HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Terra Virtua
Date: May 17th, 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 Terra Virtua.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Staking
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/thedavidmeister/tv-prestige
Commit	F85A8C7681FB029646B18928CD3FEAEBCA98A780
Deployed contract	
Timeline	12 MAY 2021 - 17 MAY 2021
Changelog	17 MAY 2021 - INITIAL AUDIT

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Terra Virtua (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 May 12th, 2021 - May 17th, 2021.

Scope

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

```
Repository: https://github.com/thedavidmeister/tv-prestige
Commit: f85a8c7681fb029646b18928cd3feaebca98a780
Files:
      IPrestige.sol
      Prestige.sol
      PrestigeByConstruction.sol
      PrestigeUtil.sol
      TVKPrestige.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 secured well.

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. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found no issues during 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
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

Prestige.sol

Description

Prestige contract used to check and set the status of an account.

Imports

Prestige contract has following imports:

- IPrestige.sol - from the project files.
- PrestigeUtil.sol - from the project files.

Inheritance

Prestige is IPrestige.

Structs

Prestige contract has no data structures.

Enums

Prestige contract has no enums.

Events

Prestige contract has no custom events.

Modifiers

Prestige has no modifiers.

Fields

Prestige contract has following fields and constants:

- mapping(address => uint256) public statuses - statuses storage;

Functions

Prestige contract has following functions:

- *statusReport*

  Description

  Used to get a status of the account.

  Visibility

  public view

  Input parameters

  - address account;

  Constraints

  None

  Events emit

  None

  Output

  Returns the status as uint256.

- *setStatus*

  Description

  Used to set a status of an account.

  Visibility

  external

  Input parameters

  - address account;
  - Status newStatus;
  - bytes memory data - additional arbitrary data to inform status update requirements;

  Constraints

  - Status must not be NIL.

  Events emit

- emit StatusChange(account, [currentStatus, newStatus]);

Output

None

- *_afterSetStatus*

Description

Used to set a status of an account.

Visibility

internal

Input parameters

- address account;
- Status oldStatus;
- Status newStatus;
- bytes memory data - additional arbitrary data to inform status update requirements;

Constraints

None

Events emit

None

Output

None

PrestigeUtil.sol

Description

PrestigeUtil contract that contains a set of utilities for checking and changing the status.

Imports

PrestigeUtil contract has following imports:

- IPrestige.sol - from the project files.

Inheritance

PrestigeUtil inherits nothing.

Structs

PrestigeUtil contract has no data structures.

Enums

PrestigeUtil contract has no enums.

Events

PrestigeUtil contract has no custom events.

Modifiers

PrestigeUtil has no modifiers.

Fields

PrestigeUtil contract has following fields and constants:

- uint256 constant public UNINITIALIZED = uint256(-1) - indicates that the account has no status;

Functions

PrestigeUtil contract has following functions:

- *statusAtFromReport*

  Description

  Used to get the highest status achieved relative to a block number and status report.

  Visibility

  internal pure

  Input parameters

  - uint256 report;
  - uint256 blockNumber;

Constraints

None

Events emit

None

Output

Returns the status as uint256.

- *statusBlock*

Description

Returns the block that a given status has been held since according to a status report.

Visibility

internal pure

Input parameters

  - uint256 report;
  - uint256 statusInt - the status integer to read the block number for;

Constraints

None

Events emit

None

Output

Returns the block number as uint256.

- *truncateStatusesAbove*

Description

Resets all the statuses above the reference status.

Visibility

internal pure

Input parameters

- uint256 report;
- uint256 statusInt - the status integer to truncate above (exclusive);

Constraints

None

Events emit

None

Output

Returns the truncated report as uint256.

- *updateBlocksForStatusRange*

Description

Updates a report with a block number for every status integer in a range.

Visibility

internal pure

Input parameters

- uint256 report;
- uint256 startStatusInt - the status integer at the start of the range (exclusive);
