VIDMA

POPNETWORK

SMART CONTRACT AUDIT

Project: POP Network
Date: March 16th, 2022

TABLE OF CONTENTS

SUMMARY

Vidma team has conducted a smart contract audit for the given codebase.

We made deep auditing of the codebase provided and can confirm that the contact is in good condition.

During the auditing process, the Vidma team has found several issues, including issues with critical severity. A detailed summary of the issues and their current state is displayed in the table below.

Severity of the issue	Total found	Resolved	Unresolved
Critical	2 issues	2 issues	0 issues
High	0 issues	0 issues	0 issues
Medium	0 issues	0 issues	0 issues
Low	12 issues	12 issues	0 issues
Informational	4 issues	4 issues	0 issues
Total	**18 issues**	**18 issues**	**0 issues**

Evaluating the findings, we can assure that the contract is safe to use and all the issues found are performed only by certain conditions and cases. Under the given circumstances we can set the following risk level:

Vidma auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. Hence, it helps to adequately evaluate the development quality. Code style, optimization of the contracts, amount, and risk level of the issues are taken into consideration. The Vidma team has developed the transparent scoring system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Based on the given findings, risk level, performance, and code style, Vidma team can grant the following overall score:

Vidma auditing team has conducted a bunch of integrated autotests to ensure that the given codebase has decent performance and security levels. The test results and the coverage can be found in the accompanying section of this audit report.

Please mind that this audit does not certify the definite reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by Vidma auditing team. If the code is under development, we recommend run one more audit once the code is finalized.

SCOPE OF WORK

POPNETWORK

POP Network is a user-friendly ecosystem of blockchain and AI applications built to power the streaming economy. Their vision is a universal system for turning attention into real-world goods and services.

Within the scope of this audit, two independent auditors deeply investigated the given codebase and analyzed the overall security and performance of smart contracts.

The debrief took place from March 7th to 16th, 2022 and the final results are present in this document.

Vidma auditing team has made a review of the following contracts:

- PopStaking

The source code was taken from the following **sources (zip archive)**:
https://drive.google.com/file/d/13I7LoIMIK6JTLg_OQnh1T6Ap_9XA4ji_/view?usp=sharing

Last commit:
256a5d40fb42b03912bf3a0386713f363e8bc4ae

To conduct a more detailed audit, POP Network has provided the following **documentation**:
https://drive.google.com/drive/folders/1vVAnJ01cI92SK6UzdUwaNNUex-4Z6yMw?usp=sharing

WORKFLOW OF THE AUDITING PROCESS

During the manual phase of the audit, Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract.

Within the testing part, Vidma auditors run integration tests using the Truffle testing framework. The test coverage and the tests themselves are inserted into this audit report.

Vidma team uses the most sophisticated and contemporary methods and techniques to ensure the contract does not have any vulnerabilities or security risks:

- Re-entrancy;
- Access Management Hierarchy;
- Arithmetic Over/Under Flows;
- Unexpected Ether;
- Delegatecall;
- Default Public Visibility;
- Hidden Malicious Code;
- Entropy Illusion (Lack of Randomness);
- External Contract Referencing;
- Short Address/Parameter Attack;
- Unchecked CALL Return Values;
- Race Conditions / Front Running;
- General Denial Of Service (DOS);
- Uninitialized Storage Pointers;
- Floating Points and Precision;
- Tx.Origin Authentication;
- Signatures Replay;
- Pool Asset Security (backdoors in the underlying ERC-20).

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For the convenience of reviewing the findings in this report, Vidma auditors classified them in accordance with the severity of the issues. (from most critical to least critical). The acceptance criteria are described below.

All issues are marked as "Resolved" or "Unresolved", depending on whether they have been fixed by POP Network or not. The latest commit, indicated in this audit report should include all the fixes made.

To ease the explanation, the Vidma team has provided a detailed description of the issues and recommendations on how to fix them.

Hence, according to the statements above, we classified all the findings in the following way:

Finding	Description
Critical	The issue bear a definite risk to the contract, so it may affect the ability to compile or operate.
High	Major security or operational risk found, that may harm the end-user or the overall performance of the contract.
Medium	The issue affects the contract to operate in a way that doesn't significantly hinder its performance.
Low	The found issue has a slight impact on the performance of the contract or its security.
Informational	The issue does not affect the performance or security of the contract/recommendations on the improvements.

