Finalized Security Assessment

Eonian

Jun 13, 2024

This smart contract audit report was created by Bunzz Audit. It utilized a database of over 1000 contract vulnerability patterns, comparing the project's contract against this database with AI to conduct a comprehensive diagnosis of vulnerabilities

BUNZZ AUDIT

Table of Contents

Summary

The smart contract audit revealed several security issues that could potentially impact the integrity and functionality of the contract. Key vulnerabilities include the lack of validation for receiver addresses during asset transfers and minting, which could result in token loss; a Denial of Service (DoS) vulnerability due to an unbounded loop in the withdrawal process; and precision loss in the reward calculation mechanism due to integer division. These issues vary in severity and could lead to loss of funds, resource exhaustion, and inaccurate financial computations if not addressed.

Fix Review:

All identified issues have been fixed.

Source URLs: https://github.com/eonian-core/farm/tree/0c6318404552e9438654c54bf8fc9818d9b36d6e

Coverage

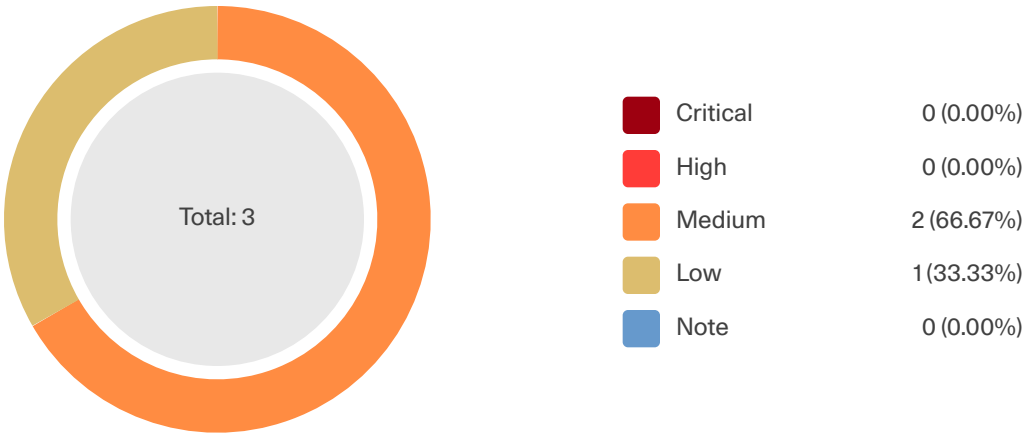
In this audit, we checked a total of 72 types of vulnerability patterns.

Found	3 (4.17%)	
Not Found	69 (95.83%)	

Title	Found
Front Running	-
Back Running	-
Sandwiching	-
Transaction order dependency	-
Fake tokens	-
Fake contracts	-
On-chain oracle manipulation	-
Governance attack	-
Token standard incompatibility	-
Flash liquidity borrow, purchase, mint, or deposit	-
Unsafe call to phantom function	-
One DeFi protocol dependency	-
Unfair slippage protection	-
Unfair liquidity providing	-
Unsafe or infinite token approval	-
Delegatecall injection	-
Unhandled or mishandled exception	-
Locked or frozen tokens	-
Absence of code logic or sanity check	ISSUE-3
Casting	ISSUE-2
Unbounded operation, including gas limit and call-stack depth	-
Arithmetic mistakes	-
Inconsistent access control	-
Visibility errors, including unrestricted action	-
Direct call to untrusted contract	-
Function Default Visibility	-
Integer Overflow and Underflow	-
Outdated Compiler Version	-
Floating Pragma	-