- uint256 endStatusInt - the status integer at the end of the range (inclusive);
- uint256 blockNumber;

Constraints

None

Events emit

None

Output

Returns the updated report as uint256.

- *updateReportWithStatusAtBlock*

Description

Updates a report to a new status.

Visibility

internal pure

Input parameters

- uint256 report - the report to update;
- uint256 currentStatusInt - the current status integer according to the report;
- uint256 newStatusInt - the new status for the report;
- uint256 blockNumber - the block number to update the status at;

Constraints

None

Events emit

None

Output

Returns the updated report as uint256.

PrestigeByConstruction.sol

Description

PrestigeByConstruction contract that is used for checking the status of the account.

Imports

PrestigeByConstruction contract has following imports:

- PrestigeUtil.sol – from the project files.

- IPrestige.sol - from the project files.

Inheritance

PrestigeByConstruction inherits nothing.

Structs

PrestigeByConstruction contract has no data structures.

Enums

PrestigeByConstruction contract has no enums.

Events

PrestigeByConstruction contract has no custom events.

Modifiers

PrestigeByConstruction has following modifiers:

- onlyStatus(address account, IPrestige.Status status) - restricts access to functions depending on the status required by the function;

Fields

PrestigeByConstruction contract has following fields and constants:

- IPrestige public prestige;
- uint256 public constructionBlock;

Functions

PrestigeByConstruction contract has following functions:

- *constructor*

  Description

  Initializes the contract.

  Visibility

  public

Input parameters

- IPrestige _prestige;

Constraints

None

Events emit

None

Output

None

- *isStatus*

Description

Used to check the status of the account since construction.

Visibility

public view

Input parameters

- address account;
- IPrestige.Status status;

Constraints

None

Events emit

None

Output

Returns bool.

TVKPrestige.sol

Description

TVKPrestige is a contract used to obtain a certain status by staking tokens.

Imports

TVKPrestige contract has following imports:

- SafeMath - from the OpenZeppelin.
- IERC20 - from the OpenZeppelin.
- SafeERC20 - from the OpenZeppelin.
- Prestige.sol - from the project files.

Inheritance

TVKPrestige inherits Prestige.

Usings

TVKPrestige contract uses SafeERC20 for IERC20.

Structs

TVKPrestige contract has no data structures.

Enums

TVKPrestige contract has no enums.

Events

TVKPrestige contract has no custom events.

Modifiers

TVKPrestige has no modifiers.

Fields

TVKPrestige contract has following fields and constants:

- IERC20 public constant TVK = IERC20(0xd084B83C305daFD76AE3E1b4E1F1fe2eCcCb3988) - the TVK token contract;
- uint256 public constant NIL = uint256(0) - the amount needed to get `NIL` level;
- uint256 public constant COPPER = uint256(0) - the amount needed to get `COPPER` level;

- uint256 public constant BRONZE = uint256(10 ** (18+3)) - the amount needed to get `BRONZE` level;
- uint256 public constant SILVER = uint256(5*10 ** (18+3)) - the amount needed to get `SILVER` level;
- uint256 public constant GOLD = uint256(10 ** (18+4)) - the amount needed to get `GOLD` level;
- uint256 public constant PLATINUM = uint256(25*10 ** (18+3)) - the amount needed to get `PLATINUM` level;
- uint256 public constant DIAMOND = uint256(10 ** (18+5)) - the amount needed to get `DIAMOND` level;
- uint256 public constant CHAD = uint256(25*10 ** (18+4)) - the amount needed to get `CHAD` level;
- uint256 public constant JAWAD = uint256(10 ** (18+6)) - the amount needed to get `JAWAD` level;

Functions

TVKPrestige contract has following functions:

- *_afterSetStatus*

  Description

  Used to set a status of an account.

  Visibility

  internal

  Input parameters

  - address account;
  - Status oldStatus;
  - Status newStatus;
  - bytes memory;

  Constraints

  None

  Events emit

  None

  Output

  None

- *levels*

  Description

  Returns status level limits as array.

  Visibility

  public pure

  Input parameters

  - address account;
  - IPrestige.Status status;

  Constraints

  None

  Events emit

  None

  Output

  - uint256[9]

Audit overview

🟥 🟥 🟥 🟥 Critical

No critical issues were found.

🟨 🟨 🟨 High

No high issues were found.

🟩 🟩 Medium

No medium issues were found.

🟦 Low

No low severity issues were found.

🟦 Lowest / Code style / Best Practice

No lowest severity issues were found.

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 no issues during the 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 its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.