# Competitive Security Assessment

uniwhale.co P2

Mar 20th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	uniwhale.co P2
Platform & Language	Solidity
Codebase	• https://github.com/uniwhale-io/uniwhale-v1 • audit commit - a83bb294d52b764483e6b02d537427b45b8c800b • final commit - 4eac1a2de89b1b6149e58d79f8d07da296ed59f9
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

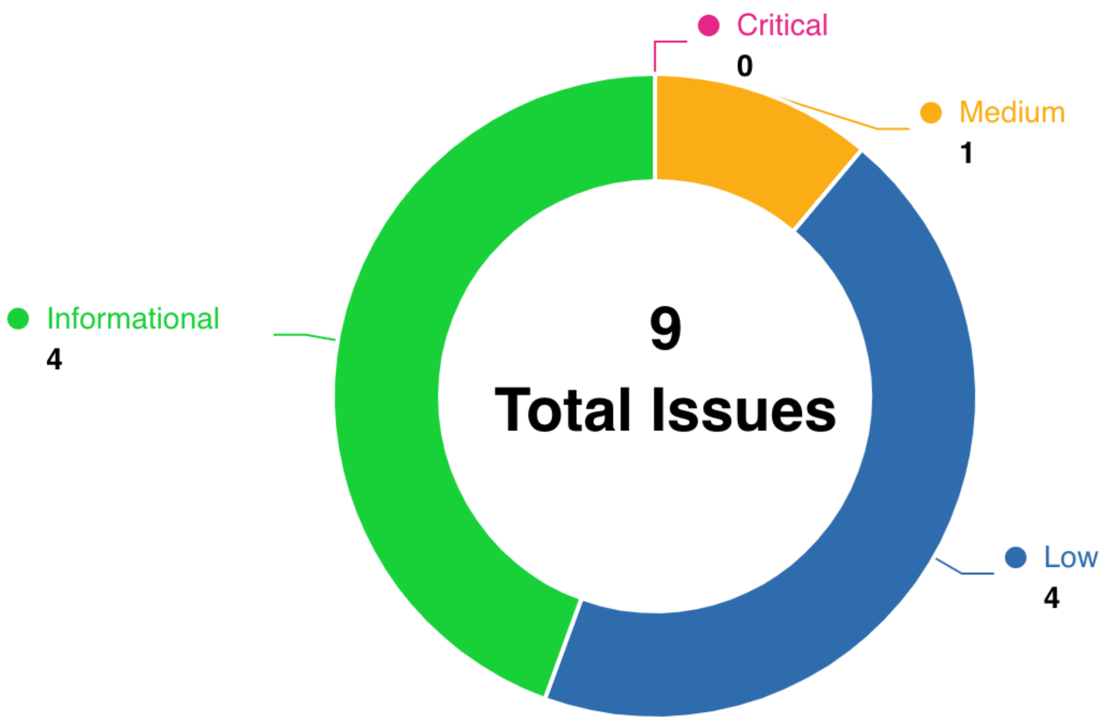
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	**0**	**0**	**0**	**0**	**0**	**0**
Medium	**1**	**0**	**0**	**1**	**0**	**0**
Low	**4**	**0**	**1**	**1**	**0**	**2**
Informational	**4**	**0**	**0**	**2**	**1**	**1**

Audit Scope

File	Commit Hash
./tokens/UniwhaleToken.sol	a83bb294d52b764483e6b02d537427b45b8c800b
./interfaces/AbstractStakeable.sol	a83bb294d52b764483e6b02d537427b45b8c800b
./RevenuePool.sol	a83bb294d52b764483e6b02d537427b45b8c800b
./interfaces/AbstractERC20Stakeable.sol	a83bb294d52b764483e6b02d537427b45b8c800b

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
UNW-1	Centralization risk in `RevenuePool::mint`	Logical	Medium	Fixed	0xxm, alansh
UNW-2	Centralization risk in `RevenuePool::transferBase` and `RevenuePool::transferFromPool`	Logical	Low	Declined	0xxm
UNW-3	Centralized risk in `UniwhaleToken::mint`	Centralized risk	Informational	Mitigated	Xi_Zi
UNW-4	Code Style in `UniwhaleToken` contract	Code Style	Informational	Fixed	Xi_Zi

UNW-5	Duplicate functionality in `RevenuePool::transferFromPool` function	Logical	Informational	Declined	Xi_Zi
UNW-6	Tokens transferred into `RevenuePool` are improperly distributed among claimers	Logical	Low	Declined	0xxm
UNW-7	Unnecessary getter for public variables in contract `AbstractStakeable` and `RevenuePool`	Gas Optimization	Informational	Fixed	0xxm
UNW-8	`setShare` might fail unexpectedly when change `_share` of existing `claimer`	Logical	Low	Fixed	0xxm, alansh
UNW-9	logic issue in `AbstractERC20Stakeable::_getRewards` function	Logical	Low	Acknowledged	alansh

