# Smart Contract Security Audit Report

# Table Of Contents

# 1 Executive Summary

On 2023.12.13, the SlowMist security team received the Lumiterra Community team's security audit application for Lumiterra Community Contracts, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

| Test method | Description |
| --- | --- |
| Black box testing | Conduct security tests from an attacker's perspective externally. |
| Grey box testing | Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses. |
| White box testing | Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc. |

The vulnerability severity level information:

| Level | Description |
| --- | --- |
| Critical | Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities. |
| High | High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities. |
| Medium | Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities. |
| Low | Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed. |
| Weakness | There are safety risks theoretically, but it is extremely difficult to reproduce in engineering. |
| Suggestion | There are better practices for coding or architecture. |

# 2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

| Serial Number | Audit Class | Audit Subclass |
| --- | --- | --- |
| 1 | Overflow Audit | - |
| 2 | Reentrancy Attack Audit | - |
| 3 | Replay Attack Audit | - |
| 4 | Flashloan Attack Audit | - |
| 5 | Race Conditions Audit | Reordering Attack Audit |
| 6 | Permission Vulnerability Audit | Access Control Audit |
| | | Excessive Authority Audit |
| 7 | Security Design Audit | External Module Safe Use Audit |
| | | Compiler Version Security Audit |
| | | Hard-coded Address Security Audit |
| | | Fallback Function Safe Use Audit |
| | | Show Coding Security Audit |
| | | Function Return Value Security Audit |
| | | External Call Function Security Audit |

| Serial Number | Audit Class | Audit Subclass |
| --- | --- | --- |
| 7 | Security Design Audit | Block data Dependence Security Audit |
| | | tx.origin Authentication Security Audit |
| 8 | Denial of Service Audit | - |
| 9 | Gas Optimization Audit | - |
| 10 | Design Logic Audit | - |
| 11 | Variable Coverage Vulnerability Audit | - |
| 12 | "False Top-up" Vulnerability Audit | - |
| 13 | Scoping and Declarations Audit | - |
| 14 | Malicious Event Log Audit | - |
| 15 | Arithmetic Accuracy Deviation Audit | - |
| 16 | Uninitialized Storage Pointer Audit | - |

# 3 Project Overview

## 3.1 Project Introduction

This is the contract part of Lumiterra Community, including Liquidity Stake, Price Field, Utility Stake, Token and VAMM modules.

## 3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

| NO | Title | Category | Level | Status |
| --- | --- | --- | --- | --- |
| N1 | Analyzing inaccuracies in reward calculation | Others | Medium | Fixed |

| NO | Title | Category | Level | Status |
| --- | --- | --- | --- | --- |
| | due to time span misalignment | | | |
| N2 | Handling reward calculation issues and function restrictions in reward cycles | Others | High | Fixed |
| N3 | Addressing overpayment risk in reward distribution due to rate update delays | Others | Critical | Fixed |
| N4 | Inaccuracies in reward calculation due to misuse of total supply in liquidity pool | Others | Critical | Fixed |
| N5 | Redundant logic in _setFloorPrice function of smart contract | Others | Low | Acknowledged |
| N6 | Potential reentrancy risk in VAMM's _mintByPRToken function | Others | High | Fixed |
| N7 | Exploitation risk with arbitrary payToken in VAMM's _mintByPRToken function | Others | High | Fixed |
| N8 | Preemptive Initialization | Race Conditions Vulnerability | Suggestion | Acknowledged |
| N9 | Lacking event logging in critical contract functions alters state without transparency issue | Malicious Event Log Audit | Suggestion | Acknowledged |
| N10 | Redundant functions | Others | Information | Fixed |
| N11 | Missing check return value | Others | Low | Acknowledged |
| N12 | Risk of excessive authority | Authority Control Vulnerability Audit | Medium | Acknowledged |
| N13 | Recommendation to | Reentrancy | Suggestion | Acknowledged |

| NO | Title | Category | Level | Status |
| --- | --- | --- | --- | --- |
| | Implement reentrancy | Vulnerability | | |

