SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Master Ventures
Date: September 30th, 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 Master Ventures.
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/PAIDNetwork/ignition-staking
Commit	25beb6313186dc7bbdcbb60815381c369a3af578
Technical Documentation	NO
JS tests	YES
Timeline	21 SEPTEMBER 2021 – 30 SEPTEMBER 2021
Changelog	23 SEPTEMBER 2021 – INITIAL AUDIT 30 SEPTEMBER 2021 – SECOND REVIEW

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Master Ventures (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 September 15th, 2021 - September 23rd, 2021. The second code review conducted on September 30th, 2021.

Scope

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

Repository:
    https://github.com/PAIDNetwork/ignition-staking
Commit:
    25beb6313186dc7bbdcbb60815381c369a3af578
Technical Documentation: No
JS tests: Yes
Contracts:
    Staking.sol
    Apollostaking.sol
    Ignitionstaking.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 secured.

Insecure	**Poor secured**	**Secured**	**Well-secured**

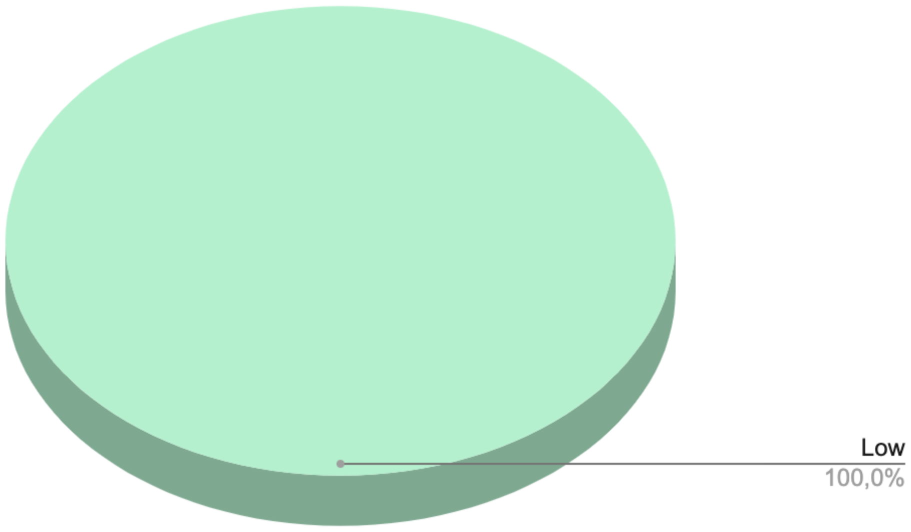
You are here

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 **4** low severity issues.

After the second review security engineers found **2** 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

No medium severity issues were found.

■ Low

1. **Vulnerability**: Block timestamp

   Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners. Both contracts are fully related on the block.timestamp

   Recommendation: Please consider relying on the block.number instead

2. A public function that could be declared external

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

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

   Fixed before the second review

3. Tautology or contradiction

   There are expressions that are tautologies or contradictions.

   Recommendation: remove tautologic comparison **period >= 0**

   Fixed before the second review

4. State variable could be constant
   State variables that never change their values should be declared constants to save gas

   Recommendation: Please consider declaring constants instead of initializing state variables in the initializer function

   Lines: Ignitionstaking.sol#17-51

```
uint256 lockup;
uint256[] rewardBaseRate;
uint256 burnMaxRate;
uint256 remnantRate;
uint256 public totalBurned;
address burnAddress;
```

```
/* @notice contract needs to be deployed with at least one token
* @param _tokenAddress address to ERC20 token
* @param _tokenReserve address to contain token reserves
* @param _symbol symbol for the token
* @param _precision used by token
*/
function initialize(
   address _tokenAddress,
   string memory _symbol,
   uint8 _precision,
   address _burnAddress) external initializer {

   require(_tokenAddress != address(0), "Invalid address for token");

   __Ownable_init();
   __Pausable_init();

   tokens[_tokenAddress] = Token(
       _tokenAddress,
       _symbol,
       _precision
   );

   lockup = 10 days;
   rewardBaseRate =  [ 15, 25, 35, 50, 60 ];
   burnMaxRate = 80;
   remnantRate = 5;
   burnAddress = _burnAddress;
}
```

Lines: Apollostaking.sol#16-37

```
uint256 lockup;

/* @notice contract needs to be deployed with at least one token
* @param _tokenAddress address to ERC20 token
* @param _symbol symbol for the token
* @param _precision used by token
*/
function initialize(
   address _tokenAddress,
   string memory _symbol,
   uint8 _precision) external initializer {

   require(_tokenAddress != address(0), "Invalid address for token");

   __Ownable_init();
   tokens[_tokenAddress] = Token(
       _tokenAddress,
       _symbol,
       _precision
   );

   lockup = 14 days;
```

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 **4** low severity issues.

After the second review security engineers found **2** low severity issues.

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 the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free 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 the security of smart contracts.

Technical Disclaimer

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