Smart Contract Security Audit Report

Prepared for Platypus Finance

Prepared by Supremacy

January 3, 2024

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the Platypus Finance, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Client

Platypus Finance protocol started as a single-side AMM (decentralized exchange) designed for exchanging stable cryptocurrencies (ERC20 tokens) on the Avalanche blockchain. They redefined stableswaps and reinvented stablecoins. They combine both stableswap and stablecoin, masterfully utilizing Platypus' underlying assets.

Item	Description
Client	Platypus Finance
Website	https://platypus.finance
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

- Repository: https://github.com/platypus-finance/core/tree/master/contracts
- Commit Hash: edef48f0959acd86f84e7cd8a5346dbb004b60a8

Below are the files in scope for this security audit and their corresponding MD5 hashes.

Filename	MD5
`./asset/AggregateAccount.sol`	`aa3aadd6b520d78ab8b2aff0a7aefbed`
`./asset/Asset.sol`	`0790e10ecd972152f25fe2a8a0fec140`
`./libraries/DSMath.sol`	`bab754e177db1ea23480204d3e7b92b8`
`./oracle/ChainlinkProxyPriceProvider.sol`	`1b70975cc87e65de9a49815b6aea3132`
`./pool/Core.sol`	`ffb7f2ebbfaf3972842e35e51daeb3e5`
`./pool/Pool.sol`	`af11cb35466cf3e932c422fd0ffa1c00`
`./pool/PoolSAvax.sol`	`394e9fe3c668a80b3f81236dc01f71a3`
`./pool/PoolSecondaryPure.sol`	`c8dfab7838fd45d6fe5a7ccda0789c68`
`./pool/WETHForwarder.sol`	`08c14801eae2f1a4ef88b37d55de2b72`
`./router/PlatypusRouter02.sol`	`334dcfea05e6145f67f69e5c599c6590`

1.3 Changelogs

Version	Date	Description
0.1	December 15, 2023	Initial Draft
1.0	January 3, 2024	Final Release

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology precipitation and innovative research.

We are reachable at Twitter (https://twitter.com/SupremacyHQ), or Email (contact@supremacy.email).

1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice
- Impact specifies the technical and business-related consequences of a finding
- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Severity

Impact			
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low
	High	*Medium*	*Low*

Likelihood

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Medium	Centralization risk	Confirmed
2	Low	The potential unsafe external call	Confirmed

2.1 Medium

1. Centralization risk [Medium]

Severity: Medium Likelihood: Low Impact: High

Status: Confirmed

Description:

In the Platypus Finance protocol, there is a privilege account, which has the right to directly transfer a specific asset in the liquidity pool.

Our analysis shows that privileged accounts need to be scrutinized. In the following, we will examine privileged accounts and the associated privileged access in the current contract.

Note that if the privileged owner account is a plain EOA, this may be worrisome and pose counter-party risk to the protocol users. A multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO. In the meantime, a timelock-based mechanism can also be considered as mitigation.

```
    /**
     * @notice Recover any funds mistakingly sent to this contract
     * @param token the address of the token to retrieve
     */
    function recoverUserFunds(address token) external onlyDev {
        uint256 currentBalance = IERC20(token).balanceOf(address(this));
        IERC20(token).safeTransfer(msg.sender, currentBalance);
    }
```

Pool.sol

Recommendation: Initially onboarding could can use multisign wallets or timelocks to initially mitigate centralization risks, but as a long-running protocol, we recommend eventually transfer the privileged account to the intended DAO-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks.

Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

2.2 Low

2. The potential unsafe external call [Low]

`Severity: Low` `Likelihood: Low` `Impact: Low`

`Status: Confirmed`

Description:

`Poolsavax::swapToETH()` is used to swap `fromToken` to native token, But the implementation of this function is slightly different from other `swap` functions. Because, in the swap logic implemented by `swapToETH` function, it does not follow Checks-Effects-Interactions Pattern. The unsafe external calls here are between `removeCash()` & `addLiability()`. At the same time, the transfer of native token assets is carried out through the `sendValue()` function in `Address` Utils. Its bottom layer is built by `call` and does not impose any restrictions on Gas, therefore, this is a potential reentrancy opportunity. However, the `Poolsavax` contract itself uses the `ReentrancyGuard` module to prevent the exploit of a single contract, it cannot be used directly in the pool contract. But it may still be a potential security hazard.

```
    /**
     * @notice Swap fromToken for ETH, ensures deadline and minimumToAmount and sends quoted amount to `to` address
     * @param fromToken The token being inserted into Pool by user for swap
     * @param fromAmount The amount of from token inserted
     * @param minimumToAmount The minimum amount that will be accepted by user as result
     * @param to The user receiving the result of swap
     * @param deadline The deadline to be respected
     * @return actualToAmount The actual amount user receive
     * @return haircut The haircut that would be applied
     */
    function swapToETH(
        address fromToken,
        uint256 fromAmount,
        uint256 minimumToAmount,
        address payable to,
        uint256 deadline
    ) external ensure(deadline) nonReentrant whenNotPaused returns (uint256 actualToAmount, uint256 haircut) {
        require(fromToken != address(0), 'Z_ADD');
        require(fromToken != weth, 'SAME_ADD');

        IERC20 fromERC20 = IERC20(fromToken);
        Asset fromAsset = _assetOf(fromToken);
        require(address(fromAsset) != address(0), 'ASST_N_EX');
        Asset toAsset = _assetOf(weth);

        require(toAsset.aggregateAccount() == fromAsset.aggregateAccount(), 'DIFF_AGG_ACC');

        (actualToAmount, haircut) = _quoteFrom(fromAsset, toAsset, fromAmount);
        require(minimumToAmount <= actualToAmount, 'AM_TOO_LOW');

        fromERC20.safeTransferFrom(address(msg.sender), address(fromAsset), fromAmount);
        fromAsset.addCash(fromAmount);
```

```
        toAsset.removeCash(actualToAmount);
        toAsset.transferUnderlyingToken(address(wethForwarder),
actualToAmount);
        wethForwarder.unwrapAndTransfer(to, actualToAmount);
        toAsset.addLiability(_dividend(haircut, _retentionRatio));

        emit Swap(msg.sender, fromToken, weth, fromAmount, actualToAmount, to);
    }
```

PoolSAvax.sol

```
    /**
     * @notice Unwrap and transfer eth. Can only be called by pool
     * @param to address receiving
     * @param amount total amount to be transferred
     */
    function unwrapAndTransfer(address payable to, uint256 amount) external
onlyPool nonReentrant {
        IWETH _weth = IWETH(weth);
        require(_weth.balanceOf(address(this)) >= amount, 'INSUFFICIENT_WETH');
        _weth.withdraw(amount);
        to.sendValue(amount);
    }
```

WETHForwarder.sol

Recommendation: Follow Checks-Effects-Interactions Pattern.

3 Disclaimer

This security audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This security audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contract(s). Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.