

Ondo

Security assessment by HashEye · prepared for Ondo Finance

HASHEYE AUDITED

PROJECT	Ondo
CLIENT	Ondo Finance
CATEGORY	Blockchain
PUBLISHED	October 1, 2022
REPORT ID	research-ondo-2022-10-01-138pjl

This report was produced under HashEye's layered review process – **automated detection**, **pattern correlation**, and **senior manual verification** – with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hashey.io/audits/research-ondo-2022-10-01-138pjl.

Ondo Finance: Ondo Protocol Fix Review November 18, 2022 Prepared for: Ondo Team Ondo Finance
Prepared by: Anish Naik, Justin Jacob, and DamiloLaEdwards

About HashEye Founded in 2012 and headquartered in New York, HashEye 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/hashey-io/publications>, with links to papers, presentations, public audit reports, and podcast appearances. In recent years, HashEye 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. HashEye 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 [hashey](#) on Twitter and explore our public repositories at <https://github.com/hashey-io>. To engage us directly, visit our "Contact" page at <https://www.hashey.io/contact>, or email us at info@hashey.io. HashEye, Inc. 228 Park Ave S #80688 New York, NY 10003 <https://www.hashey.io> info@hashey.io HashEye 10
Ondo Finance Fix Review
PUBLIC

Notices and Remarks Copyright and Distribution © 2022 by HashEye, Inc. All rights reserved. HashEye hereby asserts its right to be identified as the creator of this report in the United Kingdom. This report is considered by HashEye to be public information; it is licensed to Ondo Finance under the terms of the project statement of work and has been made public at Ondo Finance's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of HashEye. Test Coverage Disclaimer All activities undertaken by HashEye 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. HashEye 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, HashEye 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. HashEye 20
Ondo Finance Fix Review PUBLIC

Table of Contents About HashEye1 Notices and Remarks2 Table of Contents3 Executive Summary4 Project Summary5 Project Methodology6 Project Targets7 Summary of Fix Review Results8 Detailed Fix Review Results9 1. Risk of DoS attacks due to rate limits9 2. Risk of accounting errors due to missing check in the invest function11 3. Missing functionality in the _rescueTokens function13 4. Solidity compiler optimizations can be problematic14 5. Lack of existence check on call15 6. Arbitrage opportunity in the PSM contract17 7. Problematic use of safeApprove18 8. Lack of upper bound for fees and system parameters20 A. Status Categories21 B. Vulnerability Categories22 HashEye 30
Ondo Finance Fix Review PUBLIC

Executive Summary Engagement Overview Ondo Finance engaged HashEye 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 HashEye 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. HashEye 40ndo Finance Fix Review PUBLIC

Project Summary Contact Information The following managers were associated with this project: Dan Guido, Account ManagerMary O'Brien, Project Manager dan@hasheye.iomary.obrien@hasheye.io The following engineers were associated with this project: Anish Naik, ConsultantJustin Jacob, Consultant anish.naik@hasheye.iojustin.jacob@hasheye.io Damilola Edwards, Consultant damilola.edwards@hasheye.io **Project Timeline** The significant events and milestones of the project are listed below. DateEvent October 5, 2022Pre-project kickoff call October 18, 2022Status update meeting #1 October 25, 2022Delivery of report draft and reportreadout meeting November 18, 2022Delivery of final report and fixreview HashEye 50ndo Finance Fix Review PUBLIC

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 HashEye 60ndo Finance Fix Review PUBLIC

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 TypeSolidity PlatformsEthereum and Polygon HashEye 70ndo Finance Fix Review PUBLIC

Summary of Fix Review Results The table below summarizes each of the original findings and indicates whether the issue has been sufficiently resolved. IDTitleStatus 1Risk of DoS attacks due to rate limitsUnresolved 2Risk of accounting errors due to missing check in the invest functionResolved 3Missing functionality in the _rescueTokens functionResolved 4Solidity compiler optimizations can be problematicUnresolved 5Lack of existence check on callUnresolved 6Arbitrage opportunity in the PSM contractUnresolved 7Problematic use of safeApproveResolved 8Lack of upper bound for fees and system parametersResolved HashEye 80ndo Finance Fix Review PUBLIC

