PolyTrade

Lender Pool

Smart Contract Audit Report

July 04, 2022

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Introduction

1. About PolyTrade

Polytrade is a decentralized trade finance platform that aims to transform receivables financing. It will connect buyers, sellers, insurers, and investors for a seamless receivables financing experience and help users avoid existing market challenges using its platform solutions. Polytrade will provide real-world borrowers access to low interest and swift financing to free up critical working capital tapped from crypto lenders.

By onboarding on Polytrade, everybody gains because the platform bridges the gaps in traditional receivables financing by accessing untapped crypto liquidity. Through Polytrade, we want to boost business growth where liquidity is not a hindrance.

Visit https://polytrade.finance/ to know more about it.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provides professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has been able to secure 175+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-ups with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The PolyTrade team has provided the following doc for the purpose of audit:

1. https://github.com/polytrade-finance/lender-pool/tree/develop/docs
2. https://polytrade.finance/whitepaper.pdf
3. https://polytrade.finance/one-pager.pdf

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: PolyTrade
- Contracts Name: LenderPool, RedeemPool, Reward, RewardManager, Strategy, Token, Verification
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github commits for initial audit: 9dee55b0de97b6e4cb385f36fc8a14b1668072d1
- Github commits for final audit: 895ddf1527daed28964266fd4d28daecad7266de
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include
   a. Correctness
   b. Readability
   c. Sections of code with high complexity
   d. Quantity and quality of test coverage

Security Level Reference

Every issue in this report were assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues(v1)	High	Medium	Low
Open	-	-	-
Closed	1	2	6

Issues(v2)	High	Medium	Low
Open	-	-	-
Closed	-	2	1

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommen
ded.

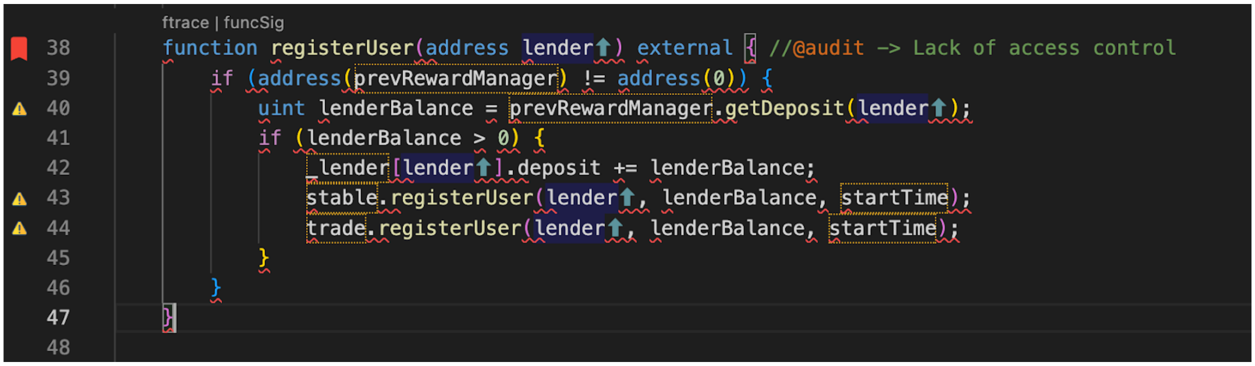
Findings(v1)

High Severity Issues

1. **Absence of adequate access control in the registerUser() function**
   Line no: 38-47(Contract: RewardManager)

   Explanation:
   The registerUser() function in the contract lacks the onlyRole(LENDER_POOL) modifier which would have ensured that this function shall only be accessible by the LenderPool.

   Unlike the current structure of the RewardManager contract where every imperative state-changing function has been assigned an onlyRole() access control, no such modifier was found for the registerUser() function.



```
       ftrace | funcSig
38     function registerUser(address lender) external { //@audit -> Lack of access control
39         if (address(prevRewardManager) != address(0)) {
40             uint lenderBalance = prevRewardManager.getDeposit(lender);
41             if (lenderBalance > 0) {
42                 _lender[lender].deposit += lenderBalance;
43                 stable.registerUser(lender, lenderBalance, startTime);
44                 trade.registerUser(lender, lenderBalance, startTime);
45             }
46         }
47     }
48
```

This leads to an undesired scenario where the function shall be accessible by any third-party actor who can trigger the function.

Recommendation:
If the above-mentioned scenario is not an intentional design, it is recommended to attach relevant access control to functions and update the functions accordingly.

Since the registerUser() function of RewardManager is being called by the LenderPool, it's a better and more secure design only to allow LenderPool the right to access this function.

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

Medium Severity Issues

1. **startTime not validated before registering users**
   Line no: 38-47(RewardManager)

   Explanation:
   While registering a new user using the registerUser() function, the contract also calls the registerUser() function of the Reward contract and passes crucial arguments like lender's address, lender's balance and the start time.

   However, it was found that no adequate validations are done within the function body to ensure whether or not the startTime state variable has been initialized yet.

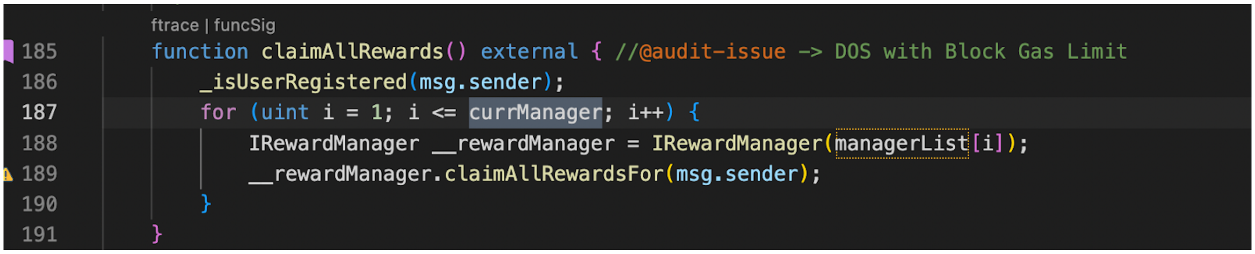
   Due to the lack of this validation, the registerUser() function can be triggered even if the startTime state variable is zero. This leads to a scenario where the zero value for startTime could be passed while registering a user to the Reward contract(Line 43, 44) and the startTime for a specific lender will be made zero.

   Recommendation:
   Since startTime state variable plays a significant role in the reward calculation procedures in the contract, its recommended to include required input validations to ensure only valid arguments are passed to functions.

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

2. **DoS due to Block Gas Limit. Pull over Push Payments could be preferred.**
   Line no: 187, 200(Contract: LenderPool)

   Explanation:
   As per the current function design of the claim functions in the LenderPool contract, there is a for loop iteration over the total value of currManager, and rewards are distributed to the lender for each reward manager address in the managerList array.



```
ftrace | funcSig
function claimAllRewards() external { //@audit-issue -> DOS with Block Gas Limit
    _isUserRegistered(msg.sender);
    for (uint i = 1; i <= currManager; i++) {
        IRewardManager __rewardManager = IRewardManager(managerList[i]);
        __rewardManager.claimAllRewardsFor(msg.sender);
    }
}
```

While such function design shall work fine for smaller iterations, as the number of currManager state variable increases the chances of DoS vector due to the block gas limit increases as well. Since each block has an upper bound on the amount of gas that can be spent, the transaction will likely fail if it exceeds that upper bound.