MANUAL REPORT

Too centralized pending reward info updating

Critical | Invalid

Only *devaddr* can call *updatePendingInfo()* function where reward multiplier is changed. When in some cases this function won't call or will call with some delays it will lead to outdated info that triggers incorrect reward distribution so the user will be able to lose his accumulated reward.

Recommendation:

Consider fixing the updating reward multiplier in a more trusted and decentralized way. Or add some validation to prevent user deposit or withdrawal before the pending reward info won't be updated.

Re-Audit:

Explanation from the POP Network team: "We are logging masternode running time on server and update staking contract on the server-side. We are updating these infos every some period and we let users know when we update infos. We know it's too centralized but it's a good choice for now."

Incorrect logic in *emergencyWithdraw()* function

Critical | Resolved

In the *emergencyWithdraw()* function user should be able to withdraw the available deposited amount. In the current logic before transferring the user's funds *user.amount* value is set to zero in line 119 and only after that appears transfer where the passed amount is variable *user. amount* which was set to zero before. In this case, the user can't be able to withdraw his funds and after a call of *emergencyWithdraw()* function user's funds will be locked in the contract balance without any chance to withdraw it.

```
function emergencyWithdraw() public {
    UserInfo storage user = userInfo[msg.sender];
    user.amount = 0;
    user.lastRewardBlock = block.number;
    user.rewardMultiplier = 0;
    pop.transfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(msg.sender, user.amount);
}
```

Recommendation:

Consider fix logic of the *emergencyWithdraw()* function.

Potential loss of accumulated user's reward

Critical | Resolved

The reward is transferred to the user when the user deposits and withdraws. To transfer rewards used *safePopTransfer()* function. When the staking contract won't have enough pop tokens to cover the reward distribution all available tokens will be transferred to the user which will be less than the actually accumulated reward. There is no tracking of the reward transferring so if the user withdraws staked tokens as it is (with fewer rewards amount) he won't be able to withdraw the missing amount.

Recommendation:

Consider managing case when there are not enough tokens to withdraw for reward in that case so the user will not loss.

Unchecked transfer return value

Low | Resolved

Ignored *return* value of *transferFrom* and *transfer* in the functions *deposit*, *withdraw*, *emergencyWithdraw*, *safePopTransfer* in lines 94, 112, 122, 131, 133.

Recommendation:

Consider validating return value by wrapping *transfer*/*transferFrom* in require or using SafeERC20 library.

Re-Audit:

In line 13 there is defined library usage using SafeERC20Upgradeable for IERC20; but functionality from SafeERC20Upgradeable library is not used in the codebase. For transfer tokens used simple *transfer*/*transferFrom* but not *safeTransfer*/*safeTransferFrom*.

Re-Audit 2:

Fixed.

Lack of validation in *deposit()* function

Low | Resolved

There is no check if the given param *_amount* is available for the deposit as the available amount for staking starts from 50K POP tokens.

Recommendation:

Consider adding a check for the local variable “amount” to check if the depositing amount isn’t zero.

Re-Audit:

Comment from POP Network team: “We use deposit function for claim rewards. So if we input amount as zero, then it only claims rewards.”

We may suggest you add separate function for claiming rewards, so user can spend less gas for calling claiming transaction.

Re-Audit 2:

Fixed.

No check for zero value in claimable rewards

Low | Resolved

In line 94 of function *deposit()* and in line 109 of function *withdraw()* there is no check if claimable user rewards are greater than 0.

Recommendation:

Consider adding a check for claimable amounts to not perform token transfer if the amount of rewards is equal to 0.

Floating pragma

Low | Resolved

The current version of solc is ^0.6.12 and it is better to lock the pragma to a specific version.

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation:

Lock pragma to a specific version.

Unspecified state variable visibility

Low | Resolved

It is best practice to set the visibility of state variables explicitly. The default visibility for *cycleLen* in line 28 is internal.

