HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Platypus Finance
Date: December 22nd, 2021

Document

Name	Smart Contract Code Review and Security Analysis Report for Platypus Finance.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; Pools
Platform	Avalanche / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/platypus-finance/core
Commit	5a997ea9ab26b4dfaf71e7c9bd8bb755f3f55ed1
Technical Documentation	YES
JS tests	YES
Website	Platypus.finance
Timeline	14 OCTOBER 2021 – 22 DECEMBER 2021
Changelog	20 OCTOBER 2021 – INITIAL AUDIT 22 DECEMBER 2021 – SECOND REVIEW

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Platypus Finance (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 October 14th, 2021 - October 20th, 2021.

Second review conducted on December 22nd, 2021.

Scope

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

Repository:
https://github.com/platypus-finance/core

Commit:
5a997ea9ab26b4dfaf71e7c9bd8bb755f3f55ed1

Technical Documentation: Yes (in repo: "/docs/")
JS tests: Yes (in repo: "/test/")
Contracts:
asset\AggregateAccount.sol
asset\Asset.sol
interfaces\IChainlinkAggregator.sol
interfaces\IPriceOracleGetter.sol
interfaces\IWETH.sol
interfaces\IAsset.sol
interfaces\IPool.sol
libraries\DSMath.sol
governance\Timelock.sol
oracle\ChainlinkProxyPriceProvider.sol
pool\Core.sol
pool\Pool.sol
pool\WETHForwarder.sol
test\IntegrationTest.sol
test\OwnableTestERC20.sol
test\TestChainlinkAggregator.sol
test\TestDSMath.sol
test\TestERC20.sol
test\TestSlippage.sol
test\TestWAVAX.sol
test\TestAggregateAccount99.sol
test\TestAssetV99.sol
test\TestPoolV99.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 well-secured.

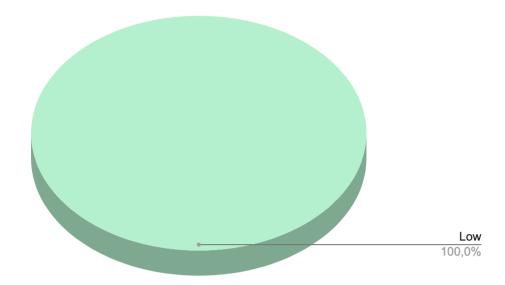| Insecure	Poor secured	Secured	Well-secured

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.

As a result of the audit, security engineers found **3** low severity issues.

After the second review security engineers found that the code was slightly changed, upgradable features were added to the assets, owner function to remove assets was added to the Pool contract, and some issues were addressed. Therefore **1** low severity issue was found.

Graph 1. The distribution of vulnerabilities after the audit.

Low
100,0%

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. No event on changing valuable addresses

When contact changes some valuable address it is strongly recommended to emit an event so the changes could be tracked off-chain.

Contracts: asset/Asset.sol#114; pool/Pool.sol(#233, #318); pool/WETHForwarder.sol#49

Functions: Asset.setPool; Pool.setDev; Pool.setPriceOracle; WETHForwarder.setPool

Recommendation: Please emit events on changing valuable addresses in contracts.

Status: Fixed

2. No event on changing valuable arithmetic values

When contact changes some values which are taking place in arithmetic calculations it is strongly recommended to emit an event so the changes could be tracked off-chain.

Contracts: asset/Asset.sol(#97, #203, #213, #229, #239); pool/Pool.sol (#259, #267, #275, #283, #292, #301, #310)

Functions: Asset.setMaxSupply; Asset.addCash; Asset.removeCash; Asset.addLiability; Asset.removeLiability; Pool.setSlippageParamK; Pool.setSlippageParamN; Pool.setC1; Pool.setXThreshold; Pool.setHaircutRate; Pool.setRetentionRatio; Pool.setMaxPriceDeviation
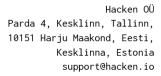
Recommendation: Please emit events on changing valuable addresses in contracts.

Status: Fixed

3. Calls inside a loop

Calls inside a loop might lead to a denial-of-service
attack.
If one of the destinations has a fallback function that reverts, the
top-caller function will always revert.

Contracts: pool/Pool.sol

Functions: getEquilibriumCoverageRatio

Recommendation: Try to find math/storage solution instead of calls
inside the loop. For example, you may want to store asset decimals in
the mapping and not call to its function "decimals()".

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 **3** low severity issues.

After the second review security engineers found that the code was slightly changed, upgradable features were added to the assets, owner function to remove assets was added to the Pool contract, and some issues were addressed. Therefore **1** low severity issue was found.

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.