Recommendation:
In order to mitigate the chances of such a scenario the function design for reward distribution could be improved. An alternate and better way for payments could be the Pull over Push mechanism.

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

Low Severity Issues

1. **No Events emitted after imperative State Variable modification**
   Line no -53(Contract: RewardManager)

Explanation:
Functions that update an imperative arithmetic state variable contract should emit an event after the update.
The registerRewardManager() function in the contract updates a crucial state variable, i.e., startTime but doesn't emit any event on its modification.

The absence of event emission for important state variables update also makes it difficult to track them off-chain as well.

Recommendation:
As per the best practices in smart contract development, an event should be fired after changing crucial arithmetic state variables.

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

2. **rewardOf() function provides misleading values in case of failure**
   Line no: 130-142(Contract: RewardManager)

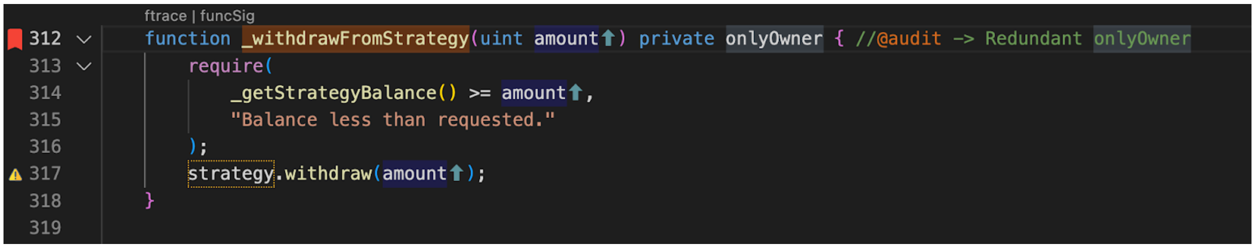
   Explanation:
   The rewardOf() function is designed to return the reward value for a specific lender and token. However, as per the current function design, if the token address passed as an argument doesn't match the reward token address of both stable and trade, it returns zero.
   This could be misleading as it represents the fact that the total reward for the lender is zero instead of symbolizing the wrong token address passed as an argument.

   Recommendation:
   While this function works fine when called via LenderPool contract, it will lead to misleading return values when called directly from the RewardManager contract. Hence, adequate error messages could be provided for this function.

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

3. **Redundant modifier found with _withdrawFromStrategy() function**
   Line no: 312(Contract: LenderPool)

   Explanation:
   During the manual code review, it was found that an onlyOwner modifier has been attached with the _withdrawFromStrategy() private function.



```
        ftrace | funcSig
312     function _withdrawFromStrategy(uint amount) private onlyOwner { //@audit -> Redundant onlyOwner
313         require(
314             _getStrategyBalance() >= amount,
315             "Balance less than requested."
316         );
317         strategy.withdraw(amount);
318     }
319
```

   This function is called twice within the Lender pool contract by switchStrategy & fillRedeemPool function and both of these function already include the onlyOwner modifier.

   Recommendation:
   In order to avoid redundant use of modifiers and reduce the gas consumption in during function execution, the onlyOwner modifer from the _withdrawFromStrategy() function can be removed.

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

4. **Storage Reads could be avoided to save gas**
   Line no: 151(Contract: LenderPool)
   The withdrawAllDepost() function includes unnecessary storage reads which could be avoided.

   In line 151, the deposited balance of the lender is read from storage (mapping) while the local variable called balance is already storing the same information.

   Recommendation:
   Unnecessary reading from storage increases the use of gas. The function could be designed to reduce gas consumption.

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

5. **Absence of input validations found**
   Line no: 23, 55(Contract: Reward)

   Explanation:
   As per the current design of the contract, the setReward function doesn't validate the newReward argument being passed to the function.

   Although the function has been marked as only accessible by the owner, it allows passing zero values for the reward as well which could wipe out the rewards for a particular round.

   Additionally, it was also found that the constructor doesn't include zero address validations.

   Recommendation:
   Input validations should be included in functions to ensure only valid arguments are passed.

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

6. **No Events emitted after imperative State Variable modification**
   Line no: 101-114, 126-136(Contract: Reward)

   Explanation:
   Functions that update an imperative arithmetic state variable contract should emit an event after the update.
   The following functions in the contract update state variables but doesn't emit any event on its modification:
   - deposit()
   - withdraw()

   The absence of event emission for important state variables update also makes it difficult to track them off-chain as well.

   Recommendation:
   As per the best practices in smart contract development, an event should be fired after changing crucial arithmetic state variables.

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

Recommendation / Informational

1. **require statements should be used instead of IF-ELSE statements**
   Line no: 39, 41(Contract: RewardManager)

   Explanation:
   registerUser() function includes some strict validation in order to execute the function. For instance,
   - The previous reward manager contract should not be a zero address
   - Lender's balance should be greater than zero.

   These are strict requirements without which the function should never execute. In solidity, it is considered a better practice to use require statements for such strict validations instead of IF-ELSE statements.

   Recommendation:
   Require statements should be preferred.

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

2. **require statements should be used instead of IF-ELSE statements**
   Line no: 40(Contract: Reward)

   Explanation:
   registerUser() function includes some strict validation in order to execute the function. For instance,

   - The lender address being passed as argument must not already be registered..

   This is a strict requirement without which the function should never execute. In solidity, it is considered a better practice to use require statements for such strict validations instead of IF-ELSE statements.

   Recommendation:
   Require statements should be preferred.

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

Findings(v2)

High Severity Issues

No issues were found.

Medium Severity Issues

1. **Hardcoded address**
   Contract: Strategy.sol

   Description: The address for AAVE is hard coded in the strategy contract.

Line	Code/Function
20	IAaveLendingPool public constant AAVE = IAaveLendingPool(0x8dFf5E27EA6b7AC08EbFdf9eB090F32ee9a30fcf);

   Recommendation:
   It is recommended to not use hardcoded address, set it using constructor and is possible add a setter for the same. Since the third party dependencies can change overtime and also it makes the contract chain depended.

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

2. **Edge case in _isUserRegistered**
   Contract: LenderPool.sol

   Description: If there is only one *rewardManager* then the require check will pass without checking whether the user is registered with that rewardManager or not. This happens because the check before the "or" will be true.

Line	Code/Function
342	require(managerList[managerToIndex[address(rewardManager)] - 1] ==

	address(0) || (_lender[_user].isRegistered[managerList[managerToIndex[address(rewardManager)] - 1]] && _lender[_user].isRegistered[address(rewardManager)]));

Recommendation:

Code/Function
require((managerList[managerToIndex[address(rewardManager)] - 1] == address(0) || _lender[_user].isRegistered[managerList[managerToIndex[address(rewardManager)] - 1]]) && _lender[_user].isRegistered[address(rewardManager)]);

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

Low Severity Issues

1. **Missing approval to new treasury**
   Contract: LenderPool.sol

   Description: The method *switchTreasury* removes all approval from old treasury but doesnt grant any to the new treasury.

Line	Code/Function
11	function switchTreasury(address newTreasury) external

Recommendation:
Grant same amount of approval to the new treasury as well.

Acknowledged (July 04th, 2022): The issue has been acknowledged by the team in commit 895ddf1527daed28964266fd4d28daecad7266de.

Recommendation / Informational

1. **Unused Variables**
   Contract: RedeemPool.sol

   Description:
   These contract defines the given state variables but never uses it.