# 4 Code Overview

## 4.1 Contracts Description

https://github.com/LumiterraCommunity/contracts

commit: d26e0b50467035a71f14d6d888a2f57af8766dcf

review commit:557ed44fa4f81ccfe7e8e493e66df839395d8aef

Audit scope:

- contracts/LiquidityStake.sol
- contracts/PriceField.sol
- contracts/UtilityStake.sol
- contracts/UtilityToken.sol
- contracts/VAMM.sol

The main network address of the contract is as follows:

**The code was not deployed to the mainnet.**

## 4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

| LiquidityStake | | | |
| --- | --- | --- | --- |
| Function Name | Visibility | Mutability | Modifiers |
| initialize | Public | Can Modify State | initializer |
| setHook | External | Can Modify State | onlyOwner |
| _sc2crvPool | Internal | - | - |

| LiquidityStake | | | |
|---|---|---|---|
| _totalSC2CRVLP | Internal | - | - |
| _metaGauge | Internal | - | - |
| calcUnstakeToUserAmount | External | - | - |
| _borrow2CRV | Internal | Can Modify State | - |
| _repay2CRV | Internal | Can Modify State | - |
| calcStakeLP | External | - | - |
| withdrawFees | External | Can Modify State | - |
| stake | External | Can Modify State | - |
| unstake | External | Can Modify State | - |
| unstake | External | Can Modify State | - |
| _unstake | Internal | Can Modify State | - |
| _checkpoint | Internal | Can Modify State | - |
| viewCanClaimAmount | External | Can Modify State | - |
| claim | External | Can Modify State | - |
| _authorizeUpgrade | Internal | Can Modify State | onlyOwner |

| PriceField | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| _setFloorPrice | Internal | Can Modify State | - |
| setFloorPrice | External | Can Modify State | onlyVamm |
| increaseSupplyWithNoPriceImpact | External | Can Modify State | onlyVamm |
| exerciseAmount | External | - | - |

| PriceField | | | |
|---|---|---|---|
| slope | External | - | - |
| slope0 | External | - | - |
| floorPrice | External | - | - |
| x1 | Public | - | - |
| x1 | Public | - | - |
| x2 | Public | - | - |
| c | Public | - | - |
| c1 | Public | - | - |
| b2 | Public | - | - |
| k | Public | - | - |
| finalPrice1 | External | - | - |
| finalPrice2 | External | - | - |
| _finalPrice1 | Internal | - | - |
| _finalPrice2 | Internal | - | - |
| getPrice1 | External | - | - |
| getPrice2 | External | - | - |
| _getPrice1 | Internal | - | - |
| _getPrice2 | Internal | - | - |
| _getPrice0 | Internal | - | - |
| getUseFPBuyPrice | Public | - | - |
| getBuyPrice | External | - | - |
| getSellPrice | External | - | - |

| PriceField | | | |
|---|---|---|---|
| getSellPrice | External | - | - |

| UtilityStake | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| initialize | Public | Can Modify State | initializer |
| setHook | External | Can Modify State | onlyOwner |
| calcLendAmount | External | - | - |
| lendPrice | External | - | - |
| _calcCanLendAmount | Private | - | - |
| _lend | Private | Can Modify State | - |
| userInfo | External | - | - |
| stake | External | Can Modify State | - |
| calcUnstakeRepayAmount | Public | - | - |
| _repay | Private | Can Modify State | - |
| unstake | External | Can Modify State | - |
| _calcClaimableAmount | Internal | - | - |
| _claim | Internal | Can Modify State | - |
| claim | External | Can Modify State | - |
| depositRewardToken | External | Can Modify State | - |
| withdrawFees | External | Can Modify State | - |
| _authorizeUpgrade | Internal | Can Modify State | onlyOwner |

| UtilityToken | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | ERC20 |
| mint | Public | Can Modify State | onlyOwner |
| burn | Public | Can Modify State | - |
| burnFrom | Public | Can Modify State | - |
| transferOwnership | Public | Can Modify State | onlyOwner |
| decimals | Public | - | - |