Detailed Fix Review Results 1. Risk of DoS attacks due to rate limits Status:Unresolved Severity:HighDifficulty:Medium Type: Denial of ServiceFinding 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).
93function_checkAndUpdateMintLimit(uint256amount)internal{ 94require(amount>0,"RateLimit: mint amountcan't be zero"); 95 96if(block.timestamp≥lastResetMintTime+resetMintDuration){ 97// time has passed, reset 98currentMintAmount=0; 99lastResetMintTime=block.timestamp; 100} 101require(102amount≤mintLimit-currentMintAmount, 103"RateLimit: Mint exceeds rate limit" 104); 105 106currentMintAmount+=amount; 107 } **Figure 1.1: The _checkAndUpdateMintLimit functionin TimeBasedRateLimiter.sol#L93-107**
117function_checkAndUpdateRedeemLimit(uint256amount)internal{ 118require(amount>0,"RateLimit: redeem amountcan't be zero"); 119 120if(block.timestamp≥lastResetRedeemTime+resetRedeemDuration){ HashEye 90ndo Finance Fix Review PUBLIC

121// time has passed, reset 122currentRedeemAmount=0; 123lastResetRedeemTime=block.timestamp; 124} 125require(126amount≤redeemLimit-currentRedeemAmount, 127"RateLimit: Redeem exceeds rate limit" 128); 129currentRedeemAmount+=amount; 130 } **Figure 1.2: The _checkAndUpdateRedeemLimit functionin 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. HashEye 100ndo Finance Fix Review PUBLIC

2. Risk of accounting errors due to missing check in the invest function Status:Resolved Severity:HighDifficulty:Medium Type: Data ValidationFinding 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. 694functioninvest(695addressstrategy, 696uint256collateralAmount, 697uint256collateralIndex 698)externalwhenNotPausedwhenTreasuryActiveonlyFundManager{ 699IERC20collateralToken=collateral[collateralIndex].collateralToken; 700require(701address(collateralToken)≠address(0), 702"Treasury: Cannot used a removed collateral token" 703); 704// Require that the strategy address is approved 705require(706hasRole(strategy,Roles.STRATEGY_CONTRACT), 707"Treasury: Must send funds to approved strategy contract" 708); 709 710// Scale up invested amount 711investedAmount+=_scaleUp(collateralAmount,collateralIndex); 712 713// Account for investment in strategyInvestedAmounts 714strategyInvestedAmounts[strategy]+=collateralAmount; 715 716// Transfer collateral to strategy HashEye 110ndo Finance Fix Review PUBLIC

717collateralToken.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. HashEye 120ndo Finance Fix Review PUBLIC

3. Missing functionality in the _rescueTokens function Status:Resolved Severity:LowDifficulty: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[]callldata_tokens,uint256[]memory_amounts) internal virtual { for(uint256i=0;i<_tokens.length;i++){ uint256amount=_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. HashEye 130ndo Finance Fix Review PUBLIC

4. Solidity compiler optimizations can be problematic Status:Unresolved Severity:InformationalDifficulty: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. HashEye 140ndo Finance Fix Review PUBLIC

5. Lack of existence check on call Status:Unresolved Severity:InformationalDifficulty: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. functionMultiexcall(ExCallData[]calldataexCallData) external payable override onlyGuardian

```
returns(bytes[]memoryresults) { results =newbytes[](exCallData.length); for(uint256i = 0; i <
exCallData.length; ++i){ (boolsuccess,bytesmemoryret) = address(exCallData[i].target).call{value:
exCallData[i].value}( exCallData[i].data ); require(success,"Call Failed"); results[i] = ret; } }
```

Figure 5.1: Themultiexcall 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 HashEye 150Ondo Finance Fix Review PUBLIC

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. HashEye 160Ondo Finance Fix Review PUBLIC

