HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Plethori
Date: September 15th, 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 Plethori - Audit Report
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Staking
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git repository	https://github.com/Plethori/Staking-Contract/tree/7e21fbda4600545af336fd2d10910fca26e98388
Timeline	15 July 2021 - 15 September 2021
Changelog	15 July 2021 - INITIAL AUDIT 23 July 2021 - SECOND REVIEW 05 August 2021 - THIRD REVIEW 03 September 2021 - FOURTH REVIEW 15 September 2021 - FIFTH REVIEW 15 September 2021 - SIXTH REVIEW

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Plethori (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 15th, 2021 - July 19th, 2021. The second review conducted on July 23rd, 2021. The third review conducted on August 05th, 2021. The fourth review conducted on September 3rd, 2021. The fifth and sixth reviews conducted on September 15th, 2021.

Scope

The scope of the project is the smart contracts in Git Repository:

```
Repository:
      https://github.com/Plethori/Staking-Contract/tree
Commit:
      7e21fbda4600545af336fd2d10910fca26e98388
Scope:
      contracts/PLEstaking.sol
```

We have scanned these smart contracts 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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Asset's integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contract is well-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.

Security engineers found **4** high, **2** medium, **3** low and **5** informational issues during the first review.

Security engineers found **4** high, **2** medium, **1** low and **2** informational issues during the second review.

Security engineers found **1** high and **1** low issue during the third review.

Security engineers found **1** high and **1** low issue during the fourth review.

Security engineers found **1** high severity issue during the fifth review.

After the sixth review security engineers found all issues were **resolved**.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

■ ■ ■ ■ Critical

No Critical severity issues were found.

■ ■ ■ High

1. **Vulnerability:** Incorrect total rewards

While provided documentation says the following: "*Plethori provides the 10,000,000 PLE tokens for the reward of staking platform*", the provided smart contract reserves only **100,000 PLE** for rewards

Recommendation: Please consider updating either the contract or documentation to have the same values.

Fixed before the second review

2. **Vulnerability:** Incorrect rewards limitation

The smart contract will return more rewards than stated in the code. The reason is there is no check for overflowing the remainRewardToken by totalClaimedRewards in the **distributeToken** function, but there is a check totalClaimedRewards > remainRewardToken in the **getPendingDivs** which will return 0 as rewards for everyone who will attempt to check their rewards, but **only** in the case (as it seen from the logic above) when contract gave more rewards then there were reserved.

Recommendation: Please consider fully revising the limitation logic.

Second Review: The latest changes to the code added more issues. Right now, if there'll be a **pendingDivs** > **remainRewardToken** (that's possible because of the checking absence) and having math in the **unchecked** (!) code block, there will be an overflow and remainRewardToken will become a 2^{256} - (remainRewardToken-pendingDivs)
and will continue distributing a lot of tokens from the staked balance.

The **recommendation** is the same: please fully revise the logic of limiting rewards!

Fixed before the third review

3. **Vulnerability:** Rewards distributed from staked balance

The current logic implementation works the way rewards are being given from the tokens staked by stakeholders. As soon as we give all rewards, some stakeholders wouldn't have an ability to unstake because of lack of balance.

Recommendation: Please consider sending entire rewards amount in the constructor.

After the fifth review: The initializer function was added but it's not forced to be called ever. Please do not allow to stake until the contract is initialized.

Fixed before the sixth review

4. **Vulnerability:** Rewards loss

Unstacking leads to the loss of rewards. Previously earned but not distributed rewards would be lost for the amount of unstaked tokens

Recommendation: Please consider distributing rewards before decreasing **depositAmount**.

Fixed before the third review

5. **Vulnerability:** Incorrect use of unchecked block

Reading solidity documentation, we can see that the **unchecked** block is used to return the old (pre-0.8.0) behaviour of the integer math, where **2-3** will result in $2^{256}-1$, while simple **2-3**, without unchecked block, will result in revert in the solidity version >= 0.8.0.

