METATRUST

Security Assessment for

# Uniwhale

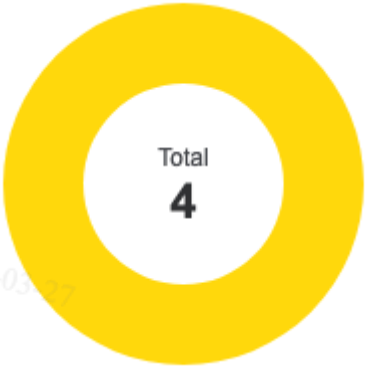
March 27, 2023

## Executive Summary

| Overview | |
|---|---|
| Project Name | Uniwhale |
| Codebase Path | git://github.com/uniwhale-io/uniwhale-v1 |
| Scan Engine | Security Analyzer |
| Scan Time | 2023/03/27 17:22:16 |
| Source Code | uniwhale-io/uniwhale-v1 commit:508f2a11 |

| Total | |
|---|---|
| Critical Issues | 0 |
| High risk Issues | 0 |
| Medium risk Issues | 4 |
| Low risk Issues | 0 |
| Informational Issues | 0 |

| | |
|---|---|
| Critical Issues | The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it. |
| High Risk Issues | The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users. |
| Medium Risk Issues | The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact. |
| Low Risk Issues | The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances. |
| Informational Issue | The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth. |




| | | |
|---|---|---|
| Critical Issues | 0% | 0 |
| High risk Issues | 0% | 0 |
| Medium risk Issues | 100% | 4 |
| Low risk Issues | 0% | 0 |
| Informational Issues | 0% | 0 |

## Summary of Findings

MetaScan security assessment was performed on **March 27, 2023 17:22:16** on project **Uniwhale** with the repository **uniwhale-io/uniwhale-v1** on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **4** vulnerabilities / security risks discovered during the scanning session, among which **0** critical vulnerabilities, **0** high risk vulnerabilities, **4** medium risk vulnerabilities, **0** low risk vulnerabilities, **0** informational issues.

| ID | Description | Severity | Alleviation |
|---|---|---|---|
| MSA-001 | Centralized Roles | Medium risk | Acknowledged |
| MSA-002 | Set Share on an Existing Claimer | Medium risk | Fixed |
| MSA-003 | Functions That are Necessary but not Called | Medium risk | Fixed |
| MSA-004 | The Rule of Update Reward in the `_update` Function | Medium risk | Acknowledged |

# Findings

## Critical (0)

No Critical vulnerabilities found here

## High risk (0)

No High risk vulnerabilities found here

## Medium risk (4)

### 1. Centralized Roles

Medium risk | Security Analyzer

In the contract UniwhaleToken, the role owner has the privileges of the below functions:

- snapshot();
- pause(): pause the contract;
- unpause(): unpause the contract;
- safeTransferrable(): set the transferrable;
- setEmission(): set the _balanceLastUpdate;
- pauseStaking(): set the _stakingPaused. In the contract UniwhaleToken, the role MINTER_ROLE has the privileges of the below functions:
- mint(): mint tokens to the to address;
- addBalance: add the balance;
- removeBalance: decrease the balance. In the contract RevenuePool, the role owner has the privileges of the below functions:
- transferBase(): transfer baseToken from the contract to any account.
- transferFromPool(): transfer baseToken from the contract to any account;
- setShare: set _shares; In the contract RevenuePool, the role approvedClaimer has the privileges of the below functions:
- addBalance: add balance to the ApprovedClaimer role;
- removeBalance: remove balance to the ApprovedClaimer role.
- mint(): mint tokens to any account.

**File(s) Affected**

contracts/RevenuePool.sol #1-12
contracts/tokens/UniwhaleToken.sol #1-21

**Examples**

```
1  // SPDX-License-Identifier: BUSL-1.1
2
3  import "./interfaces/AbstractPool.sol";
   ...
10 pragma solidity ^0.8.17;
11
12 contract RevenuePool is AbstractPool, IDistributable {
```

**Recommendation**

Applying the multi-signature wallet and the timelock to mitigate the risk of centralized roles.

**Alleviation** Acknowledged

The owner would be a multi-sig contract, partially mitigating the centralization risk.

## 2. Set Share on an Existing Claimer

Medium risk Security Analyzer

The function `setShare` requires the total shares of all the claimers to be less than or equal to `1e18`.However. it lacks a check on whether the claimer existed in the set `_shares` or not. If the parameter `claimer` existed in the array `_shares`, the `_sum` should deduct the old share of the existing `claimer` before the `_require` check.