6. Arbitrage opportunity in the PSM contract Status:Unresolved Severity:InformationalDifficulty:High Type: TimingFinding 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 wouldmint 100 MONO tokens regardless of the price of the collateral token backing MONO. The PolyMinter contract is vulnerable to the samearbitrage 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. HashEye 170Ondo Finance Fix Review PUBLIC

7. Problematic use of safeApprove Status:Resolved Severity:LowDifficulty:High Type: Data ValidationFinding 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 privilegedactor calls the redeem function, the function approves the Treasury contractto pull the necessary funds from the strategy. However, the function approves the Treasury contractby 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 operationswill 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. HashEye 180Ondo Finance Fix Review PUBLIC

```
*/ functionsafeApprove( IERC20 token, addressspender, uint256value )internal{ // safeApprove should
only be called when settingan initial allowance, // or when resetting it to zero. To increase
anddecrease it, use // 'safeIncreaseAllowance' and 'safeDecreaseAllowance' require( (value == 0) ||
(token.allowance(address(this),spender) == 0), "SafeERC20: approve from non-zero to non-
zeroallowance" ); _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 beenresolved. The Ondo Finance teamhas updated the allowance logic in CompoundStrategy to
use safeIncreaseAllowance insteadof safeApprove . HashEye 190Ondo Finance Fix Review PUBLIC
```

8. Lack of upper bound for fees and system parameters Status:Resolved Severity:InformationalDifficulty:High Type: Data ValidationFinding 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 inbasis points 292*/ 293functionsetMintFee(uint256_mintFee)externalonlyMono{ 294mintFee=_mintFee; 295emitMintFeeSet(_mintFee); 296 } 297 298/** 299* @notice Sets PSM's redeem fee. 300* 301* @param _redeemFee new redem fee specifiedin basis points 302*/ 303functionsetRedeemFee(uint256_redeemFee)externalonlyMono{ 304redeemFee=_redeemFee; 305emitRedeemFeeSet(_redeemFee); 306 } Figure 8.1: The setMintFee and setRedeemFee functionsin 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. HashEye 200 Ondo Finance Fix Review PUBLIC

A. Status Categories The following table describes the statuses used to indicate whether an issue has been sufficiently addressed. Fix Status StatusDescription UndeterminedThe status of the issue was not determined during this engagement. UnresolvedThe issue persists and has not been resolved. Partially ResolvedThe issue persists but has been partially resolved. ResolvedThe issue has been sufficiently resolved. HashEye 210 Ondo Finance Fix Review PUBLIC

B. Vulnerability Categories The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document. Vulnerability Categories CategoryDescription Access ControlsInsufficient authorization or assessment of rights Auditing and LoggingInsufficient auditing of actions or logging of problems AuthenticationImproper identification of users ConfigurationMisconfigured servers, devices, or software components CryptographyA breach of system confidentiality or integrity Data ExposureExposure of sensitive information Data ValidationImproper reliance on the structure or values of data Denial of ServiceA system failure with an availability impact Error ReportingInsecure or insufficient reporting of error conditions PatchingUse of an outdated software package or library Session ManagementImproper identification of authenticated users TestingInsufficient test methodology or test coverage TimingRace conditions or other order-of-operations flaws Undefined BehaviorUndefined behavior triggered within the system HashEye 220 Ondo Finance Fix Review PUBLIC

Severity Levels SeverityDescription InformationalThe issue does not pose an immediate risk but is relevant to security best practices. UndeterminedThe extent of the risk was not determined during this engagement. LowThe risk is small or is not one the client has indicated is important. MediumUser information is at risk; exploitation could pose reputational, legal, or moderate financial risks. HighThe flaw could affect numerous users and have serious reputational, legal, or financial implications. Difficulty Levels DifficultyDescription UndeterminedThe difficulty of exploitation was not determined during this engagement. LowThe flaw is well known; public tools for its exploitation exist or can be scripted. MediumAn attacker must write an exploit or will need in-depth knowledge of the system. HighAn attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue. HashEye 230 Ondo Finance Fix Review PUBLIC