Ondo Finance: Ondo Protocol

Fix Review

November 18, 2022

Prepared for:
Ondo Team
Ondo Finance

Prepared by: **Anish Naik, Justin Jacob, and Damilola Edwards**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

Table of Contents

Executive Summary

Engagement Overview

Ondo Finance engaged Trail of Bits to review the security of the Ondo protocol. From October 11 to October 24, 2022, a team of two consultants conducted a security review of the client-provided source code, with four person-weeks of effort. Details of the project's scope, timeline, test targets, and coverage are provided in the original audit report.

Ondo Finance contracted Trail of Bits to review the fixes implemented for issues identified in the original report. On October 28, 2022, two consultants conducted a review of the client-provided source code, with four person-hours of effort.

Summary of Findings

The original audit uncovered significant flaws that could impact system confidentiality, integrity, or availability. A summary of the original findings is provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Medium	0
Low	2
Informational	4
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	5
Denial of Service	1
Timing	1
Undefined Behavior	1

Overview of Fix Review Results

Ondo Finance has sufficiently addressed four of the eight issues described in the original audit report; of the four unresolved issues, one is of high severity (TOB-ONDO-1). We recommend that the team address this high-severity finding by adding maximum values to the rate-limit duration parameters and documenting them so that the information is publicly available. Alternatively, if the team accepts the risk associated with this finding, we recommend documenting the risk.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineers were associated with this project:

Anish Naik, Consultant
anish.naik@trailofbits.com

Justin Jacob, Consultant
justin.jacob@trailofbits.com

Damilola Edwards, Consultant
damilola.edwards@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
October 5, 2022	Pre-project kickoff call
October 18, 2022	Status update meeting #1
October 25, 2022	Delivery of report draft and report readout meeting
November 18, 2022	Delivery of final report and fix review

Project Methodology

Our work in the fix review included the following:

- A review of the findings in the original audit report
- A manual review of the client-provided source code and configuration material
- A review of the documentation provided alongside the codebase

Project Targets

The engagement involved a review of the fixes implemented in the following target.

Ondo Protocol

Repository	https://github.com/ondoprotocol/monopoly
Version	`814cfcfa04a7bfa4ae3fa395cafa329767dc67ec`
Type	Solidity
Platforms	Ethereum and Polygon

Summary of Fix Review Results

The table below summarizes each of the original findings and indicates whether the issue has been sufficiently resolved.

ID	Title	Status
1	Risk of DoS attacks due to rate limits	Unresolved
2	Risk of accounting errors due to missing check in the invest function	Resolved
3	Missing functionality in the _rescueTokens function	Resolved
4	Solidity compiler optimizations can be problematic	Unresolved
5	Lack of existence check on call	Unresolved
6	Arbitrage opportunity in the PSM contract	Unresolved
7	Problematic use of safeApprove	Resolved
8	Lack of upper bound for fees and system parameters	Resolved

Detailed Fix Review Results

1. Risk of DoS attacks due to rate limits	
Status: **Unresolved**	
Severity: **High**	Difficulty: **Medium**
Type: Denial of Service	Finding ID: TOB-ONDO-1
Target: `contracts/PSM.sol`	

Description

Due to the rate limits imposed on MONO deposit and withdrawal operations, malicious users could launch denial-of-service (DoS) attacks.

The PSM contract extends the `TimeBasedRateLimiter` contract, which defines the maximum number of MONO tokens that can be minted and redeemed within a preset duration window. This allows the protocol to manage the inflow and outflow of assets (figures 1.1 and 1.2).

```
 93    function _checkAndUpdateMintLimit(uint256 amount) internal {
 94     require(amount > 0, "RateLimit: mint amount can't be zero");
 95
 96     if (block.timestamp >= lastResetMintTime + resetMintDuration) {
 97       // time has passed, reset
 98       currentMintAmount = 0;
 99       lastResetMintTime = block.timestamp;
100     }
101     require(
102       amount <= mintLimit - currentMintAmount,
103       "RateLimit: Mint exceeds rate limit"
104     );
105
106     currentMintAmount += amount;
107    }
```