Line	Code/Function
19	mapping(address => bool) public lenderPool;

   Recommendation:
   Remove unused variables

2. **Unused Imports**
   Contract: LenderPool.sol

   Description:
   The contract contains imports that are not used within the contract and make the contract heavy.

Line	Code/Function
5	import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

   Recommendation:
   Remove unused imports

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

3. **Redundant mappings**
   Contract: LenderPool.sol, Reward.sol, RewardManager.sol

   Description: The following mapping state variable is defined and maintained in three contracts increasing the chances of inconsistencies and increasing operational gas costs.

   mapping(address => Lender) private _lender;

Recommendation:
Rethink logic to keep state information at a single point of truth

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

4. **Redundant require check**
   Contract: ReedemPool.sol

   Description: the following require checks are also present within the burn and mint functions making these require checks redundant

Line	Code/Function
68	require(tStable.balanceOf(msg.sender) >= amount, "Insufficient balance"); require(tStable.allowance(msg.sender, address(this)) >= amount, "Insufficient allowance");

   Recommendation:
   Remove unnecessary checks to save on gas consumption.

   Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

5. **Redundancy in switchStrategy**
   Contract: LenderPool.sol

   Description: switchStrategy sets strategy address twice if oldStrategy is not zero.

Line	Code/Function
55	function switchStrategy(address newStrategy) external

Recommendation:
Remove redundant assignment to save on gas consumption.

6. **Refactor _isUserRegistered**
   Contract: LenderPool.sol

   Description:
   The method does nothing if the rewardManager is zero address, which is clear on the first line and yet it is checked again in the if condition.

Line	Code/Function
341	function _isUserRegistered(address _user) private

Recommendation:
Refactor the method to optimize gas usage and avoid redundant checks

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

7. **Refactor methods**
   Contract: LenderPool.sol

   Description:
   withdrawDeposit and withdrawAllDeposit share common lines of code which increases the contract size.

   Recommendation:
   withdrawDeposit and withdrawAllDeposit can be refactored to call a common internal function. Similar can be done for redeem and redeemAll.

8. **Pragma Unlocked**
   Contract: All Contracts

   Description:
   Every Solidity file specifies in the header a version number of the format. The caret symbol before the version number implies an unlocked pragma, meaning that the compiler will use the specified version and above. This range of versions might cause some unexpected version-related issues.

Recommendation:
Fix the solidity version by removing the caret symbol from the specified version numbers.

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

9. **Missing events in setValidation**
Contract: Verification.sol

Description:
The method setValidation doesn't emit any event.

Recommendation:
Create and emit events for every setter.

Status: Open

10. **Incorrect event name in netspec comment**
Contract: Verification.sol

Description:
The netspec comment before updateValidationLimit states that it emits an event called NewValidationLimit but it emits ValidationLimitUpdated instead

Recommendation:
Update the comment

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

11. **Contract name in revert messages**

Recommendation:
It is recommended to have all require error messages be preceded with the contract name for better understanding and debugging of reasons

12. Missing validation

Contract: RewardManager.sol

Description:
Some of the best practices are not followed throughout the repo, and some simple recommendations can be implemented.

Line	Code/Function
119	function claimRewardFor(address lender, address token)

Recommendation:
Add an input validation checking validity of the address passed.

Amended (July 04th, 2022): The issue has been fixed by the team and is no longer present in commit 895ddf1527daed28964266fd4d28daecad7266de.

13. Hardcoded values

Contract: Reward.sol, Token.sol

Description:
Hardcoded values are used to initialize an year variable and to mint tokens in the tokens contract

Line	Code/Function
Token - 23	_mint(msg.sender, 1_000_000_000 * (10**decimals_));
Reward - 279	uint oneYear = (10000 * 365 days);

Recommendation:
It is recommended to pass hardcoded values as a parameter and make standard values as constants. Make a constant for oneYear variable in Reward.sol and for Token.sol pass the mint amount as a constructor parameter.

Status: Partially corrected

14. Misleading variable name

Contract: Reward.sol

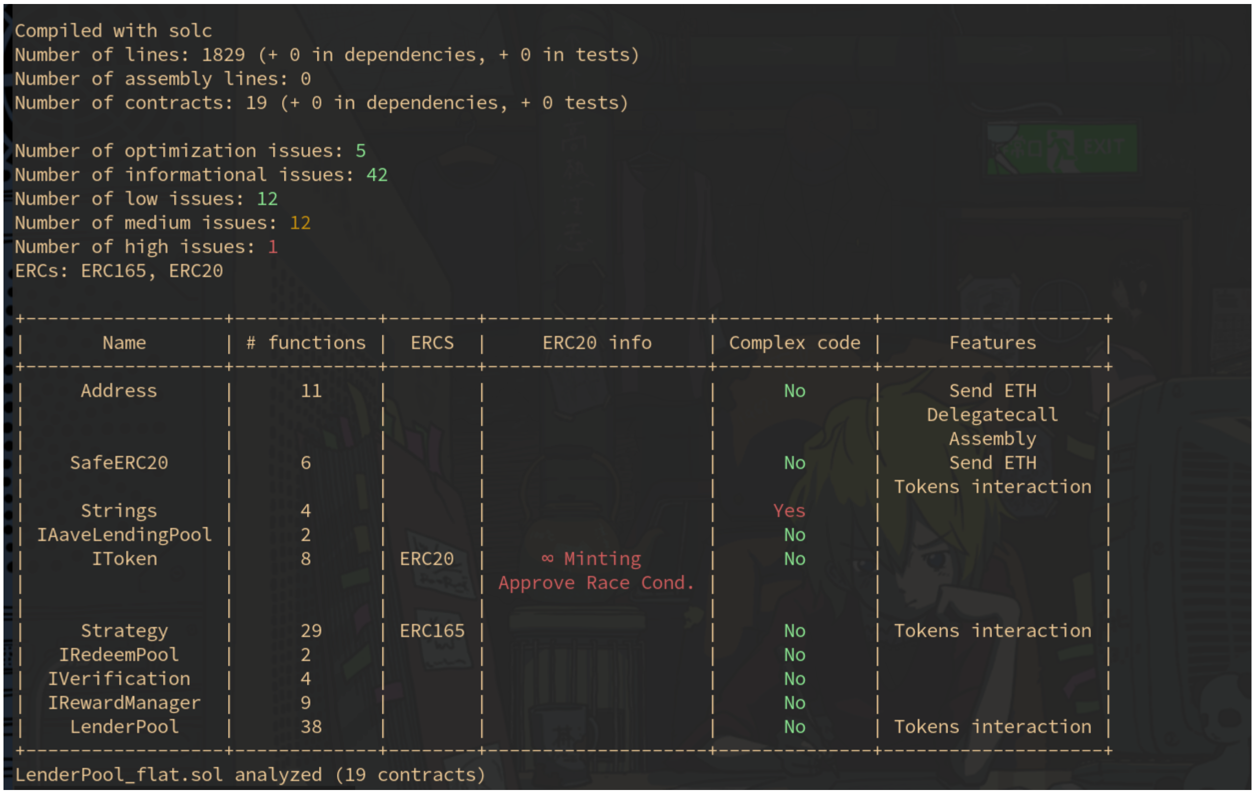
Description:
Reward contract defines pauseReward to reset reward but the name suggests it pauses contract for use

Recommendation:
It is better to name the method in accordance with what it is performing to avoid confusion.