Recommendation:

Explicitly specify the visibility of state variables.

Lack of zero address check

Low | Resolved

In functions *initialize()* and *dev()* there is no check if the *_devaddr* address isn't a zero address.

Recommendation:

Consider adding a check for zero address for functions *initialize()* and *dev()*.

Functions visibility optimization

Low | Resolved

Functions *getMultiplier()*, *deposit()*, *withdraw()*, *emergencyWithdraw()*, *updatePendingInfo()* and *dev()* can be marked as external as they are not used in the contract.

Recommendation:

Consider changing visibility from public to external to safe gas usage on calling these functions.

Function state visibility optimization

Low | Resolved

Functions *getMultiplier()* state visibility can be restricted to pure.

Recommendation:

Consider changing state visibility in *getMultiplier()* from view to pure.

Lack of validation in *withdraw()* function

Low | Resolved

There is no check if the given param *_amount* isn't zero.

Recommendation:

Consider adding a check for zero value.

Lack of validation in *emergencyWithdraw()* function

Low | Resolved

There is no check if the user has any funds to withdraw.

Recommendation:

Consider adding a check for zero value.

Lack of validation in *UpdatePopPerBlock()* function

Low | Resolved

There is no check if the input param *_popPerBlock* isn't zero.

Recommendation:

Consider adding a check for zero value.

Division by zero in *updatePendingInfo()* funciton

Low | Resolved

In line 141 *getPopPerBlock()* function can return 0.

Recommendation:

Consider managing case when function *getPopPerBlock()* return zero while calculating value of *divider*.

Lack of NatSpec annotations

Informational | Resolved

Smart contract Stakel is not covered by NatSpec annotations.

Recommendation:

Consider covering by NatSpec all contract methods.

Re-Audit:

Defined only @notice section but @param and @return sections are not disclosed.

Re-Audit 2:

Fixed.

Commented import statement

Informational | Resolved

In line 5 in the codebase there commented import statement for SafeERC20.sol, but in line 6 there used SafeERC20Upgradeable.sol.

Recommendation:

Remove redundant import statement.

Naming style

Informational | Resolved

Constant *stakeUnit* is not in *UPPER_CASE_WITH_UNDERSCORES*.

Recommendation:

Consider fixing the naming style according to the solidity style guide.

Lack of documentation

Informational | Resolved

The POP Network team didn't share any type of documentation with the Vidma team.

Recommendation:

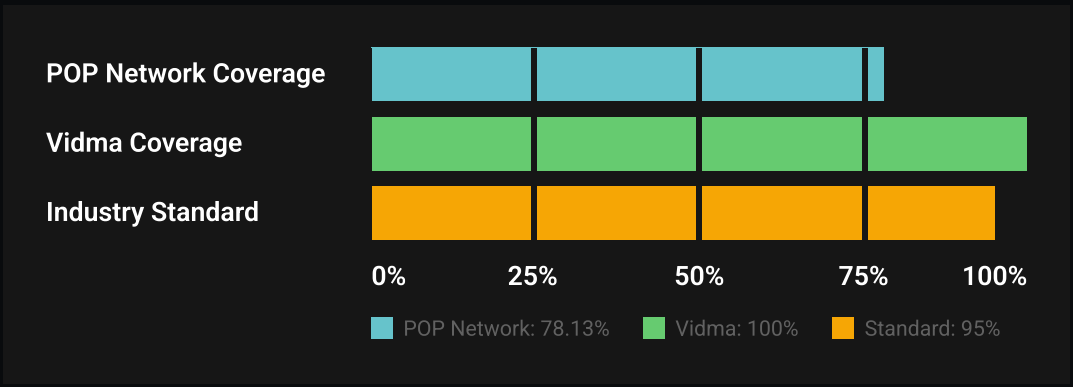
Consider providing some documentation.

TEST RESULTS

To verify the contract security and performance a bunch of integration tests were made using the Truffle testing framework.

Tests were based on the functionality of the code, business logic, and requirements and for the purpose of finding the vulnerabilities in the contacts.

In this section, we provide both tests written by POP Network and Vidma auditors.



It's important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the POP Network repo. We write totally separate tests with code coverage of a minimum of 95%, to meet the industry standards.