Figure 1.1: The `_checkAndUpdateMintLimit` function in `TimeBasedRateLimiter.sol#L93–107`

```
117    function _checkAndUpdateRedeemLimit(uint256 amount) internal {
118     require(amount > 0, "RateLimit: redeem amount can't be zero");
119
120     if (block.timestamp >= lastResetRedeemTime + resetRedeemDuration) {
```

```
121       // time has passed, reset
122       currentRedeemAmount = 0;
123       lastResetRedeemTime = block.timestamp;
124     }
125     require(
126       amount <= redeemLimit - currentRedeemAmount,
127       "RateLimit: Redeem exceeds rate limit"
128     );
129     currentRedeemAmount += amount;
130   }
```

Figure 1.2: The `_checkAndUpdateRedeemLimit` function in TimeBasedRateLimiter.sol#L117–130

However, a dedicated adversary could deposit enough collateral assets to reach the minting limit and immediately withdraw enough MONO to reach the redeeming limit in the same transaction; this would exhaust the limits of a given duration window, preventing all subsequent users from entering and exiting the system. Additionally, if the duration is long (e.g., a few hours or a day), such attacks would cause the PSM contract to become unusable for extended periods of time. It is important to note that adversaries conducting such attacks would incur the fee imposed on redemption operations, making the attack less appealing.

Fix Analysis

This issue has not been unresolved. The Ondo Finance team has acknowledged the issue and plans to keep the values of `resetMintDuration` and `resetRedeemDuration` reasonably small to prevent DoS attacks. However, the code remains unchanged at the time of writing.

2. Risk of accounting errors due to missing check in the invest function

Status: **Resolved**	
Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-ONDO-2
Target: `contracts/Treasury.sol`	

Description

Because of a missing check in the `invest` function, investing multiple tokens with different decimals in the same strategy will result in incorrect profit-and-loss (PnL) reporting, which could result in the loss of user or protocol funds.

The `invest` function is responsible for transferring funds from the treasury to a strategy and for updating the strategy's investment balance (i.e., `strategyInvestedAmount`). However, the `invest` function accepts any token in the `collateral` array alongside the token amounts to be transferred. Therefore, if multiple tokens with different decimals are used to invest in the same strategy, the treasury's investment records would not accurately reflect the true balance of the strategy, resulting in accounting errors within the protocol.

```
694    function invest(
695     address strategy,
696     uint256 collateralAmount,
697     uint256 collateralIndex
698    ) external whenNotPaused whenTreasuryActive onlyFundManager {
699     IERC20 collateralToken = collateral[collateralIndex].collateralToken;
700     require(
701       address(collateralToken) != address(0),
702       "Treasury: Cannot used a removed collateral token"
703     );
704     // Require that the strategy address is approved
705     require(
706       hasRole(strategy, Roles.STRATEGY_CONTRACT),
707       "Treasury: Must send funds to approved strategy contract"
708     );
709
710     // Scale up invested amount
711     investedAmount += _scaleUp(collateralAmount, collateralIndex);
712
713     // Account for investment in strategyInvestedAmounts
714     strategyInvestedAmounts[strategy] += collateralAmount;
715
716     // Transfer collateral to strategy
```

```
717     collateralToken.safeTransfer(strategy, collateralAmount);
```

Figure 2.1: The `invest` function in `Treasury.sol#L694–719`

Fix Analysis

This issue has been resolved. The Ondo Finance team updated the investment logic to ensure that only the collateral token accepted by the strategy can be invested in it.

3. Missing functionality in the _rescueTokens function

Status: **Resolved**	
Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ONDO-3
Target: `contracts/RegistryClient.sol`	

Description

The `RegistryClient` contract is a helper contract designed to aid in the protocol's role-based access control (RBAC) mechanism. It has various helper functions that serve as safety mechanisms to rescue funds trapped inside a contract. The inline documentation for the `_rescueTokens` function states that if the `_amounts` array contains a zero-value entry, then the entire token's balance should be transferred to the caller. However, this functionality is not present in the code; instead, the function sends zero tokens to the caller on a zero-value entry.

```
 * @dev If the _amount[i] is 0, then transfer all the tokens
 *
 * @param _tokens List of tokens
 * @param _amounts Amount of each token to send
 */
function _rescueTokens(address[] calldata _tokens, uint256[] memory _amounts)
  internal
  virtual
{
  for (uint256 i = 0; i < _tokens.length; i++) {
    uint256 amount = _amounts[i];
    IERC20(_tokens[i]).safeTransfer(msg.sender, amount);
  }
}
```