Unchecked Call Return Value	-
Unprotected Ether Withdrawal	-
Wrong Comparison Operator	-
UINT256 could be more gas efficient than smaller types	-
Magic numbers should be replaced with constants	-
Unprotected SELFDESTRUCT Instruction	-
Reentrancy	-
State Variable Default Visibility	-
Uninitialized Storage Pointer	-
Assert Violation	-
Use of Deprecated Solidity Functions	-
Delegatecall to Untrusted Callee	-
DoS with Failed Call	-
Transaction Order Dependence	-
Authorization through tx.origin	-
Block values as a proxy for time	-
Signature Malleability	-
Incorrect Constructor Name	-
Shadowing State Variables	-
Weak Sources of Randomness from Chain Attributes	-
Missing Protection against Signature Replay Attacks	-
Lack of Proper Signature Verification	-
Requirement Violation	-
Write to Arbitrary Storage Location	-
Incorrect Inheritance Order	-
Insufficient Gas Griefing	-
Arbitrary Jump with Function Type Variable	-
DoS With Block Gas Limit	ISSUE-1
Typographical Error	-
Right-To-Left-Override control character (U+202E)	-
Presence of unused variables	-
Unexpected Ether balance	-
Hash Collisions With Multiple Variable Length Arguments	-
Message call with hardcoded gas amount	-
Code With No Effects	-
Unencrypted Private Data On-Chain	-
Constant variables should be marked as private	-
Reading array length in for-loops	-
Checked arithmetic for for-loops	-
++i costs less gas than i++	-
IERC20.transfer does not support all ERC20 tokens	-
Unrestricted minting	-
Trust Issue Of Admin Roles	-

Findings



ID	Title	Severity	Status
ISSUE-1	DoS Vulnerability Due to Unbounded Loop in beforeWithdraw Function	Medium	Fixed
ISSUE-2	Precision Loss in Reward Calculation	Medium	Fixed
ISSUE-3	Missing Receiver Address Validation in Asset Transfer and Minting	Low	Fixed

Details

ISSUE-1: DoS Vulnerability Due to Unbounded Loop in beforeWithdraw Function

Severity

Medium

Location

contracts/ERC4626Lifecycle.sol:ERC4626Lifecycle:beforeWithdraw

Description

The function 'beforeWithdraw' contains an unbounded loop iterating over an array of hooks, which can lead to excessive gas consumption if the array grows large, potentially causing a DoS by hitting the block gas limit.

How to fix

Limit the number of hooks processed per transaction or implement pagination to handle large arrays without hitting the gas limit.

Code

```
function beforeWithdraw(uint256 assets, uint256 shares) internal virtual
override(ERC4626Upgradeable) {
        // if there are no hooks, then return to save gas
        if (withdrawHooks.length == 0) {
            return;
        }
        ERC4626HookPayload memory request = ERC4626HookPayload({
            assets: assets,
            shares: shares,
            requestSender: msg.sender,
            senderMaxWithdraw: maxWithdraw(msg.sender)
        });
        // iterate over hooks and call it
        for (uint256 i = 0; i < withdrawHooks.length; i++) {
            IVaultHook hook = withdrawHooks[i];
            hook.beforeWithdrawTrigger(request);
        }
    }
```

Code Suggestion

```
function beforeWithdraw(uint256 assets, uint256 shares) internal virtual
override(ERC4626Upgradeable) {
        // if there are no hooks, then return to save gas
        if (withdrawHooks.length == 0) {
            return;
        }
        uint256 maxHooks = 10; // Limit the number of hooks processed in one
transaction
        ERC4626HookPayload memory request = ERC4626HookPayload({
            assets: assets,
            shares: shares,
            requestSender: msg.sender,
            senderMaxWithdraw: maxWithdraw(msg.sender)
        });
        for (uint256 i = 0; i < withdrawHooks.length && i < maxHooks; i++) {
            IVaultHook hook = withdrawHooks[i];
            hook.beforeWithdrawTrigger(request);
        }
    }
```

Status

Fixed

Comment

Issue has been fixed.

ISSUE-2: Precision Loss in Reward Calculation

Severity

Medium

Location

contracts/RewardHolder.sol:RewardHolder:calcReward

Description

The function 'calcReward' in the RewardHolder contract suffers from precision loss due to integer division when calculating rewards. This issue arises because Solidity handles division of integers by truncating the result, which can lead to incorrect reward calculations, particularly critical in financial contexts.

How to fix

To fix this issue, use a fixed-point arithmetic library or scale the operands before division to maintain precision.