Tests are written by POP Network

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	78.13	44.44	72.73	81.25
PopStaking.sol	78.13	44.44	72.73	81.25
All files	**78.13**	**44.44**	**72.73**	**81.25**

Test Results

```
PopStaking contract
  Deployment
    ✓ Should set the right owner
    ✓ Should set correct state variables (53ms)
  Staking
    ✓ should give out POPs only after staking time - 1 (266ms)
    ✓ should give out POPs only after staking time - 2 (297ms)
    ✓ should give out POPs only after staking time - 3 (152ms)
    ✓ should give out POPs only after staking time - 4 (320ms)

6 passing (5s)
```

Tests are written by Vidma

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
PopStaking.sol	100.00	100.00	100.00	100.00
All files	**100.00**	**100.00**	**100.00**	**100.00**

Test Results

```
Contract: PopStaking
  PopStaking Initializing Test Cases
    ✓ shouldn't initialize with dev address setted to the zero
      address (915ms)
    ✓ shouldn't initialize with pop per block equal 0 (3945ms)
    ✓ should deploy with correct owner (70ms)
    ✓ should deploy with stake unit (65ms)
    ✓ should initialize with correct pop token (89ms)
    ✓ should initialize with correct dev address (91ms)
    ✓ should initialize with correct start time (40ms)
    ✓ should initialize with correct start block (94ms)
    ✓ should initialize with correct claimable block (52ms)
    ✓ should initialize with correct pop per block for all
      cycles (205ms)
    ✓ should return correct pop per block before start date (54ms)
    ✓ should return correct pop per block after start date (151ms)
    ✓ should return correct multiplier (139ms)
  PopStaking Deposit/Withdraw Test Cases
    ✓ shouldn't deposit 0 value (216ms)
    ✓ shouldn't deposit if transfer from user failed (301ms)
    ✓ should deposit correctly (766ms)
```

```
    ✓ should deposit correctly if deposit amount not a multiple
      of stake unit (691ms)
    ✓ should allow to deposit multiple times by one user (681ms)
    ✓ should return correct claimable pop when pending info not
      updated (62ms)
    ✓ should claim rewards when depositing (2061ms)
    ✓ shouldn't withdraw if user didn't deposit tokens (256ms)
    ✓ shouldn't withdraw if withdraw amount is greater than
      deposited (282ms)
    ✓ shouldn't withdraw when transfer failed (943ms)
    ✓ shouldn't withdraw when transfer failed (764ms)
    ✓ should withdraw correctly (1264ms)
    ✓ should withdraw when no rewards available for the user
      correctly (596ms)
    ✓ should update last reward after claiming rewards (310ms)
    ✓ should claim rewards during the first cycle correctly (1875ms)
    ✓ should claim rewards during the second cycle correctly (1553ms)
    ✓ should claim rewards during the third cycle correctly (1605ms)
    ✓ should claim rewards during the fourth cycle correctly (1452ms)
    ✓ should claim rewards after 10 cycles correctly (1412ms)
    ✓ shouldn't emergency withdraw if user don't have active
      stake (290ms)
    ✓ shouldn't emergency withdraw when transfer failed (638ms)
    ✓ should emergency withdraw correctly (802ms)
    ✓ shouldn't claim rewards when rewards amount is greater
      than contract balance (785ms)
  PopStaking Owner/Dev Test Cases
    ✓ shouldn't update pending info by not current the dev (217ms)
    ✓ shouldn't update pending info with different parameter
      length (172ms)
    ✓ should update  pending info correctly (447ms)
    ✓ shouldn't change dev address by not the current dev (173ms)
    ✓ shouldn't set dev address to the zero address (229ms)
    ✓ should change dev address correctly (279ms)
    ✓ shouldn't update pop per block by not the owner (187ms)
    ✓ shouldn't set pop per block to equal 0 (184ms)
    ✓ should update pop per block correctly (692ms)

  45 passing (38s)
```

We are delighted to have a chance to work together with POP Network team and contribute to their success by reviewing and certifying the security of the smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Website: **vidma.io**
Email: **security@vidma.io**

VIDMA