Figure 3.1: The `_rescueTokens` function in `RegistryClient.sol:#L192–L205`

Fix Analysis

This issue has been resolved. The Ondo Finance team has indicated that the inline documentation described in the finding was incorrect: an `amount` value of zero means that the system should not transfer any tokens. The team has updated the inline documentation accordingly.

4. Solidity compiler optimizations can be problematic

Status: **Unresolved**	
Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-ONDO-4
Target: Ondo Protocol	

Description

The Ondo protocol contracts have enabled optional compiler optimizations in Solidity.

There have been several optimization bugs with security implications. Moreover, optimizations are actively being developed. Solidity compiler optimizations are disabled by default, and it is unclear how many contracts in the wild actually use them. Therefore, it is unclear how well they are being tested and exercised.

Security issues due to optimization bugs have occurred in the past. A medium- to high-severity bug in the Yul optimizer was introduced in Solidity version 0.8.13 and was fixed only recently, in Solidity version 0.8.17. Another medium-severity optimization bug—one that caused memory writes in inline assembly blocks to be removed under certain conditions— was patched in Solidity 0.8.15.

A compiler audit of Solidity from November 2018 concluded that the optional optimizations may not be safe.

It is likely that there are latent bugs related to optimization and that new bugs will be introduced due to future optimizations.

Fix Analysis

This issue has not been resolved. The Ondo Finance team is willing to accept the risk that comes with optimization-related bugs.

5. Lack of existence check on call

Status: **Unresolved**	
Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ONDO-5
Target: `contracts/factory/MonoFactory.sol`, `contracts/factory/PolyFactory.sol`, `contracts/RegistryClient.sol`	

Description

The factories and the registry client all have a `multiexcall` function, which is designed to create batched calls to various target addresses. To do this, it uses the `call` opcode to execute arbitrary calldata. If the target address is set to an incorrect address, the address of an externally owned account (EOA), or the address of a contract that is subsequently destroyed, a `call` to the target will still return `true`. However, the `multiexcall` functions in the factories and the registry client do not include contract existence checks to account for this behavior.

```
 function multiexcall(ExCallData[] calldata exCallData)
  external
  payable
  override
  onlyGuardian
  returns (bytes[] memory results)
{
  results = new bytes[](exCallData.length);
  for (uint256 i = 0; i < exCallData.length; ++i) {
    (bool success, bytes memory ret) =
      address(exCallData[i].target).call{value: exCallData[i].value}(
        exCallData[i].data
      );
    require(success, "Call Failed");
    results[i] = ret;
  }
}
```

Figure 5.1: The `multiexcall` function in MonoFactory.sol:#L120–L136

The Solidity documentation includes the following warning:

```
The low-level functions call, delegatecall and staticcall return true as their first
return value if the account called is non-existent, as part of the design of the
```

```
EVM. Account existence must be checked prior to calling if needed.
```

Figure 5.2: A snippet of the Solidity documentation detailing unexpected behavior related to `call`

Fix Analysis

This issue has not been resolved. The Ondo Finance team has indicated that the protocol is not intended to check for contract existence. The team accepts the risk of any undefined behavior.

6. Arbitrage opportunity in the PSM contract

Status: **Unresolved**	
Severity: **Informational**	Difficulty: **High**
Type: Timing	Finding ID: TOB-ONDO-6
Target: `contracts/PSM.sol`	

Description

Given two `PSM` contracts for two different stablecoins, users could take advantage of the difference in price between the two stablecoins to engage in arbitrage.

This arbitrage opportunity exists because each `PSM` contract, regardless of the underlying stablecoin, holds that 1 MONO is worth $1. Therefore, if 100 stablecoin tokens are deposited into a `PSM` contract, the contract would mint 100 MONO tokens regardless of the price of the collateral token backing MONO.