| VAMM | | | |
|---|---|---|---|
| Function Name | Visibility | Mutability | Modifiers |
| <Constructor> | Public | Can Modify State | - |
| initialize | Public | Can Modify State | initializer |
| getPriceField | External | - | - |
| setPriceField | External | Can Modify State | onlyOwner |
| getLiquidity | Public | - | - |
| _convertPrice | Private | - | - |
| _collectFees | Private | Can Modify State | - |
| mintByPRToken | External | Can Modify State | - |
| mintByPRToken | External | Can Modify State | - |
| _mintByPRToken | Internal | Can Modify State | - |
| _mint | Internal | Can Modify State | - |
| mint | External | Can Modify State | - |
| mint | External | Can Modify State | - |

| VAMM | | | |
|---|---|---|---|
| _burn | Internal | Can Modify State | - |
| liquidityTesting | External | Can Modify State | - |
| burn | External | Can Modify State | - |
| burn | External | Can Modify State | - |
| balanceOf | Public | - | - |
| _deposit | Internal | Can Modify State | - |
| _claimCurveRewards | Internal | Can Modify State | - |
| withdrawFees | External | Can Modify State | - |
| _transfer | Internal | Can Modify State | - |
| canBorrowAmount | External | - | - |
| borrow | External | Can Modify State | - |
| repay | External | Can Modify State | - |
| _reduceT | Internal | Can Modify State | - |
| _increaseT | Internal | Can Modify State | - |
| _autoUpFP | Internal | Can Modify State | - |
| updateMFR | External | Can Modify State | onlyOperator |
| _updateMFR | Internal | Can Modify State | - |
| _authorizeUpgrade | Internal | Can Modify State | onlyOwner |

## 4.3 Vulnerability Summary

**[N1] [Medium] Analyzing inaccuracies in reward calculation due to time span misalignment**

**Category: Others**

**Content**

- contracts/UtilityStake.sol

In the `_calcClaimableAmount` function, if diff (the time elapsed since the last reward claim) exceeds `timeoutClaimPeriod`, then the result of the modulo operation `diff % timeoutClaimPeriod`, which yields `claimD`, will be less than the actual time span that should be considered. This leads to an inaccurate calculation of the reward amount.

```
function _calcClaimableAmount(
    address staker
) internal view returns (uint256) {
    StakeInfo memory stakeInfo = stakeInfoByStaker[staker];
    if (stakeInfo.stakedAmount == 0) {
        return 0;
    }
    if (stakeInfo.latestClaimedAt == 0) {
        return 0;
    }
    if (stakeInfo.latestClaimedAt > block.timestamp) {
        return 0;
    }
    if (totalStakeAmount == 0) {
        return 0;
    }
    if (rate == 0) {
        return 0;
    }
    uint256 diff = block.timestamp - stakeInfo.latestClaimedAt;
    uint256 claimD = diff % timeoutClaimPeriod;

    // Avoid delayed on-chain submissions resulting in zero rewards 0
    if (diff == timeoutClaimPeriod) {
        claimD = timeoutClaimPeriod;
    }

    uint256 claimableAmount = Math.mulDiv(
        stakeInfo.stakedAmount * claimD,
        rate,
        totalStakeAmount,
        Math.Rounding.Zero
    );
    return claimableAmount;
}
```

**Solution**

Check that this reward model meets expectations.

**Status**

Fixed; Function Delete Deactivate.

fix commit:

557ed44fa4f81ccfe7e8e493e66df839395d8aef

### [N2] [High] Handling reward calculation issues and function restrictions in reward cycles

**Category: Others**

**Content**

- contracts/UtilityStake.sol

Users will still be able to use the old rate to calculate rewards and collect them when the next cycle has not yet been set up. This also causes a problem. When there are not enough reward tokens in the pool, users can't use the `unstake` and `stake` functions.

```
function _calcClaimableAmount(
    address staker
) internal view returns (uint256) {
    StakeInfo memory stakeInfo = stakeInfoByStaker[staker];
    if (stakeInfo.stakedAmount == 0) {
        return 0;
    }
    if (stakeInfo.latestClaimedAt == 0) {
        return 0;
    }
    if (stakeInfo.latestClaimedAt > block.timestamp) {
        return 0;
    }
    if (totalStakeAmount == 0) {
        return 0;
    }
    if (rate == 0) {
        return 0;
    }
    uint256 diff = block.timestamp - stakeInfo.latestClaimedAt;
    uint256 claimD = diff % timeoutClaimPeriod;

    // Avoid delayed on-chain submissions resulting in zero rewards 0
```

```
    if (diff == timeoutClaimPeriod) {
        claimD = timeoutClaimPeriod;
    }

    uint256 claimableAmount = Math.mulDiv(
        stakeInfo.stakedAmount * claimD,
        rate,
        totalStakeAmount,
        Math.Rounding.Zero
    );
    return claimableAmount;
}
```