UNW-1:Centralization risk in `RevenuePool::mint`

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	• **code/packages/contracts/core-v1/contracts/RevenuePool.sol#L93-L100** • **code/packages/contracts/core-v1/contracts/RevenuePool.sol#L93-L98**	**Fixed**	**0xxm, alansh**

Code

```
93:  function mint(
94:    address to,
95:    uint256 amount
96:  ) external override onlyApprovedClaimer(msg.sender) {
97:    uint256 _amount = amount.min(baseToken.balanceOfFixed(address(this)));
98:    baseToken.transferFixed(to, _amount);
99:  }
100:}

93:  function mint(
94:    address to,
95:    uint256 amount
96:  ) external override onlyApprovedClaimer(msg.sender) {
97:    uint256 _amount = amount.min(baseToken.balanceOfFixed(address(this)));
98:    baseToken.transferFixed(to, _amount);
```

Description

0xxm : The `mint` function allows any approved claimer to drain all baseToken in RevenuePool.
alansh : With the current implementation, anyone that has some share in the pool can transfer all baseToken. This doesn't make sense.

Recommendation

0xxm : According to the logic of RevenuePool, the `mint` function should only allow cliamer to transfer baseToken up to its balance.

Meanwhile, it is suggested to change the name of `mint` to a more appropriate name (maybe `claim`?)

```
function mint(
  address to,
  uint256 amount
) external override onlyApprovedClaimer(msg.sender) {
  uint256 _amount = amount.min(_balances[msg.sender]);
  _balances[msg.sender] -= _amount;
  baseToken.transferFixed(to, _amount);
}
```

alansh : `_amount` should capped with the balance of `msg.sender`, and decrease after `transferFixed`.

Client Response

Mint is limited up to the balance held by minter.

UNW-2:Centralization risk in `RevenuePool::transferBase` and `RevenuePool::transferFromPool`

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/RevenuePool.sol#L41 • code/packages/contracts/core-v1/contracts/RevenuePool.sol#L48	Declined	0xxm

Code

```
41:   function transferBase(

48:   function transferFromPool(
```

Description

0xxm : Function `transferBase` and `transferFromPool` allow owner to arbitrarily remove token from RevenuePool, which can break the functionality of claimer balances. It is very likely that claimer might fail to claim tokens as stored in `_balances`.

```
function transferBase(
  address _to,
  uint256 _amount
) external override onlyOwner {
  baseToken.transferFixed(_to, _amount);
}

function transferFromPool(
  address _token,
  address _to,
  uint256 _amount
) external override onlyOwner {
  _require(_token == address(baseToken), Errors.TOKEN_MISMATCH);
  baseToken.transferFixed(_to, _amount);
}
```

Impact: The severity is set to Informational, as it is unclear about project team's intention on function `transferBase` and `transferFromPool`:

- if they are designed to rescue tokens or as a completely centralized way to tranfer tokens to claimers, it should be fine.
- if they are designed to claim tokens on behalf of claimer, it should be fixed as recommended.

Recommendation

0xxm : - Update `_balances` of `_to` at the end of above functions:

```
...
baseToken.transferFixed(_to, _amount);
_balances[_to] -= _balances[_to].min(_amount);
```

- emit events for centralized operations

Client Response

Declined.They are designed to rescue tokens in an emergency.

UNW-3:Centralized risk in `UniwhaleToken::mint`

Category	Severity	Code Reference	Status	Contributor
Centralized risk	**Informational**	• **code/packages/contracts/core-v1/contracts/tokens/UniwhaleToken.sol#L52-L58** • **code/packages/contracts/core-v1/contracts/tokens/UniwhaleToken.sol#L115**	**Mitigated**	**Xi_Zi**

Code

```
52:  function initialize(
53:    address owner,
54:    string memory name,
55:    string memory symbol,
56:    bool _transferrable,
57:    uint256 _cap
58:  ) public initializer {

115:    _mint(to, amount);
```

Description

Xi_Zi : Centralized risk, privileged accounts can be minted at will, and the risk is higher. Multiple privileged addresses in the contract, such as DEFAULT_ADMIN_ROLE, MINTER_ROLE, and OWNER, are the same account.As there are privileged accounts of various roles in the contract, which play a key role in the contract, it is necessary to implement multi-signature protection for the accounts of various roles in the contract.

Recommendation

Xi_Zi : Multi-sign protection is required for the accounts of various roles of the contract.And it is recommended to separate the permissions of different privileged accounts.

Client Response

Mitigated.The contractor owner is a multi-sig contract.