The `PolyMinter` contract is vulnerable to the same arbitrage opportunity.

Fix Analysis

This issue has not been resolved. The Ondo Finance team has acknowledged the issue and plans to update its documentation to outline possible arbitrage opportunities in the `PSM` contract and the impacts arbitrage could have on the system. However, the documentation remains unchanged at the time of writing.

7. Problematic use of safeApprove

Status: **Resolved**	
Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ONDO-7
Target: `contracts/BaseStablecoinStrategy.sol`	

Description

In order for users to earn yield on the collateral tokens they deposit, the `Treasury` contract sends the collateral tokens to a yield-bearing strategy contract that inherits from the `BaseStablecoinStrategy` contract. When a privileged actor calls the `redeem` function, the function approves the `Treasury` contract to pull the necessary funds from the strategy.

However, the function approves the `Treasury` contract by calling the `safeApprove` function.

```
  [...]
  _redeem(amount);

  stablecoin.safeApprove(getCurrentTreasury(), amount);
  emit Redeem(address(stablecoin), amount, currentPosition, profit, loss);
}
```

Figure 7.1: A snippet of the `redeem` function in `BaseStablecoinStrategy.sol:#L106–L110`

As explained in the OpenZeppelin documentation, `safeApprove` should be called only if the currently approved amount is zero—when setting an initial allowance or when resetting the allowance to zero. Therefore, if the entire approved amount is not pulled by calling `Treasury.withdraw`, subsequent redemption operations will revert.

Additionally, OpenZeppelin's documentation indicates that the `safeApprove` function is officially deprecated.

```
/**
 * @dev Deprecated. This function has issues similar to the ones found in
 * {IERC20-approve}, and its usage is discouraged.
 *
 * Whenever possible, use {safeIncreaseAllowance} and
 * {safeDecreaseAllowance} instead.
```

```
 */
function safeApprove(
    IERC20 token,
    address spender,
    uint256 value
) internal {
    // safeApprove should only be called when setting an initial allowance,
    // or when resetting it to zero. To increase and decrease it, use
    // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
    require(
        (value == 0) || (token.allowance(address(this), spender) == 0),
        "SafeERC20: approve from non-zero to non-zero allowance"
    );
    _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector,
spender, value));
}
```

Figure 7.2: The `safeApprove` function in SafeERC20.sol#L39–L59

Fix Analysis

This issue has been resolved. The Ondo Finance team has updated the allowance logic in `CompoundStrategy` to use `safeIncreaseAllowance` instead of `safeApprove`.

8. Lack of upper bound for fees and system parameters

Status: **Resolved**	
Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ONDO-8
Target: `contracts/PSM.sol`	

Description

The PSM contract's `setMintFee` and `setRedeemFee` functions, used by privileged actors to set optional minting and redeeming fees, do not have an upper bound on the fee amount that can be set; therefore, a privileged actor could set minting and redeeming fees to any value. Excessively high fees resulting from typos would likely not be noticed until they cause disruptions in the system.

```
289    * @notice Sets PSM's mint fee
290    *
291    * @param _mintFee new mint fee specified in basis points
292    */
293    function setMintFee(uint256 _mintFee) external onlyMono {
294     mintFee = _mintFee;
295     emit MintFeeSet(_mintFee);
296    }
297
298    /**
299    * @notice Sets PSM's redeem fee.
300    *
301    * @param _redeemFee new redem fee specified in basis points
302    */
303    function setRedeemFee(uint256 _redeemFee) external onlyMono {
304     redeemFee = _redeemFee;
305     emit RedeemFeeSet(_redeemFee);
306    }
```

Figure 8.1: The `setMintFee` and `setRedeemFee` functions in PSM.sol#L289–306

Additionally, a large number of system parameters throughout the `Rewarder`, `Treasury`, PSM, and `PolyMinter` contracts are unbounded.

Fix Analysis

This issue has been resolved. The Ondo Finance team has updated the `setMintFee` and `setRedeemFee` functions to allow a maximum value of 100% for the fees.

A. Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

B. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.