**Solution**

There should be no rewards to collect after a period of time.

**Status**

Fixed; Function Delete Deactivate.

fix commit:

557ed44fa4f81ccfe7e8e493e66df839395d8aef

### [N3] [Critical] Addressing overpayment risk in reward distribution due to rate update delays

**Category: Others**

**Content**

- contracts/UtilityStake.sol

In the current reward mechanism, a significant issue arises if users claim their rewards after an update to the reward rate. Specifically, if a user claims rewards accrued before the rate update, the calculation will be based on the new, potentially higher rate, leading to an overestimation of their rightful reward. This overpayment can deplete the reward pool more rapidly than anticipated, potentially leaving insufficient funds for later users. Consequently, this could impede the normal operation of unstake and stake functions, as the reward pool might not sustain the demands.

```
function depositRewardToken(address token, uint256 amount) external {
    IUtilityToken rewardToken = config.getPRToken();
    require(token == address(rewardToken), "UtilityStake: invalid token");
    rewardToken.transferFrom(msg.sender, address(this), amount);
    if (block.timestamp >= periodFinish) {
        rate = amount / periodDuration;
```

```
    } else {
        uint256 remaining = periodFinish - block.timestamp;
        uint256 leftover = remaining * rate;
        rate = (amount + leftover) / periodDuration;
    }
    periodFinish = block.timestamp + periodDuration;
    emit DepositRewardToken(amount);
}
```

**Solution**

Modification of the reward model. Allow users to receive rewards correctly

**Status**

Fixed; Function Delete Deactivate.

fix commit:

557ed44fa4f81ccfe7e8e493e66df839395d8aef

### [N4] [Critical] Inaccuracies in reward calculation due to misuse of total supply in liquidity pool

**Category: Others**

**Content**

- contracts/LiquidityStake.sol

In the `_checkpoint` function, the use of `_sc2crvPool().totalSupply()` for calculating rewards may lead to inaccuracies. This is because it utilizes the total supply of the entire liquidity pool, rather than the quantity of LP tokens controlled by the contract itself. Indeed, this approach can result in the calculation of rewards being less than what is rightfully due, consequently leading to users receiving fewer rewards than they are actually entitled to.

```
function _checkpoint(address staker) internal {
    _metaGauge().claim_rewards();
    StakeInfo memory stakeInfo = stakeInfoByStaker[staker];
    uint256 totalSupplay = _sc2crvPool().totalSupply();
    if (totalSupplay == 0) {
        return;
    }
    for (uint256 i = 0; i < MAX_REWARDS; i++) {
        address tokenAddress = _metaGauge().reward_tokens(i);
        if (tokenAddress == address(0)) {
            break;
        }
```

```
        uint256 dI = 0;
        uint256 tokenBalance = IERC20Metadata(tokenAddress).balanceOf(
            address(this)
        );
        dI =
            (10 ** 18 * (tokenBalance - rewardBalances[tokenAddress])) /
            totalSupplay;
        rewardBalances[tokenAddress] = tokenBalance;

        // integral: uint256 = self.reward_integral[token] + dI
        uint256 integral = rewardIntegral[tokenAddress] + dI;
        if (dI != 0) {
            rewardIntegral[tokenAddress] = integral;
        }

        uint256 integralFor = rewardIntegralFor[tokenAddress][staker];
        uint256 newClaimable = 0;

        if (integralFor < integral) {
            rewardIntegralFor[tokenAddress][staker] = integral;
                (stakeInfo.totalSC2CRVLP * ((integral - integralFor))) /
                PRICE_PRECISION;
        }

        uint256 claimData = claimDataByStaker[staker][tokenAddress];
        uint256 totalClaimable = (claimData >> 128) + newClaimable;

        if (totalClaimable > 0) {
            uint256 totalClaimed = claimData % 2 ** 128;
                claimDataByStaker[staker][tokenAddress] =
                    totalClaimed +
                    (totalClaimable << 128);
            }
        }
    }
}
```

**Solution**

The total amount of LPs in the current contract should be used for the calculation.

