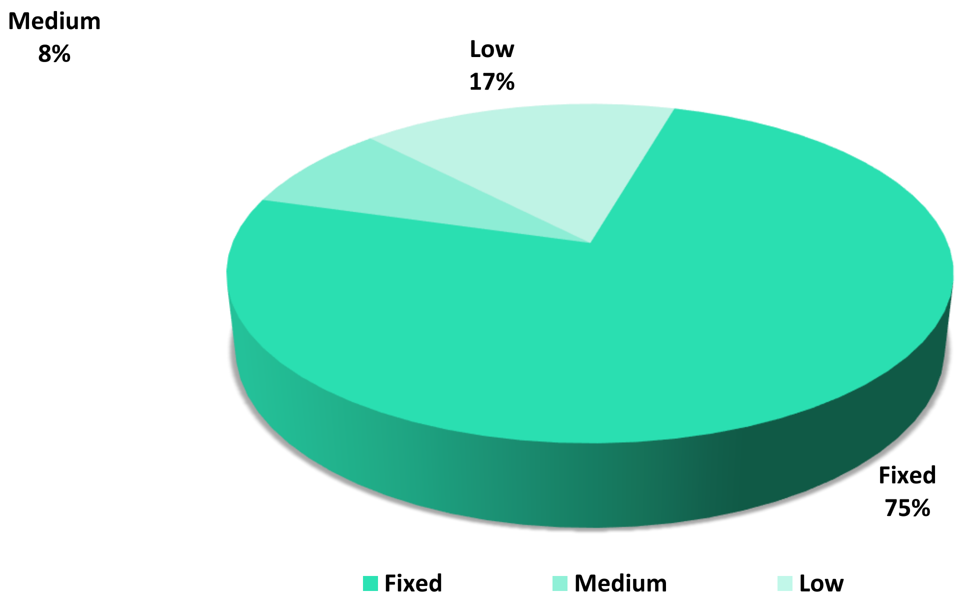
According to the assessment, Customer`s smart contracts are secured.



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 1 critical, 2 high, 4 medium, 4 low and 1 best practice issues in smart contract code during initial audit. Secondary audit undercovered 1 new issue and 2 issues were accepted by customer.

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

SafeMath library is standard library for math operations that prevents integer overflows and underflows.

IERC20 is the standard ERC20 interface.

Staking is the standard staking interface.

Contract **EQUUSProtocol** is **IERC20**, **Staking**.

Contract **EQUUSProtocol** defines following parameters:

- public constant string **name** is set to "EQUUS Protocol"
- public constant string **symbol** is set to "EQUUS"
- public constant uint8 **decimals** is set to 18
- public constant address **burnaddress** is set to 0x0

- mapping (address => uint256) balances
- mapping (address => uint256) stakedbalances
- mapping (address => uint) staketimestamps
- private mapping (address => mapping (address => uint256)) allowed
- uint256 totalSupply_
- uint256 totalstaked is set to 0
- address theowner

Contract EQUUSProtocol has 18 functions:

- constructor - sets owner and total supply and mints total supply to caller
- totalSupply - public view function that returns token total supply
- balanceOf - public view function that returns the balance of the account
- cutForBurn - public view function that returns 5% of specified value
- transfer - public function that transfers tokens from caller to receiver
- approve - public function that approves transfer of token for specified spender

- **allowance** - public view function that returns the allowance of the spender account for specified address
- **increaseAllowance** - public view function that increases allowance for user
- **decreaseAllowance** - public view function that decreases allowance for user
- **transferFrom** - public function that transfers from address to specified receiver by approved caller
- **stake** - public function that stakes tokens for caller
- **unstake** - public function that unstakes tokens for caller
- **totalStakedFor** - public view function that returns amounts staked for address
- **stakeTimestampFor** - public view function that returns the stake date for address
- **stakeTimeFor** - public view function that returns the time passed since staking for address
- **totalStaked** - public view function that returns the total amount of tokens staked in contract
- **supportsHistory** - public view function that returns the false for Staking interface compatibility
- **time** - public view function that returns current block timestamp

Audit overview

Critical

1. Deployer balance is not multiplied by 10 ** decimals in the constructor.

   Fixed in b20579a.

High

2. Stake, stakeFor and unstake functions perform token transfers, however, they don’t call Transfer event both for transfers to/from staking contract and to burn address.

   Fixed in b20579a.

3. Owner can stake for any address without any allowance and, thus, burn their tokens for stake and unstake.

   Fixed in b20579a.

Medium

4. Constructor should have Transfer event for initial supply set to caller.

   Fixed in b20579a.

5. Contract doesn’t have increaseAllowance and decreaseAllowance function to mitigate the well-known issues around setting allowances.

   Fixed in b20579a.

6. Stake and Unstake events may have no bytes description for gas economy.

Fixed in b20579a.

7. It’s possible to manipulate staketimestamps just sending low amount to tokens to staking contract. If user stakes <20 tokens - there will be no burn commission.

   Fixed in b20579a.

8. TransferFrom function should fire Transfer event.

   Introduced in b20579a.

Low

9. Solidity version is not locked in the pragma. It’s recommended lock the pragma with latest stable version of the solidity.

   Fixed in b20579a.

10. The code is not fully covered with documentation. The absence of documentation may cause differences between implementation and expected behavior of smart contracts. Documentation should be presented for all smart contract’s code.

11. Code is not covered with any unit tests. It increases the risk of the unexpected flaws in the smart contracts. It is recommended to have 100% code coverage with automated tests.

12. Burn function should be renamed - it’s calculating burn amount, not actually burning tokens.

    Fixed in b20579a.

Lowest / Code style / Best Practice

13. Time function can be removed and all its uses can be replaced with block.timestamp.

    Fixed in b20579a.

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.

Security engineers found several high and medium issues that may have serious security impact during initial audit. After secondary audit, contract has 1 medium and 2 low issues.

Overall quality of reviewed contracts is good.

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.