Recommendation: Please consider removing unneeded **unchecked** blocks in the code

Fixed before the third review

Medium

1. **Vulnerability:** Ignored result of **sub** function

SafeMath.sub function doesn't modify the given values but returns the result instead. This result is ignored in the smart contract.

Recommendation: Please consider react on the result of the function or remove this function call from the code

Second review: After switching from using the SafeMath library this point became weirder. **balance1** is being assigned as the result of unchecked subtraction (that could result in underflow) and never used after that.

Fixed before the third review

2. **Vulnerability:** Ignored result of **add** function

SafeMath.add function doesn't modify the given values but returns the result instead. This result is ignored in the smart contract.

Recommendation: Please consider react on the result of the function or remove this function call from the code

Second review: After switching from using the SafeMath library this point became weirder. **balance2** is being assigned as the result of unchecked addition (that could result in overflow) and never used after that.

Fixed before the third review

Low

1. **Vulnerability:** Constructor visibility

Visibility for constructor is ignored starting solidity v0.7.0

Recommendation: Please consider removing constructor visibility

Fixed before the second review

2. **Vulnerability:** Missing zero address validation

Missing zero-address validation in the constructor

Recommendation: Please consider adding zero-address validation

Fixed before the second review

3. **Vulnerability:** Block timestamp

Dangerous usage of **block.timestamp**. **block.timestamp** can be manipulated by miners.

Recommendation: Please consider relying on **block.number** which is more strict

Second review: Just changing block.timestamp to block.number in the code doesn't do the trick. New block is not generated each second, so you couldn't rely on that 365 days will equal to 365 * 24 * 60 * 60 blocks!

Third review: in the getPendingDivs you're calculating timeDiff as the difference between block.number and the block number where last claim was done. But, if you want a reward rate of 40% per year you can't think of the network generates one block per second.

Recommendation: Please revise your rewards calculation logic. Also please consider adding tests that will cover the entire business logic and corner cases. Now tests don't check the rewards rate.

Fixed before the sixth review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Lowest / Code style / Best Practice

1. **Vulnerability:** Dead-code

There is a function **getBalance** which is declared private but never used.

Recommendation: Please consider removing or commenting out a dead-code

Fixed before the second review

2. **Vulnerability:** Too many digits

Literals with many digits are difficult to read and review

Recommendation: Please consider using ether suffix and the scientific notation. Example: **1e5 ether** or **100_000 ether**

Fixed before the second review

3. **Vulnerability:** Conformance to Solidity naming conventions

Solidity defines a naming convention that should be followed. Constants should be named with all capital letters with underscores separating words. Examples: **MAX_BLOCKS**, **TOKEN_NAME**, **TOKEN_TICKER**, **CONTRACT_VERSION**.

Fixed before the second review

4. **Vulnerability:** State variables that could be declared constant

Constant state variables should be declared constant to save gas.

Fixed before the second review

5. **Vulnerability:** Public function that could be declared external

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

Fixed before the second review

6. **Vulnerability:** Maximum line length

Solidity has a style guide convention that regulates the maximum line length.

Lines 260 and 269 are exceeding the maximum line length

Fixed before the third review

7. **Vulnerability:** Unused logic

Having state variable admin, which is set by setAdmin. Also, the setAdmin is the only function that uses onlyOwner modified. And this modifier is the only place where admin variable is used. So this vicious circle is not used in the code at all

Recommendation: Please remove entire admin/onlyOwner/setAdmin code, because it does nothing.

Fixed before the second review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Security engineers found **4** high, **2** medium, **3** low and **5** informational issues during the first review.

Security engineers found **4** high, **2** medium, **1** low and **2** informational issues during the second review.

Security engineers found **1** high and **1** low issue during the third review.

Security engineers found **1** high and **1** low issue during the fourth review.

Security engineers found **1** high severity issue during the fifth review.

After the sixth review security engineers found all issues were **resolved**.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 the 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.