SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Splyt

Date: March 23rd, 2021

Document

Name	Smart Contract Code Review and Security Analysis Report for Splyt.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token, Vesting
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/splytcore/Splyt_TCPeNFT/tree/v2.3/contracts/Token
Commit	
Deployed contract	
Timeline	15 MAR 2021 – 23 MAR 2021
Changelog	18 MAR 2021 – INITIAL AUDIT 23 MAR 2021 – SECOND AUDIT

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Splyt (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 March 15th, 2021 – March 23rd, 2021.

Scope

The scope of the project is smart contracts in the repository:

Contract deployment address:
Repository
File:
    ERC20.sol
    Vesting.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 ▪ 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 ↑

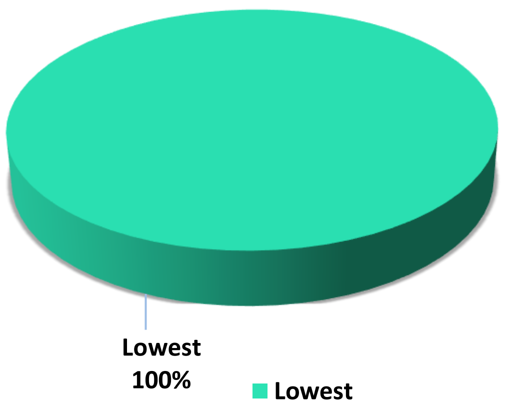
Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section,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

Security engineers found **1** lowest issue during the initial audit.

All found issues were fixed during the second audit.

Notice: the audit scope is limited and not include all files in the repository. Though, reviewed contracts are secure, we may not guarantee secureness of contracts that are not in the scope.

Graph 1. The distribution of vulnerabilities after the first review.



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
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

ERC20.sol

Description

The ERC20 contract is a fork of the OpenZeppelin ERC20 token implementation. Compared to the original, there have been no significant changes.

Vesting.sol

Description

Vesting is a contract for vesting.

Imports

Vesting contract has following imports:

- Owned.sol – from the project files.
- SafeMath.sol – from the project files.
- IERC20.sol – from the project files.

Inheritance

Vesting is Owned.

Usages

Vesting contract has following usages:

- SafeMath for uint256;

Structs

Vesting contract has no data structures.

Enums

Vesting contract has no enums.

Events

Vesting contract has following events:

- event TokensReleased(address token, uint256 amount);
- event TokenVestingRevoked(address token);

Modifiers

Vesting has no custom modifiers.

Fields

Vesting contract has following fields and constants:

- uint256 internal _cliff - end timestamp of the cliff period;
- uint256 internal _start - the timestamp at which the vesting begins;
- uint256 internal _duration - duration in seconds of the period during which tokens will be vested;
- bool internal _revocable - whether the vesting is revocable or not;
- address internal _revokeTo - the address to which revoked tokens are sent;
- address internal _beneficiary - the address of the beneficiary
- uint16 internal _firstMonthPercent - the percentage difference between the 1st month and the balance;
- uint256 constant internal _aMonth = 2629743 - a month in seconds;
- mapping (address => uint256) private _released - a map to store released amount of tokens;
- mapping (address => bool) private _revoked - a map to store revoked amount of tokens;

State-Changing Functions

Vesting contract has following state-changing functions:

- ***constructor***

  Description

  Initializes the contract.

  Visibility

  public

  Input parameters

- address beneficiary_
- uint256 start_
- uint256 cliffDuration_
- uint256 duration_
- bool revocable_
- address revokeTo_
- uint16 firstMonthPercent_

Constraints

- beneficiary_ address should not be zero.
- duration_ should be greater than or equal to cliffDuration_.
- duration_ should be greater than 0.
- The time for the end of the vesting should be in the future.

Events emit

None

Output

None

- ***release***

Description

Used to release tokens.

Visibility

public

Input parameters

- IERC20 token

Constraints

- There must be unreleased tokens.

Events emit

- TokensReleased(address(token), unreleased);

Output

None

- ***revoke***

Description

Used to revoke tokens.

Visibility

public

Input parameters

- IERC20 token

Constraints

- Only the owner can call it.
- It should be revocable.
- Only one revoke allowed.

Events emit

- TokenVestingRevoked(address(token));

Output

None

- ***_releasableAmount***

Description

Used to calculate releasable amount.

Visibility

public

Input parameters

- IERC20 token

Constraints

None

Events emit

None

Output

- uint256

- *_vestedAmount*

  Description

  Used to calculate vested amount.

  Visibility

  public

  Input parameters

  - IERC20 token

  Constraints

  None

  Events emit

  None

  Output

  - uint256

Read-Only Functions

Vesting contract has following read-only functions:

- function beneficiary() public view returns(address)
- function cliff() external view returns(uint256)
- function start() public view returns(uint256)
- function duration() public view returns(uint256)
- function revocable() public view returns(bool)
- function released(address token) public view returns(uint256)
- function revocked(address token) public view returns(bool)

Audit overview

🟥🟥🟥🟥 Critical

No critical issues were found.

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

No medium severity issues were found.

🟦 Low

No low severity issues were found.

🟦 Lowest / Code style / Best Practice

1. The _aMonth field of Vesting contract should be constant.

   Fixed during the second audit.

Conclusion

Smart contracts within the scope were 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 **1** lowest issue during the audit.

All found issues were fixed during the second audit.

Notice: the audit scope is limited and not include all files in the repository. Though, reviewed contracts are secure, we may not guarantee secureness of contracts that are not in the scope.

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 its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.