UNW-4:Code Style in `UniwhaleToken` contract

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	• **code/packages/contracts/core-v1/contracts/interfaces/AbstractERC20Stakeable.sol#L47** • **code/packages/contracts/core-v1/contracts/tokens/UniwhaleToken.sol#L102**	**Fixed**	**Xi_Zi**

Code

```
47:    if (sender != address(this))

102:      _balance += emission * (block.number.sub(_balanceLastUpdate));
```

Description

Xi_Zi : It is recommended to use curly braces in the if statement as much as possible. This can avoid possible ambiguities and errors, especially when the code needs to be modified. Using curly braces can make the code easier to understand and maintain, and it can also make the code more consistent.

Recommendation

Xi_Zi : It is recommended to use curly braces in the if statement as much as possible. This can avoid possible ambiguities and errors, especially when the code needs to be modified. Using curly braces can make the code easier to understand and maintain, and it can also make the code more consistent.

```
function setEmission(uint256 _emission) external onlyOwner {
    if (_balanceLastUpdate > 0) {
      _balance += emission * (block.number.sub(_balanceLastUpdate));//@audit
    }
    _balanceLastUpdate = block.number;
    emission = _emission;
    emit SetEmissionEvent(emission);
  }
```

Client Response

Fixed

UNW-5:Duplicate functionality in `RevenuePool::transferFromPool` function

Category	Severity	Code Reference	Status	Contributor
Logical	**Informational**	• **code/packages/contracts/core-v1/contracts/RevenuePool.sol#L41-L55**	**Declined**	**Xi_Zi**

Code

```
41:  function transferBase(
42:    address _to,
43:    uint256 _amount
44:  ) external override onlyOwner {
45:    baseToken.transferFixed(_to, _amount);
46:  }
47:
48:  function transferFromPool(
49:    address _token,
50:    address _to,
51:    uint256 _amount
52:  ) external override onlyOwner {
53:    _require(_token == address(baseToken), Errors.TOKEN_MISMATCH);
54:    baseToken.transferFixed(_to, _amount);
55:  }
```

Description

Xi_Zi : The transferFromPool function and the transferBase function have the same function. There is only one additional _token parameter, and the _token parameter can only be equal to baseToken and there is no setter function to update `baseToken`. this means `transferFromPool` has the same effect as the `transferBase` and they are duplicate functions. If onlyOwner can only transfer baseToken, it is recommended to remove the _token parameter without require judgment.

Recommendation

Xi_Zi : It is recommended to modify it according to the actual situation to reduce duplicate functions.

Client Response

Declined.While they are duplicates, RevenuePool inherits from IPool and requires to implement these functions, which are designed to rescue tokens in an emergency.

UNW-6:Tokens transferred into `RevenuePool` are improperly distributed among claimers

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/packages/contracts/core-v1/contracts/RevenuePool.sol#L76**	**Declined**	**0xxm**

Code

```
76:      _balances[address(claimer)] += amount.mulDown(share);
```

Description

0xxm : Contract `RevenuePool` uses `_shares` to store approvedClaimers and their shares, and the sum of all shares is limited to 1e18. Whereas, the distributed token balance is calculated as `amount.mulDown(share)` when amount of token is transferred in.

When the sum of all shares are less than 1e18, not all tokens transferred into RevenuePool are distributed to claimers. In other word, the sum of claimer's balance is less than the total balance of `RevenuePool`.

Impact: Considering the `RevenuePool` is highly centralized, the undistributed token can still be correctly transferred to claimers using off-chain records, the severity is *Low*

Recommendation

0xxm : Store sum of shares in a storage variable and distribute tokens using "claimer's share / sumOfShare"

```
uint256 totalShares;
function setShare(address claimer, uint256 _share) external onlyOwner {
    _shares.set(claimer, _share);
    uint256 _length = _shares.length();
    uint256 _sum = 0;
    for (uint256 i = 0; i < _length; i++) {
      (, uint256 __share) = _shares.at(i);
      _sum += __share;
    }
    _require(_sum  <= 1e18, Errors.INVALID_SHARE);
    totalShares = _sum;
    emit SetShareEvent(claimer, _share);
}

function transferIn(uint256 amount) external {
  baseToken.transferFromFixed(msg.sender, address(this), amount);
  uint256 _length = _shares.length();
  for (uint256 i = 0; i < _length; i++) {
    (address claimer, uint256 share) = _shares.at(i);
    _balances[address(claimer)] += amount.mulDown(share).divDown(totalShares);
  }
}
```

Client Response

Declined.This is an intended behaviour. While we do not want to see the sum of shares exceeding 100% (hence the check against the sum), we also do not want to see the share being re-based by the sum.