Code

```
function calcReward() public view returns (uint256) {
        if (numberCoins == 0 || rewardOwnerIndex[msg.sender] == 0) {
            return 0;
        }
        uint256 deltaIndex = rewardIndex - rewardOwnerIndex[msg.sender];
        return deltaIndex / numberCoins;
    }
```

Code Suggestion

```
function calcReward() public view returns (uint256) {
        if (numberCoins == 0 || rewardOwnerIndex[msg.sender] == 0) {
            return 0;
        }
        uint256 deltaIndex = rewardIndex - rewardOwnerIndex[msg.sender];
        return (deltaIndex * 1e18) / numberCoins; // Use a scaling factor to
maintain precision
    }
```

Status

Fixed

Comment

Issue has been fixed.

ISSUE-3: Missing Receiver Address Validation in Asset Transfer and Minting

Severity

Low

Location

contracts/ERC4626Upgradeable.sol:ERC4626Upgradeable:_receiveAndDeposit

Description

The function '_receiveAndDeposit' does not include a check to verify if the 'receiver' is a zero address before proceeding with asset transfer and minting operations, potentially leading to permanent loss of tokens if the 'receiver' is mistakenly set to a zero address.

How to fix

Add a check at the beginning of the function to ensure that the 'receiver' is not a zero address before proceeding with any state-changing operations.

Code

```
function _receiveAndDeposit(uint256 assets, uint256 shares, address receiver)
internal {
        // cases when msg.sender != receiver are error prone
        // but they are allowed by the standard... we need take care of it ourselves
        // Need to transfer before minting or ERC777s could reenter.
        asset.safeTransferFrom(msg.sender, address(this), assets);
        _mint(receiver, shares, "", "");
        emit Deposit(msg.sender, receiver, assets, shares);
        afterDeposit(assets, shares);
    }
```

Code Suggestion

```
function _receiveAndDeposit(uint256 assets, uint256 shares, address receiver)
internal {
        require(receiver != address(0), "Receiver cannot be the zero address");
        // cases when msg.sender != receiver are error prone
        // but they are allowed by the standard... we need take care of it ourselves
        // Need to transfer before minting or ERC777s could reenter.
        asset.safeTransferFrom(msg.sender, address(this), assets);
        _mint(receiver, shares, "", "");
        emit Deposit(msg.sender, receiver, assets, shares);
        afterDeposit(assets, shares);
    }
```

Status

Fixed

Comment

Issue has been fixed.

Disclaimer

This audit report is based on the analysis conducted at the time of the report, employing current industry standards for code quality, software processes, and cybersecurity practices. The report aims to identify security vulnerabilities and issues within the code base (including Source Code compilation, deployment, and functionality) as provided for audit, without making any representations or warranties regarding the exhaustive identification of all vulnerabilities or the absolute security of the code. The findings pertain exclusively to the code (branch/tag/commit hash) reviewed and may not apply to other branches or versions of the project.

It is important to understand that this report does not constitute a final or comprehensive assessment of the utility, safety, or bug-free status of the code, nor should it be interpreted as such. We have endeavored to conduct thorough analysis and provide accurate reporting; however, reliance on this single report for making critical decisions is not advised. We strongly recommend conducting additional independent audits, engaging in public bug bounty programs, and implementing continuous integration and delivery (CI/CD) processes to ensure ongoing security and quality of the code.

The security analysis is confined to the submitted smart contracts and does not extend to applications, operations, product code, the compiler layer, or any underlying hardware and software platforms which may have their own vulnerabilities. The scope of this report is limited to the materials and documentation provided and understood within the constraints of the review period. Consequently, the results may not encapsulate all possible vulnerabilities.

This report and its contents are provided "as is" and on a non-reliance basis, without any representations, warranties, or conditions of any kind, except as specifically stated herein. By accessing or using this report, you acknowledge and agree to the terms stated within this disclaimer, including the understanding that this report does not offer investment, legal, or financial advice. All liability for any kind of loss or damage arising from or in connection with this report, its use, or reliance is hereby disclaimed to the fullest extent permitted by law. No endorsement of any project, product, or service is implied by this report.

Use of this report, including any associated services, products, protocols, platforms, content, and materials, is at your sole risk. Blockchain technology is continually evolving and may carry unknown risks. We expressly disclaim all warranties, whether express or implied, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, and non- infringement.

This disclaimer serves to protect against any claims based on the report, its findings, or the absence thereof, and emphasizes the importance of conducting your own due diligence and obtaining independent advice as necessary.

Bunzz pte ltd