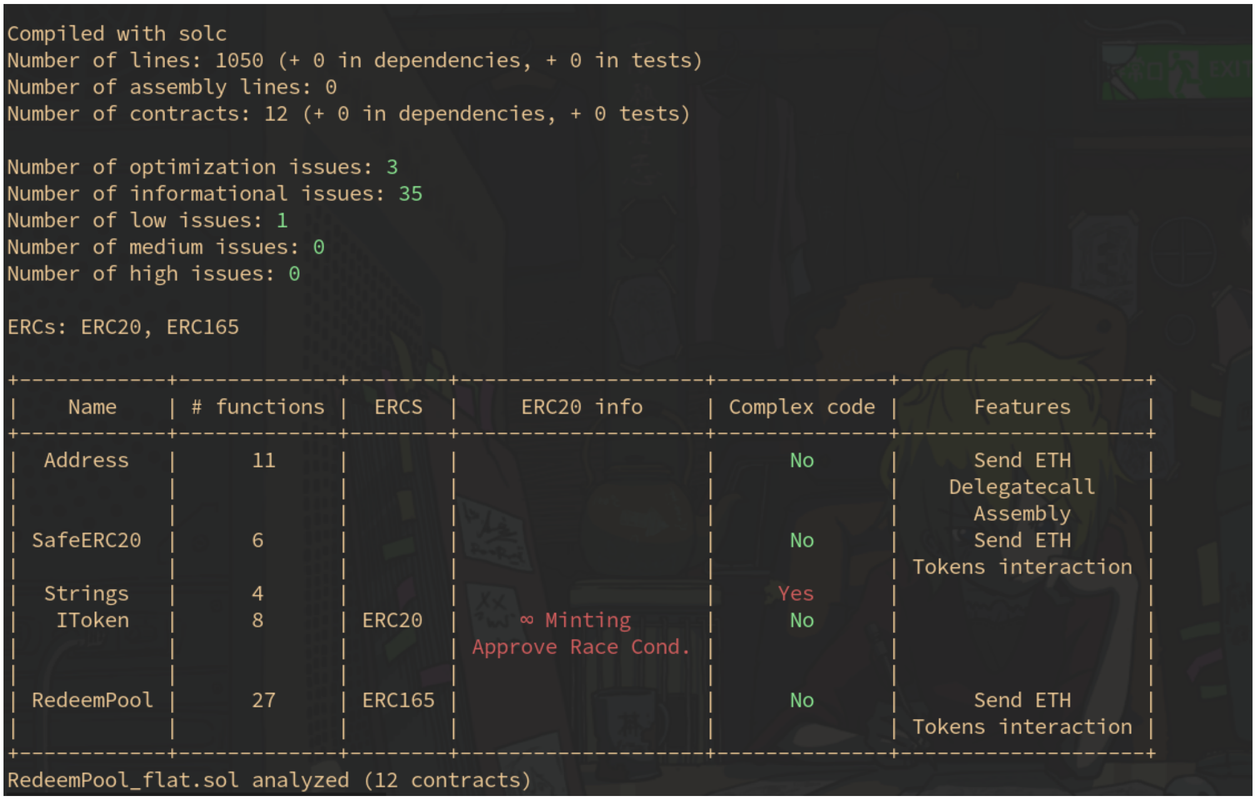
Status: Partially corrected
Remark: Comment not updates