**Status**

Fixed; fix commit:

557ed44fa4f81ccfe7e8e493e66df839395d8aef

### [N5] [Low] Redundant logic in _setFloorPrice function of smart contract

**Category: Others**

**Content**

- contracts/PriceField.sol

The `else if (_floorPrice == 0)` conditional branch in this function will not actually be executed. This is because `_floorPrice` is set to the new `floorPrice_` before any conditional judgment is entered, and `floorPrice_` cannot be zero, as verified at the beginning of the function.

```
function _setFloorPrice(uint256 floorPrice_) internal {
    require(floorPrice_ >= PRICE_PRECISION / 2, "floor price too low");
    require(floorPrice_ > _floorPrice, "floor price too low");
    uint256 previousFloorPrice = _floorPrice;
    uint256 x3 = _config.getUtilityToken().totalSupply();
    _floorPrice = floorPrice_;
    if (x3 > c()) {
        uint256 maxFloorPrice = (Math.mulDiv(
            x3 - c(),
            _slope,
            PRECENT_DENOMINATOR,
            Math.Rounding.Zero
        ) + PRICE_PRECISION) / 2;
        if (maxFloorPrice > floorPrice_) {
            _floorPrice = floorPrice_;
        }
    } else if (_floorPrice == 0) { //SLOWMIST//will not be implemented
        _floorPrice = floorPrice_;
    } else if (x3 > x1(floorPrice_) + _exerciseAmount) {
        _floorPrice = floorPrice_;
    } else if (x3 == 0) {
        _floorPrice = floorPrice_;
    }

    require(_floorPrice > previousFloorPrice, "floor price too low");
    emit UpdateFloorPrice(_floorPrice);
}
```

**Solution**

Check if this fits the design and remove the logical branch if you are sure it's not working.

**Status**

Acknowledged

### [N6] [High] Potential reentrancy risk in VAMM's _mintByPRToken function

**Category: Others**

**Content**

- contracts/VAMM.sol

The `payToken` can be passed in arbitrarily from the outside, and if the `payToken` is a token that implements a callback function, then this call may trigger malicious code.An attacker can exploit this by calling the `_mintByPRToken` function again during the callback. Since the `_mintByPRToken` function calls `payToken.transferFrom` before updating `_floorPrice` and minting new tokens, a reentry attack could allow an attacker to mint tokens multiple times at the old, more favorable price, rather than at the updated price. This could lead to improper minting of assets.

```
function _mintByPRToken(
    address payTokenAddress,
    uint256 mintAmount,
    uint256 maxPayAmount,
    address recipient
) internal {
    IERC20Metadata payToken = IERC20Metadata(payTokenAddress);
    IUtilityToken _utilityToken = _config.getUtilityToken();
    IUtilityToken _prToken = _config.getPRToken();

    require(_prToken.balanceOf(msg.sender) >= mintAmount, "VAMM:mp0");
    require(
        _prToken.allowance(msg.sender, address(this)) >= mintAmount,
        "VAMM:mp1"
    );
    (uint256 toLiquidityPrice, uint256 fees) = _priceField.getUseFPBuyPrice(
        mintAmount
    );
    require(toLiquidityPrice + fees <= maxPayAmount, "VAMM:mp2");

    // Include slippage as fee income
    fees = maxPayAmount - toLiquidityPrice; //SLOWMIST//
    uint256 maxPayAmountInPayToken = _convertPrice(
        maxPayAmount,
        payToken,
        true
    );

    _priceField.increaseSupplyWithNoPriceImpact(mintAmount);
```

```
    require(
        payToken.transferFrom(
            msg.sender,
            address(this),
            maxPayAmountInPayToken
        ),
        "VAMM:mp3"
    );//The paytoken is an arbitrary token, so any worthless token can be used as the key to transfer money to the contract.

    // burn pr token
    _prToken.burnFrom(msg.sender, mintAmount);

    _collectFees(fees);

    /// mint token
    _totalLiquidity += toLiquidityPrice;
    _utilityToken.mint(recipient, mintAmount);
    _deposit(payTokenAddress);

    _autoUpFP();

    emit Mint(recipient, mintAmount, toLiquidityPrice, fees);
}
```