**File(s) Affected**

contracts/RevenuePool.sol #63-74

**Examples**

```
63   function setShare(address claimer, uint256 _share) external onlyOwner {
64     _shares.set(claimer, _share);
65     uint256 _length = _shares.length();
   ...
72
73     emit SetShareEvent(claimer, _share);
74   }
```

**Recommendation**

Check if the `claimer` existed in the `_shares` set or not and deduct the old share of the existing `claimer`.

**Alleviation** Fixed

The client fixed this issue by updating the share for the user first and then checking the sum shares in the PR: https://github.com/uniwhale-io/uniwhale-v1/pull/258/files

## 3. Functions That are Necessary but not Called

Medium risk Security Analyzer

In the parent contract `AbstractStakeable`, the internal function `_addRewardToken()` adds the rewardToken to the set `_rewardTokens`. However, the sub-contract `UniwhaleToken` does not call the `_addRewardToken()` function in the parent contract. As a result, the set `_rewardTokens` will always be empty and users will not get any reward for their stakings. Similarly to the `_removeRewardToken()` function of the parent contract `AbstractStakeable`.

**File(s) Affected**

contracts/interfaces/AbstractStakeable.sol #103-106

**Examples**

```
103   function _addRewardToken(IMintable rewardToken) internal virtual {
104     _rewardTokens.add(address(rewardToken));
105     emit AddRewardTokenEvent(address(rewardToken));
106   }
```

**Recommendation**

Add a function in the sub-contract `UniwhaleToken` to call its parent's internal functions `_addRewardToken()` and `_removeRewardToken()`

**Alleviation** Fixed

The client fixed this issue as the recommendation, in the PR: https://github.com/uniwhale-io/uniwhale-v1/pull/258/files

## 4. The Rule of Update Reward in the `_update` Function

Medium risk Security Analyzer

The _update() function in the AbstractERC20Stakeable contract tries to update the reward of each staker. However, the following steps will make Alice withdraw all the reward:

- Assume the owner gives some rewardToken first and the rewardToken.balance() is 100;
- Alice stakes 10 tokens;
- Bob stakes 20 tokens;
- Alice stakes 10 tokens and the transaction will revert because of an overflow.

- Alice can claim 100 reward tokens without waiting once the owner gives the rewardToken to the pool. Here are the calculation steps of the _update function caused by the above three calls.

```
//Alice stakes with an amount of 10 tokens;
//_update(Alice,10)
_update(Alice, 10)
oldStaked = 0
newStaked = 0 + 10 = 10
If _length == 1

accuredRewards = _getRewards(Alice, rewardToken) = 0

balanceOut = 0 + 0 = 0
newTotalStaked = 0 - 0 = 0

balanceIn = 100 - 0 = 100 since rewardToken.balance() == 100
newTotalStaked > 0 is false

rewardToken.removeBalance(0);
rewardToken.addBalance(100); results rewardToken.balance() == 200
_balanceBaseByStaker[Alice][rewardToken] = 100

_accuredRewardsByStaker[Alice][rewardToken] += 0 = 0

_stakedByStaker[Alice] = 10
totalStaked = 0 + 10 = 10

---
_getRewards()
totalStaked == 0
  Return 0;

//Bob stakes with an amount of 20 tokens;
//_update(Alice,10)->_update(Bob,20)
_update(Bob, 20)
oldStaked = 0
newStaked = 0 + 20 = 20
If _length = 1

accruedRewards = 0

balanceOut = 0 + 0 = 0
newTotalStaked = 10 - 0 = 10

balanceIn = 200 - 0 = 200
If (newTotalStaked > 0)
  balanceIn = 200 * 20 / 10 = 400

rewardToken.removeBalance(0)
rewardToken.addBalance(400); results in rewardToken.balance() == 600
_balanceBaseByStaker[Bob][rewardToken] = 400

_accruedRewardsByStaker[Bob][rewardToken] += 0 = 0

_stakedByStaker[Bob] = 20
totalStaked = 10 + 20 = 30

----
_getRewards()
  200
  * 0
  / 10
  - 0
  = 0
```

```
//Alice stakes with an amount of 10 tokens
//_update(Alice,10)->_update(Bob,20)
//->_update(Alice,10)
oldStaked = 10
newStaked = 10 + 10 = 20
If _length == 1

accruedRewards = 100;

balanceOut = 100 + 100 = 200
newTotalStaked = 30 - 10 = 20

balanceIn = 600 - 200 = 400
newTotalStaked > 0
  balanceIn = 400 * 20 / 20 = 400

rewardToken.removeBalance(200);
rewardToken.addBalance(400); results in the rewardToken.balance() == 800;
_balanceBaseByStaker[Alice][rewardToken] = 400;

_accruedRewardsByStaker[Alice][rewardToken] += 100 = 100

_stakedByStaker[Alice] = 20
totalStaked = 30 + 10 = 40
----
_getRewards()
  600
  * 10
  / 30
  - 100
  = 100

//Alice claims tokens
//_update(Alice, 0)
oldStaked = 20;
newStaked = 20 + 0 = 20;
assumes that _length == 1

accruedRewards = 0
...
_accruedRewardByStaker[Alice][rewardToken] += 0 = 100

---
_getRewards()
  800
  * 20
  / 40
  - 400
```