Automated Audit Result

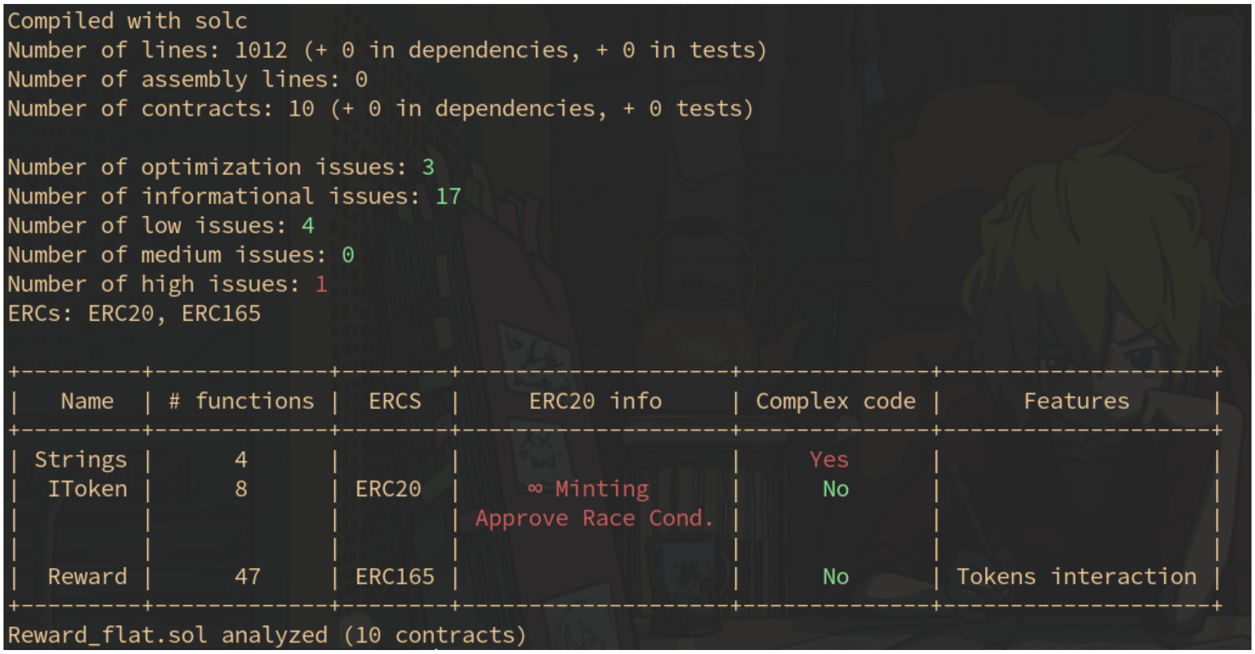
Slither



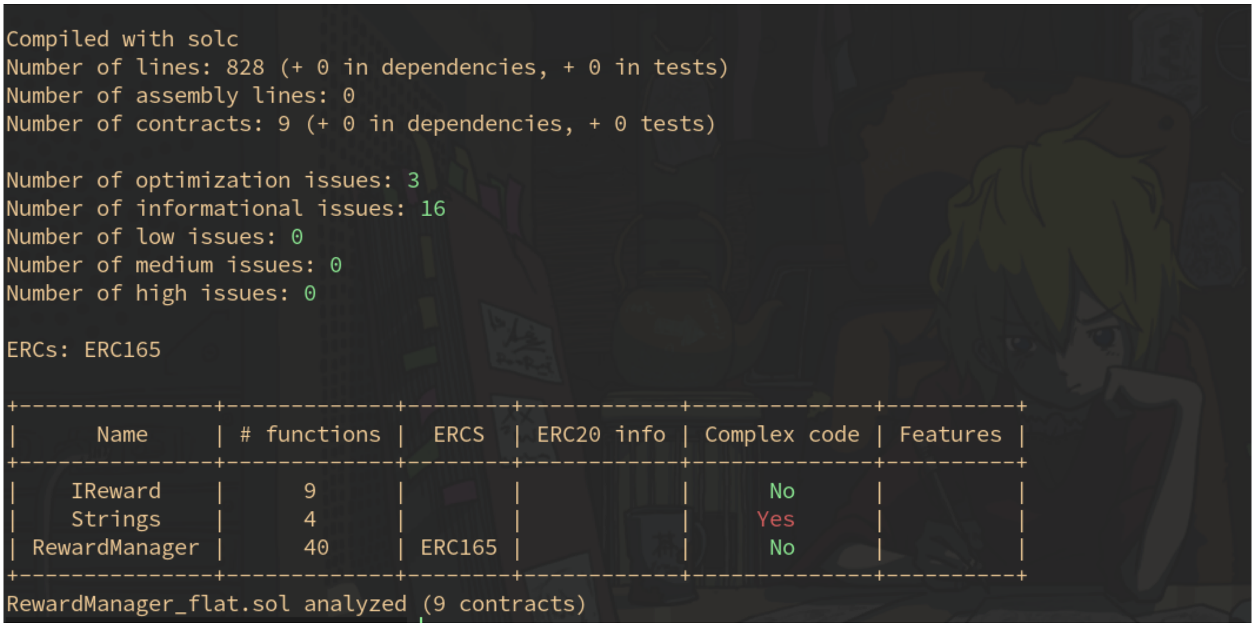
```
Compiled with solc
Number of lines: 1829 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 19 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 5
Number of informational issues: 42
Number of low issues: 12
Number of medium issues: 12
Number of high issues: 1
ERCs: ERC165, ERC20

+------------------+-------------+--------+--------------------+--------------+--------------------+
|       Name       | # functions |  ERCS  |     ERC20 info     | Complex code |      Features      |
+------------------+-------------+--------+--------------------+--------------+--------------------+
|     Address      |      11     |        |                    |      No      |      Send ETH      |
|                  |             |        |                    |              |    Delegatecall    |
|                  |             |        |                    |              |      Assembly      |
|    SafeERC20     |      6      |        |                    |      No      |      Send ETH      |
|                  |             |        |                    |              | Tokens interaction |
|     Strings      |      4      |        |                    |     Yes      |                    |
| IAaveLendingPool |      2      |        |                    |      No      |                    |
|      IToken      |      8      | ERC20  |     ∞ Minting      |      No      |                    |
|                  |             |        | Approve Race Cond. |              |                    |
|                  |             |        |                    |              |                    |
|     Strategy     |      29     | ERC165 |                    |      No      | Tokens interaction |
|    IRedeemPool   |      2      |        |                    |      No      |                    |
|   IVerification  |      4      |        |                    |      No      |                    |
|  IRewardManager  |      9      |        |                    |      No      |                    |
|    LenderPool    |      38     |        |                    |      No      | Tokens interaction |
+------------------+-------------+--------+--------------------+--------------+--------------------+
LenderPool_flat.sol analyzed (19 contracts)
```



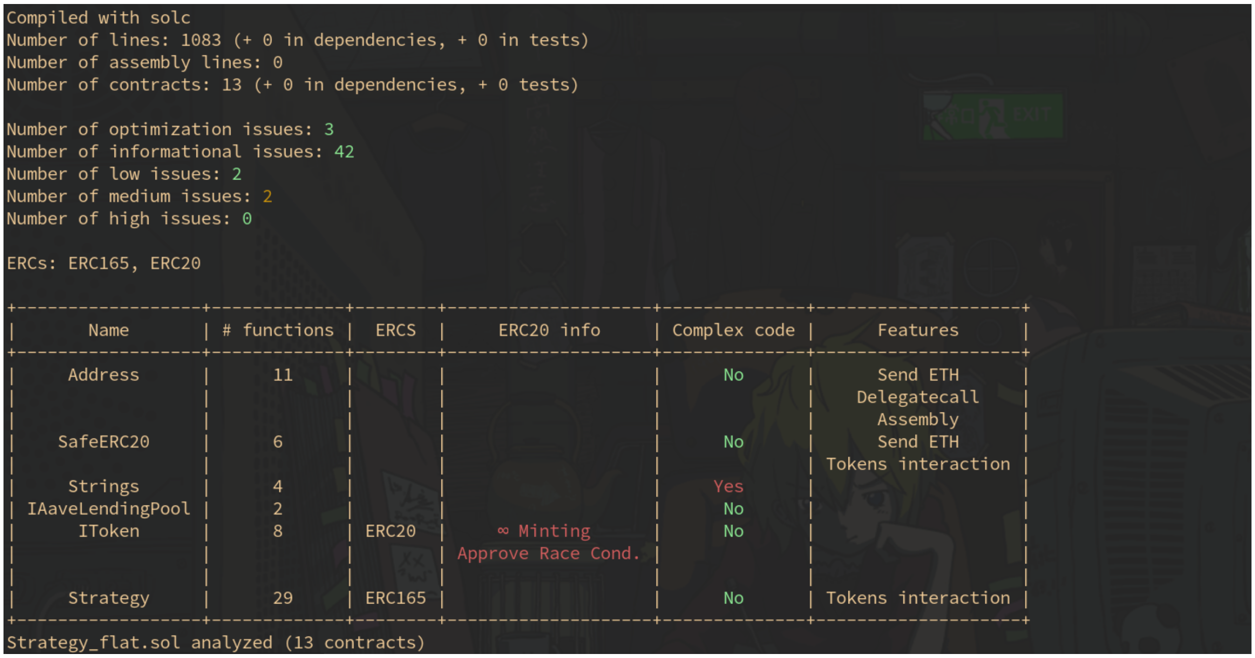
```
Compiled with solc
Number of lines: 1050 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 12 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 35
Number of low issues: 1
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20, ERC165

+------------+-------------+--------+--------------------+--------------+--------------------+
|    Name    | # functions |  ERCS  |     ERC20 info     | Complex code |      Features      |
+------------+-------------+--------+--------------------+--------------+--------------------+
|  Address   |      11     |        |                    |      No      |      Send ETH      |
|            |             |        |                    |              |    Delegatecall    |
|            |             |        |                    |              |      Assembly      |
| SafeERC20  |      6      |        |                    |      No      |      Send ETH      |
|            |             |        |                    |              | Tokens interaction |
|  Strings   |      4      |        |                    |     Yes      |                    |
|   IToken   |      8      | ERC20  |     ∞ Minting      |      No      |                    |
|            |             |        | Approve Race Cond. |              |                    |
|            |             |        |                    |              |                    |
| RedeemPool |      27     | ERC165 |                    |      No      |      Send ETH      |
|            |             |        |                    |              | Tokens interaction |
+------------+-------------+--------+--------------------+--------------+--------------------+
RedeemPool_flat.sol analyzed (12 contracts)
```