**Solution**

This can be restricted using openzeppelin's ReentrancyGuard.

**Status**

Fixed

### [N7] [High] Exploitation risk with arbitrary payToken in VAMM's _mintByPRToken function

**Category: Others**

**Content**

- contracts/VAMM.sol

The `payToken` can be any token, so it can be used to construct a malicious token to make a payment.

```
function _mintByPRToken(
    address payTokenAddress,
    uint256 mintAmount,
    uint256 maxPayAmount,
```

```
    address recipient
) internal {
    IERC20Metadata payToken = IERC20Metadata(payTokenAddress);
    IUtilityToken _utilityToken = _config.getUtilityToken();
    IUtilityToken _prToken = _config.getPRToken();

    require(_prToken.balanceOf(msg.sender) >= mintAmount, "VAMM:mp0");
    require(
        _prToken.allowance(msg.sender, address(this)) >= mintAmount,
        "VAMM:mp1"
    );
    (uint256 toLiquidityPrice, uint256 fees) = _priceField.getUseFPBuyPrice(
        mintAmount
    );
    require(toLiquidityPrice + fees <= maxPayAmount, "VAMM:mp2");

    // Include slippage as fee income
    fees = maxPayAmount - toLiquidityPrice; //SLOWMIST//
    uint256 maxPayAmountInPayToken = _convertPrice(
        maxPayAmount,
        payToken,
        true
    );

    _priceField.increaseSupplyWithNoPriceImpact(mintAmount);

    require(
        payToken.transferFrom(
            msg.sender,
            address(this),
            maxPayAmountInPayToken
        ),
        "VAMM:mp3"
    );//The paytoken is an arbitrary token, so any worthless token can be used as the
key to transfer money to the contract.

    // burn pr token
    _prToken.burnFrom(msg.sender, mintAmount);

    _collectFees(fees);

    /// mint token
    _totalLiquidity += toLiquidityPrice;
    _utilityToken.mint(recipient, mintAmount);
    _deposit(payTokenAddress);

    _autoUpFP();
```

```
    emit Mint(recipient, mintAmount, toLiquidityPrice, fees);
}
```

**Solution**

Requires whitelisting control of payToken.

**Status**

Fixed

### [N8] [Suggestion] Preemptive Initialization

**Category: Race Conditions Vulnerability**

**Content**

- contracts/LiquidityStake.sol

```
function initialize(IConfig config_) public initializer {
    __Ownable_init();
    __UUPSUpgradeable_init();
    _transferOwnership(tx.origin);
    config = config_;
}
```

- contracts/UtilityStake.sol

```
function initialize(IConfig config_) public initializer {
    __Ownable_init();
    __UUPSUpgradeable_init();
    _transferOwnership(tx.origin);
    config = config_;

    unstakeFee = 300000000;
    periodDuration = 1 weeks;
    timeoutClaimPeriod = 2 days;
}
```

- contracts/VAMM.sol

```
function initialize(
    IConfig config_,
    PriceField priceField_,
    uint256 t_,
```

```
    uint256 x_,
    uint256 c_
) public initializer {
    __Ownable_init();
    __UUPSUpgradeable_init();
    _transferOwnership(tx.origin);
    _config = config_;
    _priceField = priceField_;

    tForMFR = t_;
    maxTForMFR = 5000000000;
    minTForMFR = t_;
    cForMFR = c_;
    // ethereum block time 13s
    reduceTBlocks = 6600;
    xForMFR = x_;
}
```

**Solution**

It is suggested that the initialize operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