UNW-7:Unnecessary getter for public variables in contract `AbstractStakeable` and `RevenuePool`

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/packages/contracts/core-v1/contracts/interfaces/AbstractStakeable.sol#L16 • code/packages/contracts/core-v1/contracts/RevenuePool.sol#L19	Fixed	0xxm

Code

```
16:  uint256 public totalStaked;

19:  mapping(address => uint256) public _balances;
```

Description

0xxm : Compiler will generate a default getter function for public variable. The `getTotalStaked()` function is unnecessary for public variable `totalStaked`.

```
uint256 public totalStaked;
...
function getTotalStaked() external view virtual override returns (uint256) {
  return totalStaked;
}
```

The same case applies to `balance(address claimer)` function for public variable `_balances` in contract `RevenuePool`.

```
mapping(address => uint256) public _balances;
...
function balance() external view override returns (uint256) {
  return _balance(msg.sender);
}

//@dev may be removed before deployment
function balance(address claimer) external view returns (uint256) {
  return _balance(claimer);
}

function _balance(address claimer) internal view returns (uint256) {
  return _balances[claimer];
}
```

Recommendation

0xxm : - declare `totalStaked` and `_balances` as internal variables

Client Response

Fixed

UNW-8: `setShare` might fail unexpectedly when change `_share` of existing `claimer`

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/RevenuePool.sol#L59-L66 • code/packages/contracts/core-v1/contracts/RevenuePool.sol#L65	Fixed	0xxm, alansh

Code

```
59:    uint256 _length = _shares.length();
60:    uint256 _sum = 0;
61:    for (uint256 i = 0; i < _length; i++) {
62:      (, uint256 __share) = _shares.at(i);
63:      _sum += __share;
64:    }
65:    _require(_sum + _share <= 1e18, Errors.INVALID_SHARE);
66:    _shares.set(claimer, _share);

65:    _require(_sum + _share <= 1e18, Errors.INVALID_SHARE);
```

Description

0xxm : Contract `RevenuePool` uses `_shares` to store approvedClaimers and their shares, and the sum of all shares is limited to 1e18. However the total share should not be calculated as `_sum + _share` when modifying share of existing claimer instead of set share for new claimer.

Consider the following case:

- the total shares in `_shares` is 9e17, and Alice is one of claimers with share equals to 1e17.
- the owner want to change Alice's share from 1e17 to 2e17, which increases the share by 1e17 and total shares should be exactly 1e18
- however, the `setShare` function will always fail, as `_sum` is 9e17 and `_share` is 2e17 (_sum + _share > 1e18)

Impact: In such a case, the owner can still use a two-step workaround that first set share to zero and then to expect value, without modifying the code. The severity is defined as *LOW*
alansh : The current implementation doesn't consider that `claimer` may already be in the `_shares` map.

Recommendation

0xxm : Move the check after the shares.set operation:

```
function setShare(address claimer, uint256 _share) external onlyOwner {
    _shares.set(claimer, _share);
    uint256 _length = _shares.length();
    uint256 _sum = 0;
    for (uint256 i = 0; i < _length; i++) {
      (, uint256 __share) = _shares.at(i);
      _sum += __share;
    }
    _require(_sum  <= 1e18, Errors.INVALID_SHARE);
    emit SetShareEvent(claimer, _share);
}
```

alansh :

```
_require(_sum + _share <= 1e18, Errors.INVALID_SHARE);
```

should be changed to:

```
_require(_sum - _shares[claimer] + _share <= 1e18, Errors.INVALID_SHARE);
```

Client Response

Fixed.We accept 0xxm’s suggestion

UNW-9:logic issue in `AbstractERC20Stakeable::_getRewards` function

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/packages/contracts/core-v1/contracts/interfaces/AbstractERC20Stakeable.sol#L66-L71	Acknowledged	alansh

Code

```
66:    return
67:      _rewardToken
68:        .balance()
69:        .sub(_balanceBaseByStaker[user][_rewardToken])
70:        .mulDown(_stakedByStaker[user])
71:        .divUp(totalStaked);
```

Description

alansh : With the current implementation, users will experience sudden reward decrease.

Imagine that a whale suddenly stakes a large amount of token, then in the reward formula, only `totalStaked` changes significantly, and the reward will suddenly decrease a lot. This is not what users expect.

Recommendation

alansh : The correct logic is to globally maintain a `reward_per_staked_token`, and when a user stakes, save a snapshot of current `reward_per_staked_token`, then when calculating reward, simply `staked_amount * (current_reward_per_staked_token-snapshot_reward_per_staked_token)`. And the global `reward_per_staked_token` is calculated incrementally:

```
reward_per_staked_token += total_reward_delta/total_staked_amount;
```

Client Response

Acknowledged.The reward distribution is being re-worked on based on feedbacks and the recommended fix will be implemented as part of the upgrade

Disclaimer