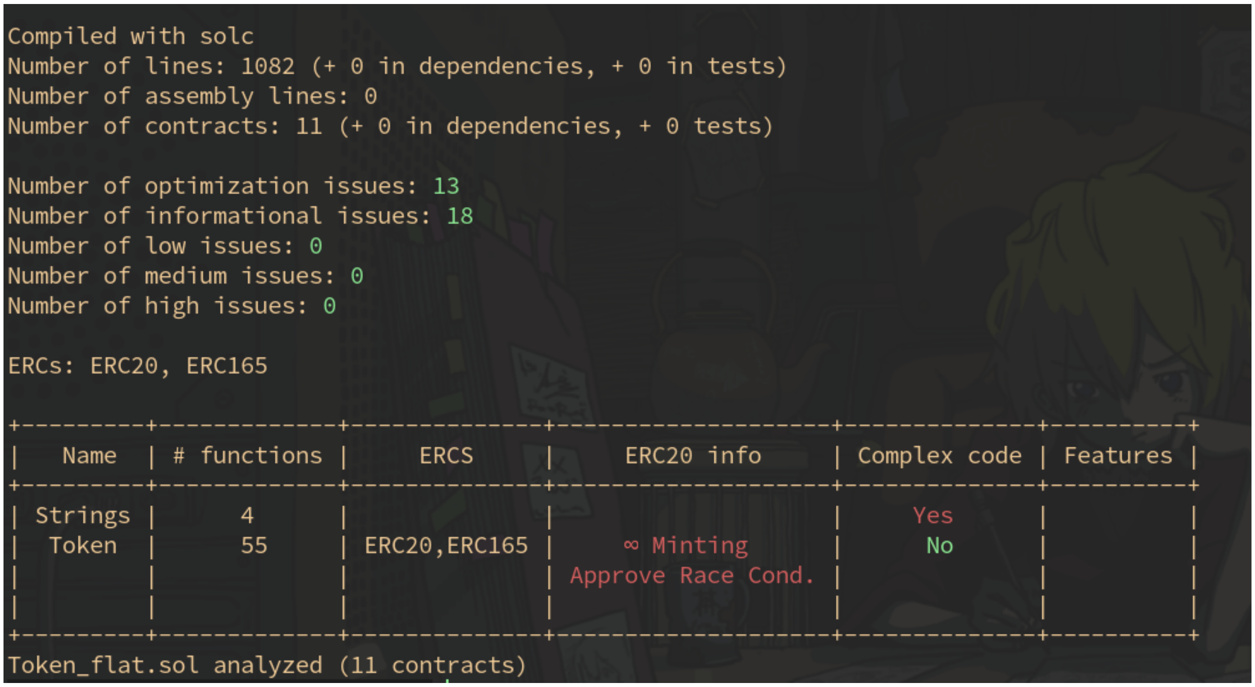
```
Compiled with solc
Number of lines: 1012 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 10 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 17
Number of low issues: 4
Number of medium issues: 0
Number of high issues: 1
ERCs: ERC20, ERC165

+---------+-------------+--------+--------------------+--------------+--------------------+
|   Name  | # functions |  ERCS  |     ERC20 info     | Complex code |      Features      |
+---------+-------------+--------+--------------------+--------------+--------------------+
| Strings |      4      |        |                    |     Yes      |                    |
|  IToken |      8      | ERC20  |     ∞ Minting      |      No      |                    |
|         |             |        | Approve Race Cond. |              |                    |
|         |             |        |                    |              |                    |
|  Reward |      47     | ERC165 |                    |      No      | Tokens interaction |
+---------+-------------+--------+--------------------+--------------+--------------------+
Reward_flat.sol analyzed (10 contracts)
```



```
Compiled with solc
Number of lines: 828 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 9 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 16
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC165

+---------------+-------------+--------+------------+--------------+----------+
|      Name     | # functions |  ERCS  | ERC20 info | Complex code | Features |
+---------------+-------------+--------+------------+--------------+----------+
|    IReward    |      9      |        |            |      No      |          |
|    Strings    |      4      |        |            |     Yes      |          |
| RewardManager |      40     | ERC165 |            |      No      |          |
+---------------+-------------+--------+------------+--------------+----------+
RewardManager_flat.sol analyzed (9 contracts)
```

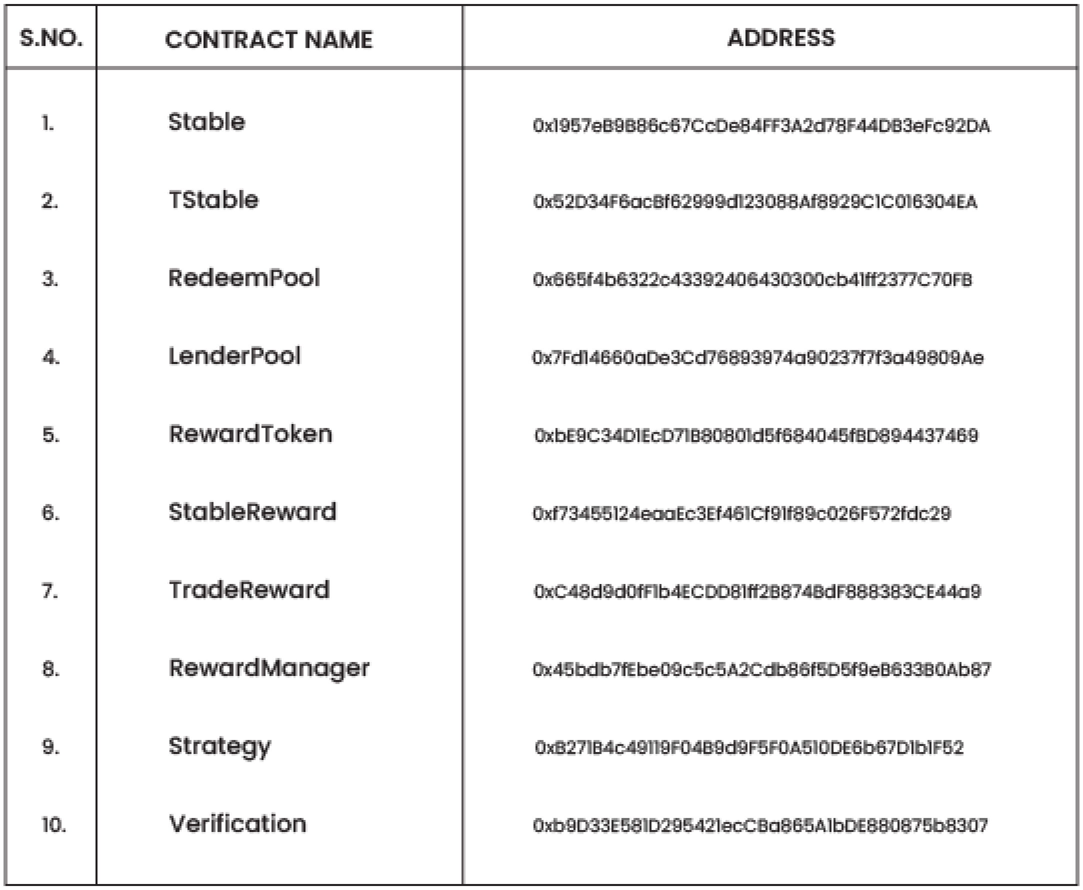


```
Compiled with solc
Number of lines: 1083 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 13 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 42
Number of low issues: 2
Number of medium issues: 2
Number of high issues: 0

ERCs: ERC165, ERC20

+-----------------+-------------+--------+--------------------+--------------+--------------------+
|       Name      | # functions |  ERCS  |     ERC20 info     | Complex code |      Features      |
+-----------------+-------------+--------+--------------------+--------------+--------------------+
|     Address     |      11     |        |                    |      No      |      Send ETH      |
|                 |             |        |                    |              |    Delegatecall    |
|                 |             |        |                    |              |      Assembly      |
|    SafeERC20    |      6      |        |                    |      No      |      Send ETH      |
|                 |             |        |                    |              | Tokens interaction |
|     Strings     |      4      |        |                    |     Yes      |                    |
| IAaveLendingPool |      2      |        |                    |      No      |                    |
|      IToken     |      8      | ERC20  |     ∞ Minting      |      No      |                    |
|                 |             |        | Approve Race Cond. |              |                    |
|                 |             |        |                    |              |                    |
|     Strategy    |      29     | ERC165 |                    |      No      | Tokens interaction |
+-----------------+-------------+--------+--------------------+--------------+--------------------+
Strategy_flat.sol analyzed (13 contracts)
```