**Status**

Acknowledged

### [N9] [Suggestion] Lacking event logging in critical contract functions alters state without transparency issue

**Category: Malicious Event Log Audit**

**Content**

- contracts/LiquidityStake.sol

```
function setHook(LiquidityStakeHook hook_) external onlyOwner {
    hook = hook_;
}
```

- contracts/UtilityStake.sol

```
function setHook(UtilityStakeHook hook_) external onlyOwner {
    hook = hook_;
```

```
}
```

- contracts/PriceField.sol

```
function increaseSupplyWithNoPriceImpact(uint256 amount) external onlyVamm {
    _exerciseAmount += amount;
}
```

- contracts/VAMM.sol

```
function setPriceField(PriceField priceField_) external onlyOwner {
    _priceField = priceField_;
}
```

**Solution**

Recording events

**Status**

Acknowledged

### [N10] [Information] Redundant functions

**Category: Others**

**Content**

- contracts/VAMM.sol

Redundant function code, which can be deleted if it is not useful.

```
function liquidityTesting() external {

}
```

**Solution**

Delete Code

**Status**

Fixed

### [N11] [Low] Missing check return value

**Category: Others**

**Content**

It is recommended to check the return value of transferFrom, as there may be problems if a non-ERC20 standard token is subsequently used.

- contracts/UtilityStake.sol

```
line 142: utilityToken.transferFrom(staker, address(this), amount);
line 196: stablecoin.transferFrom(staker, address(this), fees);
line 287: rewardToken.transferFrom(msg.sender, address(this), amount);
```

- contracts/LiquidityStake.sol

```
line 154: stablecoin.transferFrom(staker, address(this), _amount);
```

- contracts/VAMM.sol

```
line 504: token.transferFrom(msg.sender, address(this), _repayAmount);
```

**Solution**

It is recommended to check the return value or use the SafeERC20.

**Status**

Acknowledged

### [N12] [Medium] Risk of excessive authority

**Category: Authority Control Vulnerability Audit**

**Content**

- contracts/VAMM.sol

The owner can set the key parameters, if the private key is lost, the price will be out of control.

```
owner can setPriceField
operator can updateMFR
```

- contracts/UtilityToken.sol

The owner can mint the token, if the private key is lost it will cause the token to be incremented.

```
owner can mint
owner can transferOwnership
```

Other contracts have key roles and key parameters that are mostly controlled by external `config` contracts.

**Solution**

In the short term, in order to cope with the scenario that the protocol needs to frequently set parameters in the early stage, the Admin can be divided into two roles, one is an EOA address, which is used to manage the protocol's emergency pause permission, and the other is a multisign address, which is used to manage necessary parameter configuration and modification. This can solve the single-point risk without losing too much flexibility, but it cannot effectively mitigate the risk of excessive privileges. In the long run, it is more reasonable to entrust the protocol's parameter configuration and modification permissions to the timelock contract, and to entrust the timelock contract to community governance can effectively mitigate the risk of excessive privileges. This can also improve the trust of community users in the protocol.

**Status**

Acknowledged; The owner of the UtilityToken has been transferred to the VAMM contract. The owner of the VAMM contract is controlled by a multi-signature contract.

Multi-signatory contract address:0x7e502E2cF358d9191EB5ae437b7e6d1e674E7498

### [N13] [Suggestion] Recommendation to Implement reentrancy

**Category: Reentrancy Vulnerability**

**Content**

- contracts/VAMM.sol

In `_mint`, `_burn`, there is no utilization scenario at the moment, but it is recommended to add reentrant locks.

**Solution**

This can be restricted using openzeppelin's ReentrancyGuard.

**Status**

Acknowledged

# 5 Audit Result

| Audit Number | Audit Team | Audit Date | Audit Result |
| --- | --- | --- | --- |
| 0X002312220001 | SlowMist Security Team | 2023.12.13 - 2023.12.22 | Low Risk |

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 critical risk, 3 high risk, 2 medium risk, 2 low risk, 3 suggestion vulnerabilities.

# 6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

**Official Website**

www.slowmist.com

**E-mail**

team@slowmist.com

**Twitter**

@SlowMist_Team

**Github**

https://github.com/slowmist