Here is another case where Bob can withdraw most of the reward by using the sandwich attack with the following steps:

- Alice stakes 10 tokens for 1 month;
- A month later.
- Bob stakes 1000 tokens (before the owner distributes the reward token);
- Assume the owner gives some rewardToken first and the rewardToken.balance() is 100;
- Bob claims the 99 reward token, even though Alice has been staking longer; Here are the calculation steps of the _update function caused by the above three calls.

```
//Alice stakes with an amount of 10 tokens;
//_update(Alice,10)
_update(Alice, 10)
oldStaked = 0
newStaked = 0 + 10 = 10
If _length == 1
```

```
accuredRewards = _getRewards(Alice, rewardToken) = 0

balanceOut = 0 + 0 = 0
newTotalStaked = 0 - 0 = 0

balanceIn = 0 - 0 = 0
newTotalStaked > 0 is false

rewardToken.removeBalance(0);
rewardToken.addBalance(0); results rewardToken.balance() == 0
_balanceBaseByStaker[Alice][rewardToken] = 0

_accuredRewardsByStaker[Alice][rewardToken] += 0 = 0

_stakedByStaker[Alice] = 10
totalStaked = 0 + 10 = 10

---
_getRewards()
totalStaked == 0
  Return 0;

//Bob stakes with an amount of 1000 tokens;
//_update(Alice,10)->_update(Bob,1000)
_update(Bob, 100)
oldStaked = 0
newStaked = 0 + 1000 = 1000
If _length = 1

accruedRewards = 0

balanceOut = 0 + 0 = 0
newTotalStaked = 10 - 0 = 10

balanceIn = 0 - 0 = 0
If (newTotalStaked > 0)
  balanceIn = 0 * 1000 / 10 = 0

rewardToken.removeBalance(0)
rewardToken.addBalance(0);
_balanceBaseByStaker[Bob][rewardToken] = 0

_accruedRewardsByStaker[Bob][rewardToken] += 0 = 0

_stakedByStaker[Bob] = 1000
totalStaked = 10 + 1000 = 1010

----
_getRewards()
  0
  - 0
  = 0

//Alice stakes with an amount of 10 tokens
//_update(Alice,10)->_update(Bob,1000)-> minter add 100 reward token

rewardToken.balance() == 100

//Bob claims tokens
//_update(Bob, 0)
oldStaked = 1000;
newStaked = 1000 + 0 = 1000;
assumes that _length == 1

accruedRewards = 0
...
```

```
_accruedRewardByStaker[Alice][rewardToken] += 0 = 99

---
_getRewards()
  100
  * 1000
  / 1010
  - 0
```

**File(s) Affected**

contracts/interfaces/AbstractStakeable.sol #113-158

**Examples**

```
113    function _update(address staker, int256 stakedDelta) internal virtual {
114      uint256 oldStaked = _stakedByStaker[staker];
115      uint256 newStaked = oldStaked.add(stakedDelta).toUint256();
  •••
156          .divDown(_totalStaked)
157          .sub(_balanceBaseByStaker[staker][_rewardToken]);
158    }
```

**Alleviation** Acknowledged

The Uniwhale team responded that the team mitigates the issue by distributing the rewards in a linear way, and the team is looking to improve on the economic design/logic.

## Low risk (0)

No Low risk vulnerabilities found here

## Informational (0)

No Informational vulnerabilities found here

## Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without MetaTrust's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts MetaTrust to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. MetaTrust's position is that each company and individual are responsible for their own due diligence and continuous security. MetaTrust's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by MetaTrust is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS Security Assessment AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, MetaTrust HEREBY DISCLAIMS ALL

WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, MetaTrust SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, MetaTrust MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, MetaTrust PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER MetaTrust NOR ANY OF MetaTrust'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. MetaTrust WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT MetaTrust'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING Security Assessment MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST MetaTrust WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF MetaTrust CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST MetaTrust WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.