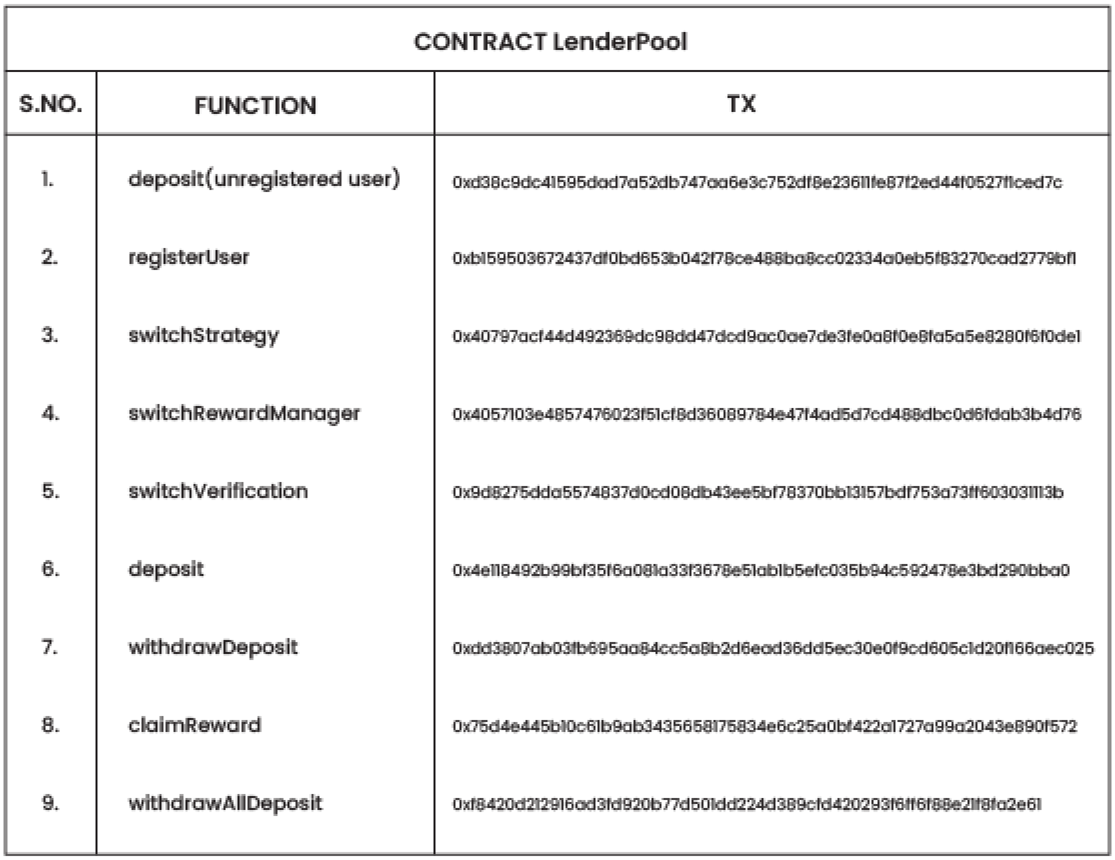


```
Compiled with solc
Number of lines: 1082 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 11 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 13
Number of informational issues: 18
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20, ERC165

+---------+-------------+--------------+--------------------+--------------+----------+
|   Name  | # functions |     ERCS     |     ERC20 info     | Complex code | Features |
+---------+-------------+--------------+--------------------+--------------+----------+
| Strings |      4      |              |                    |     Yes      |          |
|  Token  |      55     | ERC20,ERC165 |     ∞ Minting      |      No      |          |
|         |             |              | Approve Race Cond. |              |          |
|         |             |              |                    |              |          |
+---------+-------------+--------------+--------------------+--------------+----------+
Token_flat.sol analyzed (11 contracts)
```

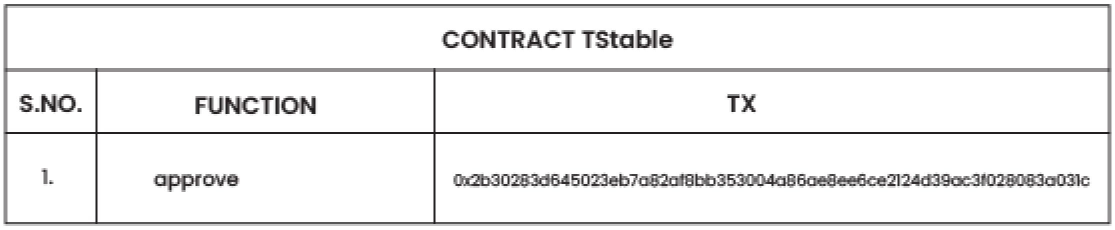
Functional Testing / Tesnet



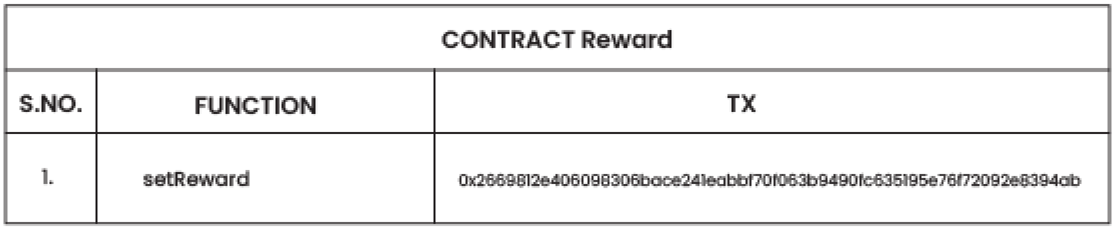
S.NO.	CONTRACT NAME	ADDRESS
1.	Stable	0x1957eB9B86c67CcDe84FF3A2d78F44DB3eFc92DA
2.	TStable	0x52D34F6acBf62999d123088Af8929C1C016304EA
3.	RedeemPool	0x665f4b6322c43392406430300cb41ff2377C70FB
4.	LenderPool	0x7Fd14660aDe3Cd76893974a90237f7f3a49809Ae
5.	RewardToken	0xbE9C34D1EcD71B80801d5f684045fBD894437469
6.	StableReward	0xf73455124eaaEc3Ef461Cf91f89c026F572fdc29
7.	TradeReward	0xC48d9d0fF1b4ECDD81ff2B874BdF888383CE44a9
8.	RewardManager	0x45bdb7fEbe09c5c5A2Cdb86f5D5f9eB633B0Ab87
9.	Strategy	0xB271B4c49119F04B9d9F5F0A510DE6b67D1b1F52
10.	Verification	0xb9D33E581D295421ecCBa865A1bDE880875b8307



