SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Populous

Date: December 31rd, 2020

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Populous.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Multiple purposes contracts
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/bitpopulous/defi_audits/
Commit	3944B72830AA1F514ECBEEFCD11EDBF3EC377C53
Deployed contract	
Timeline	21 DEC 2020 – 31 DEC 2020
Changelog	30 DEC 2020 – INITIAL AUDIT

Table of contents

Introduction 4
Scope 4
Executive Summary 5
Severity Definitions 6
AS-IS overview 7
Conclusion 29
Disclaimers 30

Introduction

Hacken OÜ (Consultant) was contracted by Populous (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted between December 21st, 2020 – December 31st, 2020.

Scope

The scope of the project is smart contracts in the repository:

Contract deployment address:
Repository
Commit
Files:
/reward/RewardPoolAddressManager.sol
/reward/RewardPool.sol
/lendingpool/LendingPoolConfigurator.sol
/lendingpool/LendingPoolDataProvider.sol
/lendingpool/LendingPool.sol
/governance/governance/PopulousProtoGovernance.sol
/governance/governance/governance/PopulousPropositionPower.sol
/governance/governance/GovernanceParamsProvider.sol
/governance/governance/AssetVotingWeightProvider.sol

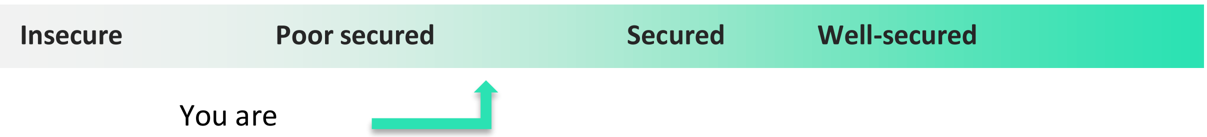
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts has some issues that should be fixed.

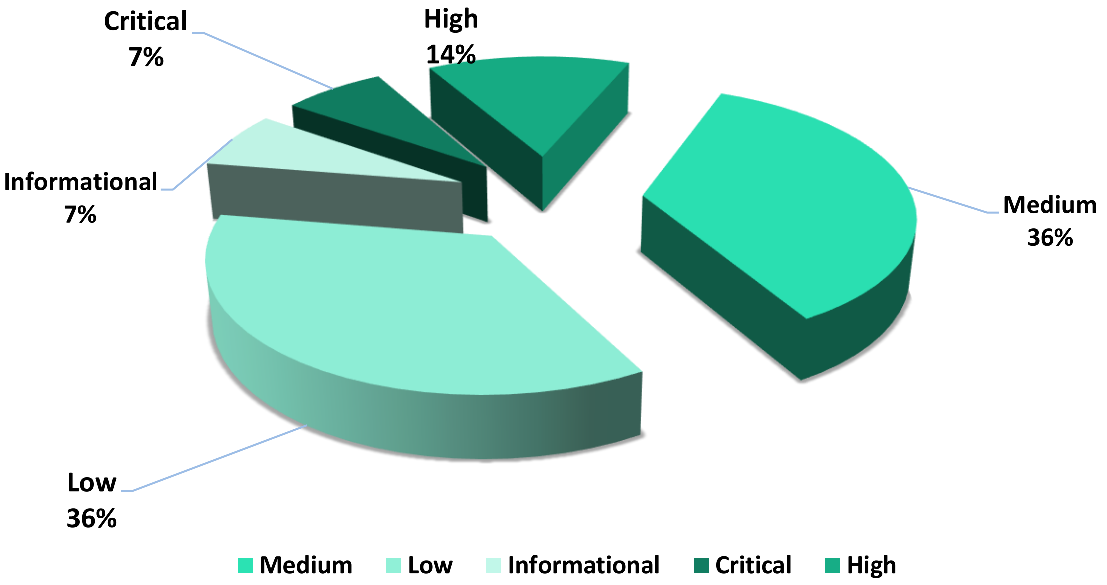


Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found **1** critical, **2** high, **5** medium, **5** low, and **2** informational issue during the audit.

Notice: some contracts in the repository are not in the audit scope. They can be used by or can use contacts from the scope. During the audit we consider out-of-scope contracts as secure but cannot guaranty that they really are. We recommend reviewing those contracts before using the system. Due to the limited scope, we cannot guarantee that the whole system will work properly all together. We recommend performing the full audit and UAT testing at the production environment as it can reveal issues which cannot be reproduced during the audit.

Graph 1. The distribution of vulnerabilities after the first review.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

LendingPool.sol

Description

LendingPool is a contract used to provide a loans and flash-loans functionality.

Imports

LendingPool contract has the following imports:

- @openzeppelin/contracts/math/SafeMath.sol
- @openzeppelin/contracts/utils/ReentrancyGuard.sol
- @openzeppelin/contracts/utils/Address.sol
- @openzeppelin/contracts/token/ERC20/IERC20.sol
- ../libraries/openzeppelin-upgradeability/VersionedInitializable.sol
- ../configuration/LendingPoolAddressesProvider.sol
- ../configuration/LendingPoolParametersProvider.sol
- ../tokenization/PToken.sol
- ../libraries/CoreLibrary.sol
- ../libraries/WadRayMath.sol
- ../interfaces/IFeeProvider.sol
- ../flashloan/interfaces/IFlashLoanReceiver.sol
- ./LendingPoolCore.sol
- ./LendingPoolDataProvider.sol
- ./LendingPoolLiquidationManager.sol
- ../libraries/EthAddressLib.sol
- ./LendingPoolConfigurator.sol
- ./DefaultReserveInterestRateStrategy.sol

Inheritance

LendingPool contract is ReentrancyGuard, VersionedInitializable**.**

Usages

LendingPool contract has following usages:

- SafeMath for uint256.
- WadRayMath for uint256.
- Address for address.

Structs

LendingPool contract has following data structures:

- BorrowLocalVars – used for local computations in the `borrow` function.
- RepayLocalVars – used for local computations in the `repay` function.

Enums

LendingPool contract has no custom enums.

Events

LendingPool contract has following events:

- Deposit – emitted on deposit.
- RedeemUnderlying – mitted during a redeem action.
- Borrow – emitted on borrow.
- Repay – emitted on repay.
- Swap - emitted when a user performs a rate swap.
- ReserveUsedAsCollateralEnabled – emitted when a user enables a reserve as collateral.
- ReserveUsedAsCollateralDisabled – emitted when a user disables a reserve as collateral.
- RebalanceStableBorrowRate – emitted when the stable rate of a user gets rebalanced.
- FlashLoan – emitted when a flashloan is executed.
- OriginationFeeLiquidated – emitted when a borrow fee is liquidated.
- LiquidationCall – emitted when a borrower is liquidated.

Modifiers

LendingPool has the following modifiers:

- onlyOverlyingPToken – functions affected by this modifier can only be invoked by the PToken contract.
- onlyActiveReserve - functions affected by this modifier can only be invoked if the reserve is active.
- onlyUnfreezedReserve – functions affected by this modifier can only be invoked if the reserve is not frozen.
- onlyAmountGreaterThanZero – functions affected by this modifier can only be invoked if the provided `_amount` input parameter is not zero.

Fields

LendingPool contract has following constants and fields:

- LendingPoolAddressesProvider public addressesProvider
- LendingPoolCore public core
- LendingPoolDataProvider public dataProvider
- LendingPoolParametersProvider public parametersProvider
- IFeeProvider feeProvider
- uint256 public constant UINT_MAX_VALUE = uint256(-1)
- uint256 public constant LENDINGPOOL_REVISION = 0x5

Functions

LendingPool has following public functions:

- ***initialize***
  Description
  Initializes the contract.
  Visibility
  public
  Input parameters
  - LendingPoolAddressesProvider _addressesProvider – the address of the LendingPoolAddressesProvider registry.

  Constraints
  - Can only be called once.

  Events emit
  None
  Output
  None
- ***deposit***
  Description
  Deposits the underlying asset into the reserve. A corresponding amount of the overlying asset (PTokens) is minted.
  Visibility
  external payable
  Input parameters
  - address _reserve – the reserve address.
  - uint256 _amount – an amount to be deposited.
  - uint16 _referralCode – referral code.

  Constraints

- onlyActiveReserve modifier.
- onlyUnfreezedReserve modifier.
- onlyAmountGreaterThanZero modifier.

Events emit

Emits the Deposit event.

Output

None

- ***redeemUnderlying***

  Description

  Redeems the underlying amount of assets requested by `_user`.

  Visibility

  external

  Input parameters

  - address _reserve – the reserve address.
  - address payable _user – the address of the user performing the action.
  - uint256 _amount – the underlying amount to be redeemed.
  - uint256 _PTokenBalanceAfterRedeem – PToken balance after redeem.

  Constraints

  - onlyOverlyingPToken modifier.
  - onlyActiveReserve modifier.
  - onlyAmountGreaterThanZero modifier.
  - The `_amount` should be less or equal to `currentAvailableLiquidity`.

  Events emit

  Emits the RedeemUnderlying event.

  Output

  None

- ***calculateUserReserveCollateralETHInvoicePool***

  Description

  Redeems the underlying amount of assets requested by `_user`.

  Visibility

  public view

  Input parameters

  - address _reserve – the reserve address.
  - address payable _user – the address of the user performing the action.
  - uint256 _amount – the underlying amount to be redeemed.

- uint256 _PTokenBalanceAfterRedeem – PToken balance after redeem.

Constraints

- Stable interest rates should be set for reserve to borrow from.
- Only stable rate mode allowed.
- A `userCollateralBalanceETH` should not exceed the `amountOfCollateralNeededETH`.

Events emit

None

Output

- bool – always true.
- uint256 – userCollateralBalanceETH
- uint256 – amountOfCollateralNeededETH

- ***borrow***

Description

Allows users to borrow a specific `amount` of the reserve underlying asset, provided that the borrower already deposited enough collateral.

Visibility

external

Input parameters

- address _reserve – the reserve address.
- uint256 _amount – an amount to be borrowed.
- uint256 _interestRateMode – the interest rate mode at which a user wants to borrow.
- uint16 _referralCode – a referral code.

Constraints

- onlyActiveReserve modifier.
- onlyUnfreezedReserve modifier.
- onlyAmountGreaterThanZero modifier.
- The `_amount` should be less or equal to `currentAvailableLiquidity`.
- Reserve should be enabled for borrowing.
- Only STABLE interest rate mode is allowed.
- The `_amount` should not exceed an availableLiquidity.
- The borrower health factor should not be below threshold.
- The borrow fee should be greater than 0.
- The borrower should have enough collateral balance to take a loan.

- The borrower should be allowed to borrow at stable interest rate mode.
- The `_amount` should not exceed a `maxLoanSizeStable`

Events emit

Emits the Borrow event.

Output

None

- ***repay***

Description

Repays a borrow on the specific reserve, for the specified amount (or for the whole amount, if uint256(-1) is specified).

Visibility

external

Input parameters

- address _reserve – the reserve address.
- uint256 _amount – an amount to repay.
- address payable _onBehalfOf – address for wich msg.sender is repaying.

Constraints

- onlyActiveReserve modifier.
- onlyAmountGreaterThanZero modifier.
- The user should have an active borrow.
- If a msg.sender is repaying a borrow for another address, an `_amount` should be an exact sum and cannot be max uint256 value.
- msg.value should be equal to `_value` if repay is in ETH.

Events emit

Emits the Repay event.

Output

None

- ***swapBorrowRateMode***

Description

Used to swap between stable and variable borrow rate modes.

Visibility

external

Input parameters

- address _reserve – the reserve address.

Constraints

- onlyActiveReserve modifier.
- onlyUnfreezedReserve modifier.
- msg.sender should have an active borrow.
- msg.sender should be allowed to borrow at stable mode if he wants to change it to the variable mode.

Events emit

Emits the Swap event.

Output

None

- ***rebalanceStableBorrowRate***

Description

Rebalances the stable interest rate of a user if current liquidity rate > user stable rate.

Visibility

external

Input parameters

- address _reserve – the reserve address.
- address _user – an address of the user to be rebalanced.

Constraints

- onlyActiveReserve modifier.
- `_user` should have an active borrow.
- `_user` should have a variable rate mode.

Events emit

Emits the RebalanceStableBorrowRate event.

Output

None

- ***setUserUseReserveAsCollateral***

Description

Allows depositors to enable or disable a specific deposit as collateral.

Visibility

external

Input parameters

- address _reserve – the reserve address.
- bool _useAsCollateral – true if a user wants to use the deposit as collateral, false otherwise.

Constraints

- onlyActiveReserve modifier.
- onlyUnfreezedReserve modifier.
- msg.sender should have deposited liquidity

  - Cannot be disabled if already being used as collateral.

Events emit

Emits ReserveUsedAsCollateralEnabled or ReserveUsedAsCollateralDisabled events.

Output

None

- ***liquidationCall***

  Description

  A proxy function used to liquidate an undercollateralized position.

- ***flashLoan***

  Description

  Allows smartcontracts to access the liquidity of the pool within one transaction, as long as the amount taken plus a fee is returned.

  Visibility

  external

  Input parameters

  - address _receiver – the loan receiver.
  - address _reserve – the reserve address.
  - uint256 _amount – the loan amount.
  - bytes memory _params – params to be passed to the receiver.

  Constraints

  - onlyActiveReserve modifier.
  - onlyAmountGreaterThanZero modifier.
  - `_amount` should not exceed an available liquidity.
  - _receiver should be a contract that implements the IFlashLoanReceiver interface.
  - The requested amount should be big enough so that fees will be greater than 0.
  - In the end of transaction, the contract balance should be equal to the initial balance plus fees.

  Events emit

  Emits the FlashLoan event.

- ***getReserveConfigurationData, getReserveData, getUserAccountData, getUserReserveData, getReserves***

  Description

  Simple view functions.

LendingPoolConfigurator.sol

Description

LendingPoolConfigurator allows lending pool manager to configure lending pool parametres. Contains only simple protected setter functions ang getters.

LendingPoolDataProvider.sol

Description

LendingPoolDataProvider contains only view functions that allows to receive all the necessary information about the Lending Pool.

AssetVotingWeightProvider.sol

Description

AssetVotingWeightProvider is a contract used to register whitelisted assets with its voting weight per asset. Only owner can change voting weights.

GovernanceParamsProvider.sol

Description

GovernanceParamsProvider is a contract used to store parameters of the governance contract. Only owner can change parameters.

PopulousPropositionPower.sol

Description

PopulousPropositionPower is an Asset to control the permissions on the actions in PopulousProtoGovernance.

PopulousProtoGovernance.sol

Description

PopulousProtoGovernance provides voting functionality.

Imports

PopulousProtoGovernance contract has the following imports:

- @openzeppelin/contracts/math/SafeMath.sol
- @openzeppelin/contracts/token/ERC20/IERC20.sol
- @openzeppelin/contracts/cryptography/ECDSA.sol
- ../interfaces/IGovernanceParamsProvider.sol
- ../interfaces/IAssetVotingWeightProvider.sol
- ../interfaces/IProposalExecutor.sol
- ../interfaces/IPopulousProtoGovernance.sol

Inheritance

PopulousProtoGovernance contract is IPopulousProtoGovernance**.**

Usages

PopulousProtoGovernance contract has following usages:

- SafeMath for uint256
- ECDSA for bytes32

Structs

PopulousProtoGovernance contract has following data structures:

- Voter – used to store vote result.
- Proposal – used to store a proposal info.

Enums

PopulousProtoGovernance contract has following enums:

- ProposalStatus {Initializing, Voting, Validating, Executed} – stores proposal statuses.

Events

PopulousProtoGovernance contract has following events:

- ProposalCreated – emitted when a new proposal is created.
- StatusChangeToVoting – emitted when a proposal status changes to Voting.

- StatusChangeToValidating – emitted when a proposal status changes to Validating.
- StatusChangeToExecuted – emitted when a proposal status changes to Executed.
- VoteEmitted – emitted on a new vote.
- VoteCancelled – emitted when a vote is cancelled.
- YesWins – emitted when a proposal wins with "Yes".
- NoWins – emitted when a proposal wins with "No".
- AbstainWins – emitted when a proposal wins with "Abstain".

Modifiers

PopulousProtoGovernance has no custom modifiers.

Fields

PopulousProtoGovernance contract has following constants and fields:

- uint256 public constant COUNT_CHOICES = 2
- uint256 public constant MIN_THRESHOLD = 13000000 ether
- uint256 public constant MIN_STATUS_DURATION = 1660;
- uint256 public constant MIN_MAXMOVESTOVOTINGALLOWED = 2
- uint256 public constant MAX_MAXMOVESTOVOTINGALLOWED = 6
- IGovernanceParamsProvider private govParamsProvider
- Proposal[] private proposals

Functions

PopulousProtoGovernance has following public functions:

- ***Fallback function***
  Description
  Forbid transferring ETH to the contract.
- ***newProposal***
  Description
  Registers a new proposal.
  Visibility
  external
  Input parameters
  - bytes32 _proposalType
  - bytes32 _ipfsHash
  - uint256 _threshold

  - address _proposalExecutor
  - uint256 _votingBlocksDuration
  - uint256 _validatingBlocksDuration
  - uint256 _maxMovesToVotingAllowed

  Constraints
  - A caller should have voting power greater or equal to threshold.
  - `_votingBlocksDuration` and `_validatingBlocksDuration` should be at least MIN_STATUS_DURATION.
  - `_maxMovesToVotingAllowed` should be between MIN_MAXMOVESTOVOTINGALLOWED and MAX_MAXMOVESTOVOTINGALLOWED.

  Events emit

  Emits the ProposalCreated event.

  Output

  None

- ***verifyParamsConsistencyAndSignature***

  Description

  Verifies the consistency of the action's params and their correct signature.

- ***verifyNonce***

  Description

  Verifies the nonce of a voter on a proposal.

- ***validateRelayAction***

  Description

- ***submitVoteByVoter***

  Description

  Function called by a voter to submit his vote directly.

  Visibility

  external

  Input parameters
  - uint256 _proposalId
  - uint256 _vote
  - IERC20 _asset

  Constraints
  - A proposal should be in the Voting status.
  - Asset weights of an `_asset` should be greater than 0.
  - `_vote` should be 0, 1 or 2.
  - The voter balance should be greater than 0.

  Events emit

Emits the VoteEmitted event. Also, can emit the StatusChangeToValidating event.

Output

None

- ***submitVoteByRelayer***

  Description

  Function called by any address relaying signed vote params from another wallet.

  Visibility

  external

  Input parameters

  - uint256 _proposalId
  - uint256 _vote
  - IERC20 _asset
  - uint256 _nonce
  - bytes calldata _signature
  - bytes32 _paramsHashByVoter

  Constraints

  - Signature and _nonce should be valid.
  - A proposal should be in the Voting status.
  - Asset weights of an `_asset` should be greater than 0.
  - `_vote` should be 0, 1 or 2.
  - The voter balance should be greater than 0.

  Events emit

  Emits the VoteEmitted event. Also, can emit the StatusChangeToValidating event.

  Output

  None

- ***cancelVoteByVoter***

  Description

  Revokes a vote in the proposal with `_proposalId`.

  Visibility

  external

  Input parameters

  - uint256 _proposalId

  Constraints

  - A proposal should be in the Voting status.

  Events emit

  Emits the VoteCancelled event.

Output

None

- ***cancelVoteByRelayer***

  Description

  Revokes a vote in the proposal with `_proposalId`.

  Visibility

  external

  Input parameters

  - uint256 _proposalId
  - address _voter
  - uint256 _nonce
  - bytes calldata _signature
  - bytes32 _paramsHashByVoter

  Constraints

  - Signature and _nonce should be valid.
  - A proposal should be in the Voting status.

  Events emit

  Emits the VoteCancelled event.

  Output

  None

- ***tryToMoveToValidating***

  Description

  Moves a proposal to the Validating status.

  Visibility

  external

  Input parameters

  - uint256 _proposalId

  Constraints

  - A proposal should be in the Voting status.
  - All the requirements of moving from Voting to Validating status should be met.

  Events emit

  Emits the StatusChangeToValidating event.

  Output

  None

- ***challengeVoters***

  Description

Called during the Validating period in order to cancel invalid votes where the voter was trying a double-voting attack.

Visibility

external

Input parameters

- uint256 _proposalId
- address[] calldata _voters

Constraints

- A proposal should be in the Validating status.

Events emit

Can emit StatusChangeToVoting and VoteCancelled events.

Output

None

- ***resolveProposal***

  Description

  Resolves a proposal if all requirements are met.

  Visibility

  external

  Input parameters

  - uint256 _proposalId

  Constraints

  - A proposal should be in the Validating status.
  - Validating period should pass.
  - A proposal should not be expired.

  Events emit

  Can emit YesWins, NoWins or AbstainWins events. Emits StatusChangeToExecuted event.

  Output

  None

- ***getLimitBlockOfProposal, getLeadingChoice, getProposalBasicData, getVoterData, getVotesData, getGovParamsProvider***

  Description

  Simple view functions.

RewardPool.sol

Description

RewardPool is a staking contract.

Imports

RewardPool contract has the following imports:

- ../tokenization/PToken.sol
- ../lendingpool/LendingPoolCore.sol

Inheritance

RewardPool contract is Ownable**.**

Usages

RewardPool contract has following usages:

- SafeMath for uint256
- SafeERC20 for IERC20
- Address for address

Structs

RewardPool contract has following data structures:

- UserInfo – stores staking amount of a user.

Enums

RewardPool contract has no enums.

Events

RewardPool contract has following events:

- RewardAdded – emitted when a Reward is added.
- Staked – emitted after a successful stake.
- Withdrawn – emitted after withdrawal.
- RewardPaid – emitted after successful reward claim.
- RewardDenied – never emitted.

Modifiers

RewardPool has the following modifiers:

- updateReward – functions affected by this modifier updates reward balance of a caller.

Fields

RewardPool contract has following constants and fields:

- LendingPoolCore public core

- PToken public pToken
- IERC20 public rewardToken
- uint256 public duration
- uint256 public periodFinish = 0
- uint256 public rewardRate = 0
- uint256 public lastUpdateTime
- uint256 public rewardPerTokenStored
- mapping(address => uint256) public userRewardPerTokenPaid
- mapping(address => uint256) public rewards
- address public exclusiveAddress
- mapping(address => UserInfo) internal userInfo

Functions

RewardPool has following public functions:

- ***constructor***
  Description
  Initializes the contract.
  Visibility
  public
  Input parameters
    - LendingPoolCore _core
    - address _rewardToken
    - address _reserve
    - uint256 _duration

  Constraints
  None
  Events emit
  None
  Output
  None
- ***lastTimeRewardApplicable, rewardPerToken, getuserinfo, earned***
  Description
  Simple view functions.
- ***stake***
  Description
  Stakes an `amount` of tokens.
  Visibility
  public

Input parameters

- uint256 amount

Constraints

- \`amount\` should be greater than 0.

Events emit

Emits the \`Staked\` event.

Output

None

- ***withdraw***

  Description

  Withdraw an \`amount\` of tokens.

  Visibility

  public

  Input parameters

  - uint256 amount

  Constraints

  - \`amount\` should be greater than 0.

  Events emit

  Emits the \`Withdrawn\` event.

  Output

  None

- ***withdraw***

  Description

  Withdraw an \`amount\` of tokens.

  Visibility

  public

  Input parameters

  - uint256 amount

  Constraints

  - \`amount\` should be greater than 0.

  Events emit

  Emits the \`Withdrawn\` event.

  Output

  None

- ***exit***

  Description

  Withdraw all tokens and rewards.

  Visibility

  public

  Input parameters

None

Constraints

- A caller should have active stake.

Events emit

Emits `Withdrawn` and `RewardPaid` events.

Output

None

- ***pushReward***

Description

Withdraw rewards of a `recipient`.

Visibility

public

Input parameters

- address recipient

Constraints

- onlyOwner modifier.

Events emit

Emits the `RewardPaid` event.

Output

None

- ***getReward***

Description

Withdraw rewards of a caller.

Visibility

public

Input parameters

- address recipient

Constraints

None

Events emit

Emits the `RewardPaid` event.

Output

None

- ***notifyRewardAmount***

Description

Withdraw rewards of a caller.

Visibility

external

Input parameters

- uint256 reward

Constraints

- onlyOwner modifier.
- \`reward\` should not exceed max uint value divided by 10^18.

Events emit

Emits the \`RewardAdded\` event.

Output

None

RewardPoolAddressManager.sol

Description

RewardPoolAddressManager is a contract used to deploy new RewadPool contracts. Can be used only by the owner.

Audit overview

🟥🟥🟥🟥 Critical

1. The `PopulousProtoGovernance` contract is not secured from double-voting. Manual calls of the challengeVoters function with a limited list of voters is not enough.

   We recommend allowing a proposal resolving only after all voters are validated or to redesign a way votes are collected.

🟨🟨🟨 High

1. The `setReserveDecimals` function of the `LandingPoolConfiguration` allows to specify a reserve token decimals manually.

   We recommend changing this function to `updateReserveDecimals` and take decimals value directly from a token.

2. The `borrow` function of the `LendingPool` allows borrows only with the STABLE interest rate mode. Such behavior is enforced by the `calculateUserReserveCollateralETHInvoicePool` function.

   If it is done intentionally, we recommend removing the `_interestRateMode` parameter and not allow to pass this value to the function.

🟩🟩 Medium

1. The `initReserveWithData` function of the `LandingPoolConfigurator` is lack of validations.

   Consider validation for a reserve existence.

2. The `addReserve` function of the `RewardPoolAddressManagercan` can be used to overwrite reward pool that is already exist.

   We recommend to add validation for this case.

3. Purpose of the `liquidationCall` of the `LendingPool` function is unknown and it can be called by anyone. The underlying contract is out of the audit scope.

   We recommend the Customer to ensure that it’s safe to allow everyone to call this function.

4. Old compiler version is used.

   We recommend updating to the latest stable one.

5. The `UserInfo` data structure of the RewardPool contains only 1 field and can be removed to optimize gas consumption.

Low

1. The `ReentrancyGuard` inheritance in the LendingPool is redundant because `nonReentrant` modifier is never used.

   We recommend adding this modifier to all external function of the contract.

2. Returning of the bool value in the `calculateUserReserveCollateralETHInvoicePool` function is redundant. Its value is always true.
3. Passing of the `_interestRateMode` to the `calculateUserReserveCollateralETHInvoicePool` function is redundant. Its value should always be equal to the stable interest mode value.
4. A `RewardDenied` event of the RewardPool is never used.
5. The `exclusiveAddress` field of the RewardPool is never used.

Lowest / Code style / Best Practice

1. The `calculateUserInvoiceCollateralETH` function of the `LendingPoolDataProvider` has commented out code.

2. Multiple code style issues were found by static code analyzers.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality was presented in As-Is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found **1** critical, **2** high, **5** medium, **5** low, and **2** informational issue during the audit.

Notice: some contracts in the repository are not in the audit scope. They can be used by or can use contacts from the scope. During the audit we consider out-of-scope contracts as secure but cannot guaranty that they really are. We recommend reviewing those contracts before using the system. Due to the limited scope, we cannot guarantee that the whole system will work properly all together. We recommend performing the full audit and UAT testing at the production environment as it can reveal issues which cannot be reproduced during the audit.

Violations in the following categories were found and addressed to Customer:

Category	Check Item	Comments
Code review	▪ Data Consistency	▪ Double voting is possible ▪ Inconsistent state may occur as a result of manual decimals set up.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.