CONTRACT LenderPool		
S.NO.	FUNCTION	TX
1.	deposit(unregistered user)	0xd38c9dc41595dad7a52db747aa6e3c752df8e23611fe87f2ed44f0527f1ced7c
2.	registerUser	0xb159503672437df0bd653b042f78ce488ba8cc02334a0eb5f83270cad2779bf1
3.	switchStrategy	0x40797acf44d492369dc98dd47dcd9ac0ae7de3fe0a8f0e8fa5a5e8280f6f0de1
4.	switchRewardManager	0x4057103e4857476023f51cf8d36089784e47f4ad5d7cd488dbc0d6fdab3b4d76
5.	switchVerification	0x9d8275dda5574837d0cd08db43ee5bf78370bb13157bdf753a73ff603031113b
6.	deposit	0x4e118492b99bf35f6a081a33f3678e51ab1b5efc035b94c592478e3bd290bba0
7.	withdrawDeposit	0xdd3807ab03fb695aa84cc5a8b2d6ead36dd5ec30e0f9cd605c1d20f166aec025
8.	claimReward	0x75d4e445b10c61b9ab3435658175834e6c25a0bf422a1727a99a2043e890f572
9.	withdrawAllDeposit	0xf8420d212916ad3fd920b77d501dd224d389cfd420293f6ff6f88e21f8fa2e61

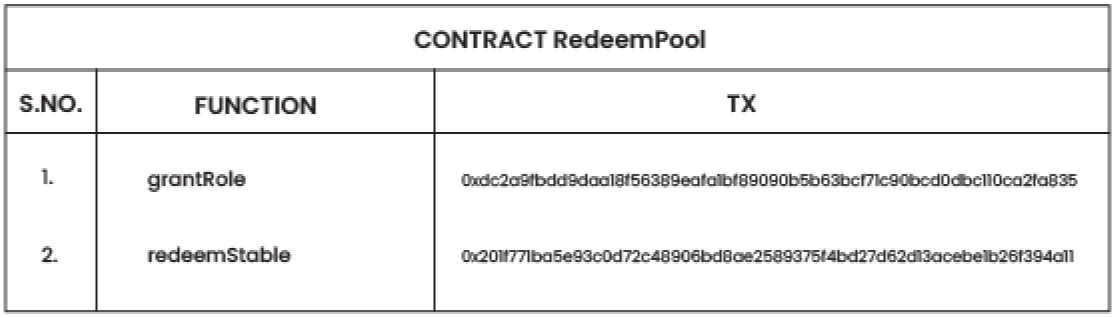


CONTRACT TStable		
S.NO.	FUNCTION	TX
1.	approve	0x2b30283d645023eb7a82af8bb353004a86ae8ee6ce2124d39ac3f028083a031c

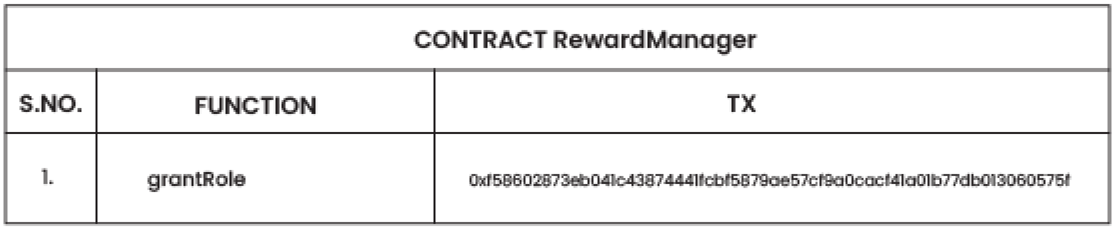


CONTRACT Reward		
S.NO.	FUNCTION	TX
1.	setReward	0x2669812e406098306bace241eabbf70f063b9490fc635195e76f72092e8394ab

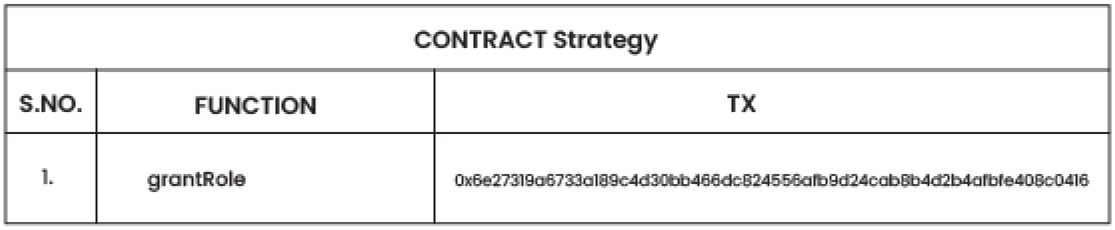
This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommen
ded.



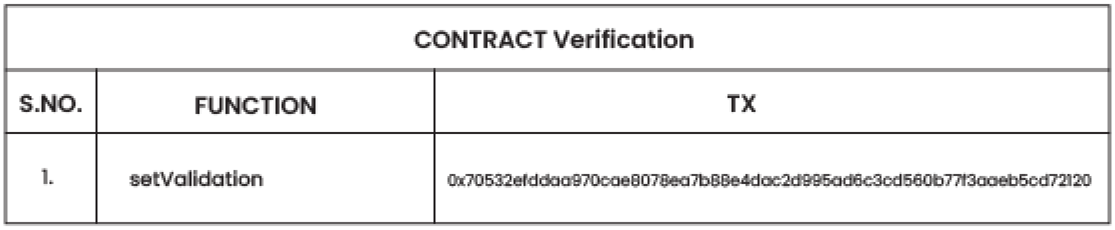
CONTRACT RedeemPool		
S.NO.	FUNCTION	TX
1.	grantRole	0xdc2a9fbdd9daa18f56389eafa1bf89090b5b63bcf71c90bcd0dbc110ca2fa835
2.	redeemStable	0x201f771ba5e93c0d72c48906bd8ae2589375f4bd27d62d13acebe1b26f394a11



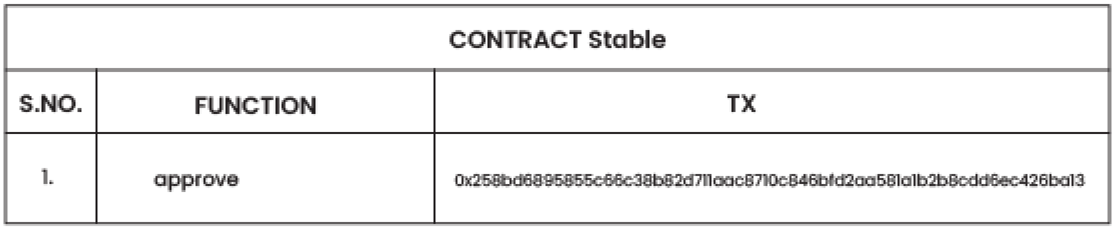
CONTRACT RewardManager		
S.NO.	FUNCTION	TX
1.	grantRole	0xf58602873eb041c43874441fcbf5879ae57cf9a0cacf41a01b77db013060575f



CONTRACT Strategy		
S.NO.	FUNCTION	TX
1.	grantRole	0x6e27319a6733a189c4d30bb466dc824556afb9d24cab8b4d2b4afbfe408c0416



CONTRACT Verification		
S.NO.	FUNCTION	TX
1.	setValidation	0x70532efddaa970cae8078ea7b88e4dac2d995ad6c3cd560b77f3aaeb5cd72120



CONTRACT Stable		
S.NO.	FUNCTION	TX
1.	approve	0x258bd6895855c66c38b82d711aac8710c846bfd2aa581a1b2b8cdd6ec426ba13

Concluding Remarks

While conducting the audits of the PolyTrade Finance smart contract, it was observed that the contracts contain High, Medium, and Low severity issues.

Our auditors suggest that High, Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the PolyTrade Finance platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes