BLOCKHAT
SECURITY

Clixpesa

Smart Contract Security Audit

Prepared by BlockHat

September 19th, 2023 - September 29th, 2023

BlockHat.io

contact@blockhat.io

Document Properties

Client	Clixpesa Solutions Ltd
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/clixpesa/smart-contracts/tree/main	6ce256d3171aba2fc2ee55312c8064596d879962

Re-Audit Files

Repository	Commit Hash
https://github.com/clixpesa/smart-contracts/tree/main	f753f7009a19188e12a78d780446d9cb65f7ab78

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

Clixpesa Solutions Ltd engaged BlockHat to conduct a security assessment on the Clixpesa beginning on September 19th, 2023 and ending September 29th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Clixpesa

- Clixpesa Spaces Clixpesa spaces is basically a savings feature where users can save for personal goals, participate in saving challenges and also save in groups through RoSCAs. With Rotating Savings & Credit Associations (RoSCAs) users can come together as a group to help each other stay financially resilient. Users contribute to a pot, and the target amount goes to one of the users in a particular order until everyone has received a pot and the cycle starts over. This utility commonly known in Kenya as Chamas, helps many raise funds for otherwise big financial goals such as business capital or bills. Within the RoSCAs members can also ask for financial support for financial needs outside of the pot allocations. Users can create a RoSCA easily by inviting their friends through their phone numbers. Once the RoSCAs is created they can select their admins and around can be started. Funds disbursement happens automatically once a pot deadline is reached. Signatories to the RoSCA funds are randomised by the platform in order to give all members equal control over their funds.

- Clixpesa P2P Lending: 68% of loans in the alternative lending market in Africa are P2P loans. With Clixpesa P2P users are able to offer or request loans from each other at their own terms. Clixpesa Finance helps with monitoring the Credit scores of users and only recommending matches to users in order to minimize the risk of default among users. This feature is very useful for those who survive on day loans to run small businesses for purposes such as inventory purchases. This product greatly reduces the cost of loans as it democratises lending and also opens other

earning avenues for users through interest.

Issuer	Clixpesa Solutions Ltd
Website	www.clixpesa.com
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

BlockHat used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

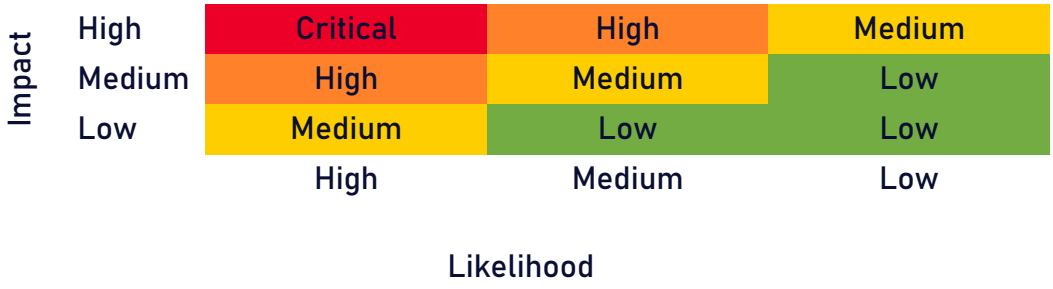
1.2.1 Risk Methodology

Vulnerabilities or bugs identified by BlockHat are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.



2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Clixpesa implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 5 critical-severity, 7 high-severity, 9 medium-severity, 20 low-severity, 2 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
Potential Reentrancy Attack in payoutPot Function	CRITICAL	Fixed
Transfer Amount Might Be Zero	CRITICAL	Fixed
Token Address Update Risk	CRITICAL	Fixed
Potential reentrancy attack	CRITICAL	Fixed
Borrower Loan Repayment Guarantee	CRITICAL	Not Fixed
Pot Funding Exceeding Goal	HIGH	Fixed
Overfunding Vulnerability	HIGH	Fixed
Missing Allowance Check before Token Transfer	HIGH	Fixed
Logic Bug: Non-owners can't fund personal spaces	HIGH	Fixed
Missing Token Address Check	HIGH	Fixed
Missing Allowance Check	HIGH	Fixed
Potential Infinite Loop in _getDayNo and _getOcurranceNo	HIGH	Fixed
Missing Update of isPotted	MEDIUM	Fixed

No Member Limit	MEDIUM	Fixed
Recipient and Caller Overlap	MEDIUM	Fixed
No Removal or Deletion Mechanism	MEDIUM	Fixed
Lack of Input Validation	MEDIUM	Fixed
No way to retrieve ERC20 tokens if sent directly	MEDIUM	Not Fixed
No way to delete personal spaces	MEDIUM	Not Fixed
Potential Revert on Already Withdrawn Personal Space	MEDIUM	Not Fixed
Unsupported Ether Transactions	MEDIUM	Fixed
Floating Pragma	LOW	Fixed
Lack of Input Validation	LOW	Fixed
Potential Scalability Issue with Data Retrieval	LOW	Fixed
No Emergency Shutdown	LOW	Acknowledged
Lack of Input Validation for Array Access	LOW	Fixed
No Rate Limiting	LOW	Fixed
Redundant Data Structures	LOW	Acknowledged
Floating Pragma	LOW	Fixed
Floating Pragma	LOW	Fixed
Redundant Goal Amount Check	LOW	Fixed
Ambiguous Function Return	LOW	Fixed
Lack of Input Length Check	LOW	Fixed
Duplicate storage and checks for space IDs	LOW	Not Fixed
Floating Pragma	LOW	Fixed
Floating Pragma	LOW	Fixed
Unused Variables in Contract	LOW	Not Fixed
Error Message Inconsistency	LOW	Fixed
Lack of Input Validation	LOW	Fixed
Misleading Comments	LOW	Fixed
Use of block.timestamp	LOW	Acknowledged
Reliance on External Contracts	INFORMATIONAL	Fixed
Lack of Access Control	INFORMATIONAL	Acknowledged

3 Finding Details

A Rosca.sol

A.1 Potential Reentrancy Attack in payoutPot Function [CRITICAL]

Description:

The payoutPot function involves an external call to an unknown ERC-20 token contract using the transfer method. Given that the transfer function of an ERC-20 token can be overridden by a malicious token implementation, this opens the door for a potential reentrancy attack. State variables such as RSD.PS and RSD.currentPotBalance are updated after the external call, which can be exploited if the token's transfer method is maliciously implemented to re-enter the payoutPot function.

Code:

Listing 1: Rosca.sol

```
178     function payoutPot() external {
179         require(
180             RSD.RS == RoscaState.isLive,
181             "You can only payout a live Rosca"
182         );
183         require(
184             currentPD.potBalance == RSD.RD.goalAmount,
185             "Pot is not fully funded"
186         );
187         require(
188             RSD.currentPotId == memberIndex[msg.sender],
189             "You are not due to payout"
190         );
191         require(
192             RSD.RD.token.transfer(
```

```
193                 currentPD.potOwner,
194                 currentPD.potBalance.sub(RSD.RD.goalAmount)
195             ),
196             "Transfer failed"
197         );
198         RSD.currentPotBalance = 0;
199         RSD.roscaBalance = RSD.RD.token.balanceOf(address(this));
200         RSD.PS = PotState.isPayedOut;

202         emit PotPayedOut(msg.sender, currentPD.potBalance);
203         _createPot();
204     }
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

Implement the checks-effects-interactions pattern to mitigate reentrancy attacks. Update state variables before calling external contracts. Specifically, move the state-modifying lines above the transfer line to ensure that state is updated before any external interaction.

Status - Fixed

A.2 Transfer Amount Might Be Zero [CRITICAL]

Description:

In the payoutPot function of the Rosca.sol contract, the transfer amount is calculated as the difference between currentPD.potBalance and RSD.RD.goalAmount. Given that a previous require statement ensures that currentPD.potBalance is equal to RSD.RD.goalAmount, the transfer amount will always be zero. This logic suggests that even though the pot is supposed to pay out, no actual funds are being transferred.

Code:

Listing 2: Rosca.sol

```
178     function payoutPot() external {
179         require(
180             RSD.RS == RoscaState.isLive,
181             "You can only payout a live Rosca"
182         );
183         require(
184             currentPD.potBalance == RSD.RD.goalAmount,
185             "Pot is not fully funded"
186         );
187         require(
188             RSD.currentPotId == memberIndex[msg.sender],
189             "You are not due to payout"
190         );
191         require(
192             RSD.RD.token.transfer(
193                 currentPD.potOwner,
194                 currentPD.potBalance.sub(RSD.RD.goalAmount)
195             ),
196             "Transfer failed"
197         );
198         RSD.currentPotBalance = 0;
199         RSD.roscaBalance = RSD.RD.token.balanceOf(address(this));
200         RSD.PS = PotState.isPayedOut;

202         emit PotPayedOut(msg.sender, currentPD.potBalance);
203         _createPot();
204     }
```

Risk Level:

Likelihood – 5
Impact - 4

Recommendation:

Reassess the logic for calculating the transfer amount. If the intention is to transfer the entire currentPD.potBalance, then the subtraction operation is unnecessary. Directly transfer the currentPD.potBalance instead.

Status - Fixed

The smart contract checks if currentPD.potBalance is greater than, equal to, or less than RSD.RD.goalAmount and calculates the dueAmount to be transferred accordingly.

A.3 Pot Funding Exceeding Goal [HIGH]

Description:

The contributeToPot function allows for exact contributions that match the goalAmount to close the pot. There is no mechanism to prevent overfunding or to handle excess funds.

Code:

Listing 3: Rosca.sol

```
171         if (currentPD.potBalance == RSD.RD.goalAmount) {
172             RSD.PS = PotState.isClosed;
173         }
```

Risk Level:

Likelihood – 2
Impact - 5

Recommendation:

Implement logic to prevent contributions if it would cause the pot to exceed the goalAmount. Alternatively, handle refunds for excess contributions.

Status - Fixed

The developer has implemented a mechanism in the contributeToPot function to prevent overfunding and to handle excess funds.

A.4 Overfunding Vulnerability [HIGH]

Description:

In payoutPot function The current validation checks that the currentPD.potBalance is fully funded by ensuring it is equal to RSD.RD.goalAmount. However, this condition does not account for cases where the pot might be overfunded, i.e., if currentPD.potBalance is greater than RSD.RD.goalAmount.

Code:

Listing 4: Rosca.sol

```
183         require(
184             currentPD.potBalance == RSD.RD.goalAmount,
185             "Pot is not fully funded"
186         );
```

Risk Level:

Likelihood – 2
Impact - 5

Recommendation:

Modify the condition to ensure that currentPD.potBalance is not greater than RSD.RD.goalAmount. You could implement an additional require statement to specifically

check for overfunding, or change the current condition to also handle this scenario.

Status - Fixed

The developer handles the case when the pot is overfunded by tracking the excess amount and ensuring only the goal amount is paid out.

A.5 Missing Update of isPotted [MEDIUM]

Description:

When a pot is paid out using payoutPot, there's no logic updating the isPotted status for the member who received the payout.

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Ensure the member's isPotted status is updated to true after they receive a payout.

Status - Fixed

the developer ensures that the isPotted status of the corresponding member is appropriately set to true

A.6 No Member Limit [MEDIUM]

Description:

The comment suggests a maximum of 255 members, but there's no logic in joinRosca enforcing this limit.

Code:

Listing 5: Rosca.sol

```
82      mapping(address => uint256) memberIndex; //maz members 255
```

Risk Level:

Likelihood – 2
Impact - 5

Recommendation:

If there's a need to limit the number of members, add a check in joinRosca to ensure the RSD.members.length doesn't exceed the desired limit.

Status - Fixed

Listing 6: Rosca.sol

```
82  require(RSD.members.length <= 255, "Rosca is full"); //max 255 members
```

A.7 Recipient and Caller Overlap [MEDIUM]

Description:

In the payoutPot function of the Rosca.sol contract, there's a potential confusion arising from the overlap between the caller (msg.sender) and the recipient of the funds (currentPD.potOwner). The function's logic ensures that the caller must be the owner of the current pot by checking if RSD.currentPotId == memberIndex[msg.sender]. However, the actual transfer of funds happens to the address stored in currentPD.potOwner. Even though the requirement ensures they should be the same, this overlap can be confusing for developers or auditors who are reviewing the code.

Code:

Listing 7: Rosca.sol

```
82      function payOutPot() external {
83          uint256 dueAmount;
84          require(RSD.RS == RoscaState.isLive, "!RoscaIsLive");
85          require(currentPD.potBalance >= RSD.RD.goalAmount, "!FullyFunded
                ↪ ");
86          if (currentPD.potBalance > RSD.RD.goalAmount) {
87              //Keep excess in the Rosca
88              dueAmount = RSD.RD.goalAmount;
89              uint256 excessAmount = currentPD.potBalance.sub(RSD.RD.
                    ↪ goalAmount);
90              potExcesses.push(
91                  PotExcess({potId: currentPD.potId, excessAmount:
                        ↪ excessAmount})
92              );
93          } else if (currentPD.potBalance == RSD.RD.goalAmount) {
94              //Transfer all funds to potOwner
95              dueAmount = currentPD.potBalance;
96          }
97          currentPD.potBalance = 0;
98          RSD.currentPotBalance = 0;
99          RSD.PS = PotState.isPayedOut;
100         //change is potted to true
101         RSD.members[memberIndex[currentPD.potOwner].sub(1)].isPotted =
                ↪ true;

103         require(
104             TOKEN.transfer(currentPD.potOwner, dueAmount),
105             "Transfer failed"
106         );
107         RSD.roscaBalance = TOKEN.balanceOf(address(this));
108         delete currentPD.contributions;
```

```
109         emit PotPayedOut(currentPD.potOwner, dueAmount);
110         _createPot();
111     }
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

- Consider simplifying the function by directly using msg.sender as the recipient for the transfer, since the requirement already ensures that the caller must be the owner of the current pot.
- Add clarifying comments in the code to explain the relationship between msg.sender and currentPD.potOwner, stating that they should always be the same at this point in the code.
- Review and ensure other parts of the contract don't introduce inconsistencies between msg.sender and currentPD.potOwner.

Status - Fixed

A.8 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.19. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 8: Rosca.sol

```
8 pragma solidity ^0.8.19;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

A.9 Reliance on External Contracts [INFORMATIONAL]

Description:

The contract relies on an external contract CalcTime for date and time calculations. If there are issues or vulnerabilities in CalcTime, they might impact this contract.

Code:

Listing 9: Rosca.sol

```
11 import "./CalcTime.sol";
```

Risk Level:

Likelihood – 1
Impact - 1

Recommendation:

Ensure that the CalcTime contract is reviewed and tested extensively. Additionally, consider adding the ability to update the address of this contract (with appropriate access controls) in case it ever needs to be upgraded or replaced.

Status - Fixed

B RoscaSpaces.sol

B.1 No Removal or Deletion Mechanism [MEDIUM]

Description:

The contract relies on an external contract CalcTime for date and time calculations. If there are issues or vulnerabilities in CalcTime, they might impact this contract.

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Implement a mechanism to safely end, remove, or archive a RoSCA. Ensure that appropriate access controls are in place, and funds (if any) are safely returned to participants before deletion.

Status - Fixed

B.2 Lack of Input Validation [LOW]

Description:

There's no validation to ensure the provided _roscaAddress is associated with the provided _owner.

Code:

Listing 10: RoscaSpaces.sol

```
47      function getRoscaSpacesByOwner(
48          address _owner
49      ) public view returns (Rosca[] memory) {
50          return myRoscas[_owner];
51      }
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Add checks to validate that the provided _roscaAddress actually belongs to the _owner. This will prevent potential misinformation or errors.

Status - Fixed

B.3 Potential Scalability Issue with Data Retrieval [LOW]

Description:

The getRoscaSpaces function returns the entire roscaSpaces array. As the number of RoSCAs in this array grows, retrieving and processing the entire array may become inefficient. While view functions don't consume gas, the sheer volume of data returned can pose performance issues or even timeouts for dApps or services interfacing with this function.

Code:

Listing 11: RoscaSpaces.sol

```
43      function getRoscaSpaces() public view returns (Rosca[] memory) {
```

```
44          return roscaSpaces;
45      }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

To improve scalability and efficiency, consider implementing pagination or introducing a mechanism to limit the number of returned results. This approach will help to ensure that the function remains performant as the number of RoSCAs increases.

Status - Fixed

B.4 No Emergency Shutdown [LOW]

Description:

The contract doesn't have a way to stop its operations in the event of a detected bug or vulnerability.

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Consider adding a "circuit breaker" or "pause" functionality, controlled by the owner, to halt certain contract functions in emergencies.

Status - Acknowledged

B.5 Lack of Input Validation for Array Access [LOW]

Description:

There's no validation to ensure that the index accessed within myRoscas[_owner] exists. This can lead to out-of-bounds errors.

Code:

Listing 12: RoscaSpaces.sol

```
53      function getRoscaSpaceByOwnernAddress(
54          address _owner,
55          address _roscaAddress
56      ) public view returns (Rosca) {
57          return myRoscas[_owner][myRoscasIdx[_owner][_roscaAddress]];
58      }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Always check that an array index is within bounds before accessing it. Implement checks to ensure the provided index or derived index is valid.

Status - Fixed

B.6 No Rate Limiting [LOW]

Description:

There's no rate limiting on the creation of RoSCAs. This might expose the contract to potential spamming, where a malicious actor could repeatedly create RoSCAs, leading to blockchain bloat.

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider introducing rate-limiting mechanisms or deploying a small fee for creating a RoSCA to prevent spam.

Status - Fixed

The developer add "max 10 roscaSpaces per user" check

B.7 Redundant Data Structures [LOW]

Description:

There seems to be redundancy in how data is stored. While roscaSpacesIndex tracks the index of a specific RoSCA in the roscaSpaces array, myRoscasIdx does a similar job for the user-specific ROSCAs. Redundant data structures increase the complexity and potential for errors and also use more gas for storage and retrieval.

Code:

Listing 13: RoscaSpaces.sol

```
17      mapping(address => uint256) roscaSpacesIndex;
18      mapping(address => Rosca[]) myRoscas;
19      mapping(address => mapping(address => uint256)) myRoscasIdx;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider simplifying the data structure. One approach could be to use a single mapping that points an address directly to its Rosca details, and if user-specific tracking is essential, another mapping that lists RoSCA addresses for a particular user.

Status - Acknowledged

B.8 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.19. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 14: RoscaSpaces.sol

```
8  pragma solidity ^0.8.19;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

B.9 Lack of Access Control [INFORMATIONAL]

Description:

The createRoscaSpace function is public and doesn't have any access controls. This means that any user can call this function and create new RoSCAs without any restrictions.

Code:

Listing 15: RoscaSpaces.sol

```
29      function createRoscaSpace(
30          RoscaDetails memory _RD,
31          string memory _aCode
32      ) public {
33          Rosca newRosca = new Rosca(_RD, _aCode, msg.sender);
34          roscaSpaces.push(newRosca);
35          roscaSpacesIndex[address(newRosca)] = roscaSpaces.length - 1;
36          myRoscas[msg.sender].push(newRosca);
37          myRoscasIdx[msg.sender][address(newRosca)] =
38              myRoscas[msg.sender].length -
39              1;
40          emit RoscaSpaceCreated(address(newRosca), msg.sender, _RD.
                ↪ roscaName);
```

```
41      }
```

Recommendation:

Implement access controls such as a modifier to restrict the creation of RoSCAs only to specific addresses, if that's intended. Alternatively, if the design requires any user to create a RoSCA, then ensure other checks and balances are in place to avoid misuse or spamming.

Status - Acknowledged

C LoansInterest.sol

C.1 Lack of Input Validation [MEDIUM]

Description:

The functions _getInterest and _getNewBalance accept loan amount, rate, and duration as inputs. However, there is no validation for these input values. Incorrect or extreme values can lead to unexpected results.

Code:

Listing 16: LoansInterest.sol

```
17      function _getInterest(
18          uint256 _amount,
19          uint256 _rate,
20          uint256 _duration
21      ) internal pure returns (uint256) {
22          UD60x18 thisAmt = convert(_amount); //AmountInEther*1e18
23          UD60x18 thisRate = convert(_rate);
24          UD60x18 rateAsec = thisRate.div(ud(10000e18)).div(ud(31536000e18)
                ↪ );
25          UD60x18 thisDuration = convert(_duration);
26          UD60x18 thisInterest = (thisAmt.mul(rateAsec)).mul(thisDuration);
```

```
27          uint256 _interest = convert(thisInterest);
28          return _interest;
29      }

31      function _getNewBalance(
32          uint256 _amount,
33          uint256 _rate,
34          uint256 _duration
35      ) internal pure returns (uint256) {
36          uint256 _interest = _getInterest(_amount, _rate, _duration);
37          uint256 _newBalance = _amount.add(_interest);
38          return _newBalance;
39      }
40  }
```

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Introduce validation checks. For instance, ensure that the rate is within reasonable bounds (e.g., between 0 and some maximum possible value). Duration should also be validated to ensure it represents a feasible loan term.

Status - Fixed

C.2 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.19. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not

unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 17: LoansInterest.sol

```
8 pragma solidity ^0.8.19;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

D PersonalSpaces.sol

D.1 Token Address Update Risk [CRITICAL]

Description:

The function allows updating various details of a personal space, including the token address (_SD.token). If the token address is changed after funds have been added, there could be a risk of funds getting stuck or being inaccessible since the withdrawal function might only consider the latest token set.

Code:

Listing 18: PersonalSpaces.sol

```
138     function updatePersonalSpace(SpaceDetails memory _SD) external {
139         require(msg.sender == _SD.owner, "Must be owner");
140         require(personalSpaceIndex[_SD.spaceId] != 0, "SpaceId does not
                ↪ exist");
141         require(
142             myPersonalSpaceIdx[msg.sender][_SD.spaceId] != 0,
143             "SpaceId does not exist"
144         );
145         require(_SD.owner != address(0), "Owner cannot be 0 address");
146         require(_SD.token != IERC20(address(0)), "Token cannot be 0
                ↪ address");
```

Risk Level:

Likelihood – 5
Impact - 3

Recommendation:

Avoid allowing the update of the token address once it has been set or ensure there's a mechanism to track and withdraw funds from all tokens ever associated with a personal space.

Status - Fixed

D.2 Potential reentrancy attack [CRITICAL]

Description:

The function _withdrawFromPersonalSpace seems to have a sequence where tokens are transferred out (_PD.SD.token.transfer(msg.sender, _amount)) before the contract's internal balance is updated. This order of operations can potentially expose the contract to a reentrancy attack if the token contract's transfer function isn't implemented safely.

Code:

Listing 19: PersonalSpaces.sol

```
247 require(_PD.SD.token.transfer(msg.sender, _amount), "Transfer failed");
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

To mitigate the risk of reentrancy, you should follow the "Checks-Effects-Interactions" pattern. First, make all the necessary checks (e.g., validating inputs or verifying conditions). Second, change the state. Finally, interact with external contracts. In this specific scenario, you should adjust the contract such that it decreases the balance (or makes the necessary state changes) before proceeding with the transfer of tokens.

Status - Fixed

D.3 Missing Allowance Check before Token Transfer [HIGH]

Description:

The line _PD.SD.token.transferFrom(msg.sender, address(this), _amount) attempts to transfer tokens from the sender's address to the contract's address. However, there's no preceding check to ensure that the contract has the necessary allowance to transfer the specified _amount of tokens from the sender's address.

Code:

Listing 20: PersonalSpaces.sol

```
182         require(
183             _PD.SS.currentFundState == FundsState.isFundable,
184             "Funding is not allowed / Fully funded"
```

```
185         );
186         require(
187             _PD.SS.currentActivityState == ActivityState.isActive,
188             "Space should be active"
189         );
190         require(
191             _PD.SD.deadline > block.timestamp,
192             "Deadline must be in future"
193         );
194         require(_PD.SD.goalAmount > 0, "Goal must be greater than 0"); //
                ↪ @audit-issue This is already created in personal space. I
                ↪ guess you want to check if the goal amount is exceeded or
                ↪ not. If you want to stop when the goal is acheived if not
                ↪ the if statement in the bottom does the job
195         require(_amount > 0, "Amount must be greater than 0");
196         require(
197             _PD.SD.token.transferFrom(msg.sender, address(this), _amount)
                    ↪ ,
198             "Transfer failed"
199         );
```

Risk Level:

Likelihood – 4

Impact - 4

Recommendation:

Before attempting to transfer tokens, add a check to ensure the contract has been granted the necessary allowance

Status - Fixed

D.4 Logic Bug: Non-owners can't fund personal spaces [HIGH]

Description:

In the fundPersonalSpace function, the contract checks if the personal space ID exists for both the contract-wide list (personalSpaceIndex) and the user-specific list (myPersonalSpaceIdx[msg.sender]). This check incorrectly allows only the owner of the space to fund a personal space. Now, this means if Alice created a personal space with _spaceId equal to "1234", then only Alice (the owner of that space) can fund it because the mapping myPersonalSpaceIdx[Alice's address]["1234"] will have a non-zero value.

If Bob tries to fund Alice's personal space with _spaceId "1234", the mapping myPersonalSpaceIdx[Bob's address]["1234"] will be 0 (assuming Bob never created a personal space with the same ID), and the transaction will revert.

If the intention of the project is that funding should only be done by the owner of the personal space, then this issue becomes informational and should be acknowledged by the team.

Code:

Listing 21: PersonalSpaces.sol

```
170     function fundPersonalSpace(
171         string memory _spaceId,
172         uint256 _amount
173     ) external {
174         require(personalSpaceIndex[_spaceId] != 0, "SpaceId does not
                ↪ exist");
175         require(
176             myPersonalSpaceIdx[msg.sender][_spaceId] != 0,
177             "SpaceId does not exist"
178         );
179         PersonalDetails memory _PD = allPersonalSpaces[
```

```
            personalSpaceIndex[_spaceId].sub(1)
        ];
        require(
            _PD.SS.currentFundState == FundsState.isFundable,
            "Funding is not allowed / Fully funded"
        );
        require(
            _PD.SS.currentActivityState == ActivityState.isActive,
            "Space should be active"
        );
        require(
            _PD.SD.deadline > block.timestamp,
            "Deadline must be in future"
        );
        require(_PD.SD.goalAmount > 0, "Goal must be greater than 0");
        require(_amount > 0, "Amount must be greater than 0");
        require(
            _PD.SD.token.transferFrom(msg.sender, address(this), _amount)
                ↪ ,
            "Transfer failed"
        );
        //_PD.token.transferFrom(msg.sender, address(this), _amount);
        allPersonalSpaces[personalSpaceIndex[_spaceId].sub(1)]
            .currentBalance = _PD.currentBalance.add(_amount);
        myPersonalSpaces[msg.sender][
            myPersonalSpaceIdx[msg.sender][_spaceId].sub(1)
        ].currentBalance = _PD.currentBalance.add(_amount);

        emit FundedPersonalSpace(msg.sender, _spaceId, _amount);

        //check if current balance is greater than goal amount and change
            ↪  state
        if (
            allPersonalSpaces[personalSpaceIndex[_spaceId].sub(1)]
```

```
212                 .currentBalance >=
213             allPersonalSpaces[personalSpaceIndex[_spaceId].sub(1)].SD.
                    ↪ goalAmount
214         ) {
215             allPersonalSpaces[personalSpaceIndex[_spaceId].sub(1)]
216                 .SS
217                 .currentFundState = FundsState.isFullyFunded;
218             myPersonalSpaces[msg.sender][
219                 myPersonalSpaceIdx[msg.sender][_spaceId].sub(1)
220             ].SS.currentFundState = FundsState.isFullyFunded;
221         }
222     }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Remove the check myPersonalSpaceIdx[msg.sender][_spaceId] != 0 from the fundPersonalSpace function. Only the existence of the space in personalSpaceIndex should be validated.

Status - Not Fixed

D.5 No way to retrieve ERC20 tokens if sent directly [MEDIUM]

Description:

If someone mistakenly transfers ERC20 tokens directly to the contract's address (rather than using the fundPersonalSpace method), these tokens will be locked with no way to retrieve them.

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Implement a function that allows the contract owner (or a specific privileged role) to transfer any ERC20 token from the contract, in case they are mistakenly sent.

Status - Not Fixed

D.6 No way to delete personal spaces [MEDIUM]

Description:

While there's an event called DeletedPersonalSpace, there's no function that allows a user to permanently delete a personal space, which means the data will stay in the contract forever, even if it's inactive.

Risk Level:

Likelihood – 2
Impact - 3

Recommendation:

Consider adding a function to allow the deletion of personal spaces and free up storage, potentially giving a gas refund.

Status - Not Fixed

D.7 Potential Revert on Already Withdrawn Personal Space [MEDIUM]

Description:

The function closePersonalSpace seems to call _withdrawFromPersonalSpace to withdraw the current balance of the personal space. If the personal space has already been withdrawn, invoking this function might cause a revert.

Code:

Listing 22: PersonalSpaces.sol

```
279     function closePersonalSpace(string memory _spaceId) external {
280         require(personalSpaceIndex[_spaceId] != 0, "SpaceId does not
                ↪ exist");
281         require(
282             myPersonalSpaceIdx[msg.sender][_spaceId] != 0,
283             "SpaceId does not exist"
284         );
285         PersonalDetails memory _PD = allPersonalSpaces[
286             personalSpaceIndex[_spaceId].sub(1)
287         ];
288         require(_PD.SD.owner == msg.sender, "Must be owner");

290         _withdrawFromPersonalSpace(_spaceId, _PD.currentBalance);
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Implement a check to ensure that the personal space hasn't been withdrawn before proceeding. If the personal space's balance is already zero or if its state indicates that it has been withdrawn, the function should gracefully handle the situation without reverting.

Status - Not Fixed

D.8 Redundant Goal Amount Check [LOW]

Description:

The function fundPersonalSpace appears to have a redundant check for the goal amount. The line require(_PD.SD.goalAmount > 0, "Goal must be greater than 0"); checks if the goal amount is greater than zero. However, this same check seems to have been performed when the personal space was initially created, making this check redundant during the funding process.

Code:

Listing 23: PersonalSpaces.sol

```
182         require(
183             _PD.SS.currentFundState == FundsState.isFundable,
184             "Funding is not allowed / Fully funded"
185         );
186         require(
187             _PD.SS.currentActivityState == ActivityState.isActive,
188             "Space should be active"
189         );
190         require(
191             _PD.SD.deadline > block.timestamp,
192             "Deadline must be in future"
193         );
194         require(_PD.SD.goalAmount > 0, "Goal must be greater than 0");
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

If the goal amount is indeed validated during the creation of the personal space, you can safely remove the redundant check from the fundPersonalSpace function to optimize gas usage.

Status - Fixed

D.9 Ambiguous Function Return [LOW]

Description:

The function doesn't explicitly return true if both conditions are not met. Although in Solidity, not having a return value will return the default value (in this case false for a boolean),

Code:

Listing 24: PersonalSpaces.sol

```
127     function doesPersonalSpaceExist(
128         address owner,
129         string memory _spaceId
130     ) external view returns (bool isExistent) {
131         if (personalSpaceIndex[_spaceId] == 0) {
132             return false;
133         }
134         if (myPersonalSpaceIdx[owner][_spaceId] == 0) {
135             return false;
136         }
137     }
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

it's better for readability and assurance to explicitly return true at the end.

Status - Fixed

D.10 Lack of Input Length Check [LOW]

Description:

The contract's createPersonalSpace function accepts a SpaceDetails struct as an argument. Within this struct, fields like spaceName and spaceId might be prone to unnecessarily large inputs. By not enforcing a maximum length on these fields, a malicious actor might submit exceptionally long strings with the intent of consuming more gas than a typical operation would. This could lead to increased transaction costs and could be used as a form of Denial of Service (DoS) attack, making the function prohibitively expensive to call.

Code:

Listing 25: PersonalSpaces.sol

```
65     function createPersonalSpace(SpaceDetails memory _SD) external {
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

It's advisable to implement input validation checks, especially for string lengths. For critical fields like spaceName and spaceId, define a reasonable maximum length and enforce this in the contract logic. This would ensure that the provided inputs won't consume an excessive amount of gas and helps prevent potential DoS vectors where malicious actors attempt to inflate transaction costs.

Status - Fixed

D.11 Duplicate storage and checks for space IDs [LOW]

Description:

The contract uses both allPersonalSpaces and myPersonalSpaces mappings to store details of personal spaces. This redundancy might lead to increased gas costs and can be a source of bugs.

Code:

Listing 26: PersonalSpaces.sol

```
87          allPersonalSpaces.push(_PD);
88          personalSpaceIndex[_SD.spaceId] = allPersonalSpaces.length;
89          myPersonalSpaces[msg.sender].push(_PD);
90          myPersonalSpaceIdx[msg.sender][_SD.spaceId] = myPersonalSpaces[
91              msg.sender
92          ].length;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider refactoring the contract to use a single mapping structure for storing personal spaces. This can improve efficiency and reduce potential errors.

Status - Not Fixed

D.12 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.19. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 27: PersonalSpaces.sol

```
8 pragma solidity ^0.8.19;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

E P2PLoans.sol

E.1 Borrower Loan Repayment Guarantee [CRITICAL]

Description:

The smart contract lacks mechanisms to ensure that borrowers will repay their loans. Without such mechanisms, lenders face a high risk of defaults, making the platform less attractive for lending.

Risk Level:

Likelihood – 5
Impact - 4

Recommendation:

Implement collateral mechanisms where borrowers lock up a certain value (in tokens or other assets) that can be claimed by lenders in case of defaults. Consider using third-party credit scoring or linking loan repayment to real-world identities, although this might raise privacy concerns and provide legal terms and conditions that borrowers must agree to before taking a loan, ensuring they understand their obligations and potential consequences of defaulting.

Status - Not Fixed

E.2 Missing Token Address Check [HIGH]

Description:

There's no check to ensure LD.token is not the zero address.

Code:

Listing 28: P2PLoans.sol

```
107     function createLoanRequest(LoanRequestDetails memory LRD) external {
108         require(LRD.LD.initiator == msg.sender, "MBO");
109         require(LRD.LD.principal > 0, "Principal<0");
110         require(LRD.LD.interest > 0, "Interest<0");
111         require(LRD.LD.minDuration > 0 && LRD.LD.maxDuration > 0, "
                ↪ Duration<0");
112         require(LRD.LD.minDuration <= LRD.LD.maxDuration, "MinD > MaxD");

114         allRequests.push(LRD);
115         requestIndex[LRD.LD.loanId] = allRequests.length;

117         myRequests[msg.sender].push(LRD);
118         myRequestIdx[msg.sender][LRD.LD.loanId] = myRequests[msg.sender].
                ↪ length;

120         emit CreatedLoanRequest(msg.sender, LRD);
121     }
```

Risk Level:

Likelihood – 4

Impact - 4

Recommendation:

Always validate external inputs. Add a check to ensure that LD.token is not the zero address before proceeding.

Status - Fixed

E.3 Missing Allowance Check [HIGH]

Description:

Before transferring tokens using the transferFrom method, there's no check to ensure the contract has the necessary allowance.

Code:

Listing 29: P2PLoans.sol

```
143         require(
144             _thisRequest.LD.token.transferFrom(
145                 msg.sender,
146                 _thisRequest.LD.initiator,
147                 _thisRequest.LD.principal
148             ),
149             "!Transfer"
150         );
```

Listing 30: P2PLoans.sol

```
192         require(
193             allP2PLoans[p2pLoanIndex[_loanId].sub(1)].LD.token.
                ↪ transferFrom(
194                 msg.sender,
195                 allP2PLoans[p2pLoanIndex[_loanId].sub(1)].LP.lender,
196                 _amount
197             ),
198             "!Transfer"
199         );
```

Listing 31: P2PLoans.sol

```
313         require(
```

```
314             allP2PLoans[p2pLoanIndex[_loanId].sub(1)].LD.token.
                    ↪ transferFrom(
315                 msg.sender,
316                 allP2PLoans[p2pLoanIndex[_loanId].sub(1)].LP.borrower,
317                 allP2PLoans[p2pLoanIndex[_loanId].sub(1)].LD.principal
318             ),
319             "!Transfer"
320         );
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

Before attempting a token transfer, always check the allowance set for the contract. If the allowance is insufficient, revert the transaction with a clear error message.

Status - Fixed

E.4 Unsupported Ether Transactions [MEDIUM]

Description:

Certain functions are marked as payable but don't support ether transactions.

Code:

Listing 32: P2PLoans.sol

```
126     function fundLoanRequest(
127         string memory _requestId,
128         string memory _loanId
129     ) external payable {
```

Listing 33: P2PLoans.sol

```
170     function repayLoan(
171         uint256 _amount
172     ) external payable {
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

If the function isn't intended to accept ether, remove the payable modifier. Alternatively, add logic to handle ether payments if they're supported.

Status - Fixed

E.5 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.19. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 34: P2PLoans.sol

```
8  pragma solidity ^0.8.19;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

E.6 Unused Variables in Contract [LOW]

Description:

The variables isPrivate and bCreditScore in the LoanDetails struct are not used throughout the contract.

Code:

Listing 35: P2PLoans.sol

```
34          uint256 bCreditScore;
35          bool isPrivate;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

If these variables have no purpose, consider removing them to simplify the contract and reduce gas costs. If they're intended for future use, make sure to document their purpose and ensure their correct implementation.

Status - Not Fixed

E.7 Error Message Inconsistency [LOW]

Description:

The error message for zero values is inconsistent, mentioning both "Principal=0 and Interest = 0".

Code:

Listing 36: P2PLoans.sol

```
109         require(LRD.LD.principal > 0, "Principal<0");
110         require(LRD.LD.interest > 0, "Interest<0");
```

Risk Level:

Likelihood – 2
Impact - 2

Recommendation:

Clarify and correct the error message to accurately reflect the error condition.

Status - Fixed

F CalcTime.sol

F.1 Potential Infinite Loop in _getDayNo and _getOcurranceNo [HIGH]

Description:

If a user provides a string that doesn't match any of the strings in weekList or ocurranceList, the function will run to completion without returning a value, reverting due to the lack of a

return statement.

Code:

Listing 37: CalcTime.sol

```
71          bytes32 encodedElement = keccak256(abi.encode(_day));
72          for (uint256 i = 0; i < weekList.length; i++) {
73              if (encodedElement == keccak256(abi.encode(weekList[i]))) {
74                  return i + 1;
75              }
76          }
```

Listing 38: CalcTime.sol

```
87      for (uint256 i = 0; i < ocurranceList.length; i++) {
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

Add a default return statement or revert at the end of these functions, providing a message indicating that the input string is not recognized.

Status - Fixed

F.2 Lack of Input Validation [LOW]

Description:

While there's a requirement check for the year to be greater than or equal to 1970, there's no validation for month (should be 1-12) or day (depending on month and leap years, should be 1-28/29/30/31).

Code:

Listing 39: CalcTime.sol

```
94  function _daysFromDate(
95          uint year,
96          uint month,
97          uint day
98      ) internal pure returns (uint _days) {
99          require(year >= 1970);
100         int _year = int(year);
101         int _month = int(month);
102         int _day = int(day);

104         int __days = _day -
105             32075 +
106             (1461 * (_year + 4800 + (_month - 14) / 12)) /
107             4 +
108             (367 * (_month - 2 - ((_month - 14) / 12) * 12)) /
109             12 -
110             (3 * ((_year + 4900 + (_month - 14) / 12) / 100)) /
111             4 -
112             OFFSET19700101;

114         _days = uint(__days);
115     }
```

Risk Level:

Likelihood – 2

Impact - 2

Recommendation:

Implement thorough validation checks for all function inputs where relevant to ensure the data's accuracy and prevent potential errors or unexpected behaviors.

Status - Fixed

F.3 Misleading Comments [LOW]

Description:

The comment suggests that "Monthly" corresponds to 30, but in the code, "Monthly" corresponds to 28. This can mislead developers who rely on comments for clarity.

Code:

Listing 40: CalcTime.sol

```
80      /// @param _ocurrance 1. Daily 7. Weekly 30 Monthly
```

Risk Level:

Likelihood – 1
Impact - 1

Recommendation:

Ensure that comments are consistent with the actual code logic. Correct the misleading comment to reflect the accurate mapping of strings to values.

Status - Fixed

F.4 Use of block.timestamp [LOW]

Description:

The library makes use of the block.timestamp for time calculations. While block.timestamp is generally reliable, miners can manipulate it to a certain degree (usually within a 15-minute window). In scenarios where precision or certain time guarantees are important, relying on block.timestamp can introduce vulnerabilities.

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Add a default return statement or revert at the end of these functions, providing a message indicating that the input string is not recognized.

Status - Acknowledged

4 Best Practices

BP.1 Misleading Transfer Message

Description:

In the payoutPot function of the Rosca.sol contract, the error message associated with the token transfer operation is generic and non-descriptive. The message "Transfer failed" does not provide specific details or context as to why the transfer might have failed, potentially causing confusion for developers or users interacting with the contract.

Code:

Listing 41: Rosca.sol.sol

```
157         require(
158             RSD.RD.token.transferFrom(msg.sender, address(this), _amount)
                ↪ ,
159             "Transfer failed"
160         );
```

Recommendation:

Refine the error message to provide more context or specificity about the potential reasons for failure. For example, "Token transfer to pot owner failed." This would make debugging and understanding the contract's behavior easier for developers and users.

BP.2 Contract Upgradability

Description:

The contract does not appear to support upgradability. If bugs are found in the contract after deployment, or if new features need to be added, the contract cannot be upgraded without deploying a new one and migrating state.

Recommendation:

If upgradability is a concern, consider implementing an upgradable smart contract pattern. However, be cautious as upgradability introduces its own set of complexities and potential vulnerabilities. Using standardized libraries, such as OpenZeppelin's upgradable contracts, can help address common pitfalls.

BP.3 Limited Documentation : LoansInteresr

Description:

The functions in the library are not well-documented. There's no clarity on what specific parameters like _rate represent (is it an annual rate, monthly rate, etc.?).

Recommendation:

Include comprehensive documentation for each function, clearly outlining its purpose, expected inputs, and any assumptions or specific behaviors. This makes the code easier to maintain and understand for any future developers or auditors.

5 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
INFO:Detectors:
CalcTime._nextDayAndTime(string,string) (CalcTime.sol#20-55) uses a weak
   ↪  PRNG: "nextDay_scope_0 = day + ((28 + _day - dayOfWeek) % 28) (
   ↪ CalcTime.sol#45)"
CalcTime._nextDayAndTime(string,string) (CalcTime.sol#20-55) uses a weak
   ↪  PRNG: "dayOfWeek = ((_days + 3) % 7) + 1 (CalcTime.sol#33)"
CalcTime._nextDayAndTime(string,string) (CalcTime.sol#20-55) uses a weak
   ↪  PRNG: "nextDay = day + ((7 + _day - dayOfWeek) % 7) (CalcTime.
   ↪ sol#36)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #weak-PRNG
INFO:Detectors:
CalcTime._daysFromDate(uint256,uint256,uint256) (CalcTime.sol#94-115)
   ↪ performs a multiplication on the result of a division:
       - __days = _day - 32075 + (1461 * (_year + 4800 + (_month - 14) /
          ↪  12)) / 4 + (367 * (_month - 2 - ((_month - 14) / 12) *
          ↪ 12)) / 12 - (3 * ((_year + 4900 + (_month - 14) / 12) /
          ↪ 100)) / 4 - OFFSET19700101 (CalcTime.sol#104-112)
CalcTime._daysToDate(uint256) (CalcTime.sol#117-136) performs a
   ↪ multiplication on the result of a division:
       - N = (4 * L) / 146097 (CalcTime.sol#123)
```

```
        - L = L - (146097 * N + 3) / 4 (CalcTime.sol#124)
CalcTime._daysToDate(uint256) (CalcTime.sol#117-136) performs a
   ↪ multiplication on the result of a division:
        - _year = (4000 * (L + 1)) / 1461001 (CalcTime.sol#125)
        - L = L - (1461 * _year) / 4 + 31 (CalcTime.sol#126)
CalcTime._daysToDate(uint256) (CalcTime.sol#117-136) performs a
   ↪ multiplication on the result of a division:
        - _month = (80 * L) / 2447 (CalcTime.sol#127)
        - _day = L - (2447 * _month) / 80 (CalcTime.sol#128)
CalcTime._daysToDate(uint256) (CalcTime.sol#117-136) performs a
   ↪ multiplication on the result of a division:
        - L = _month / 11 (CalcTime.sol#129)
        - _month = _month + 2 - 12 * L (CalcTime.sol#130)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #divide-before-multiply
INFO:Detectors:
Rosca._createPot() (Rosca.sol#295-331) uses a dangerous strict equality:
        - RSD.RS == RoscaState.isStarting (Rosca.sol#296)
Rosca._createPot() (Rosca.sol#295-331) uses a dangerous strict equality:
        - currentPD.potId == RSD.members.length (Rosca.sol#303)
Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292) uses a
   ↪ dangerous strict equality:
        - require(bool,string)(RSD.RS == RoscaState.isLive,You can only
           ↪ withdraw from a live Rosca) (Rosca.sol#281-284)
Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274) uses a
   ↪ dangerous strict equality:
        - require(bool,string)(RSD.members[memberIndex[msg.sender].sub(1)
           ↪ ].memberAddress == msg.sender,You are not a member) (Rosca
           ↪ .sol#247-251)
Rosca.contributeToPot(uint256) (Rosca.sol#164-195) uses a dangerous
   ↪ strict equality:
        - require(bool,string)(RSD.RS == RoscaState.isLive,You can only
           ↪ contribute to a live Rosca) (Rosca.sol#165-168)
```

```
Rosca.contributeToPot(uint256) (Rosca.sol#164-195) uses a dangerous
    ↪ strict equality:
        - require(bool,string)(RSD.PS == PotState.isOpen,You can only
            ↪ contribute to an open pot) (Rosca.sol#169-172)
Rosca.contributeToPot(uint256) (Rosca.sol#164-195) uses a dangerous
    ↪ strict equality:
        - currentPD.potBalance == RSD.RD.goalAmount (Rosca.sol#191)
Rosca.joinRosca(string) (Rosca.sol#138-160) uses a dangerous strict
    ↪ equality:
        - require(bool,string)(memberIndex[msg.sender] == 0,You are
            ↪ already a member of this Rosca) (Rosca.sol#144-147)
Rosca.payOutPot() (Rosca.sol#198-213) uses a dangerous strict equality:
        - require(bool,string)(RSD.RS == RoscaState.isLive,!RoscaIsLive)
            ↪ (Rosca.sol#199)
Rosca.payOutPot() (Rosca.sol#198-213) uses a dangerous strict equality:
        - require(bool,string)(currentPD.potBalance == RSD.RD.goalAmount
            ↪ ,!FullyFunded) (Rosca.sol#200)
Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242) uses a
    ↪ dangerous strict equality:
        - require(bool,string)(RSD.members[memberIndex[msg.sender].sub(1)
            ↪ ].memberAddress == msg.sender,You are not a member) (Rosca
            ↪ .sol#219-223)
Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242) uses a
    ↪ dangerous strict equality:
        - require(bool,string)(RSD.members[memberIndex[_member].sub(1)].
            ↪ memberAddress == _member,They are not a member) (Rosca.sol
            ↪ #224-227)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dangerous-strict-equalities
INFO:Detectors:
Reentrancy in Rosca._withdrawFromRosca(uint256,address) (Rosca.sol
    ↪ #280-292):
        External calls:
```

```
        - require(bool,string)(RSD.RD.token.transfer(_member,_amount),
            ↪ Transfer failed) (Rosca.sol#289)
        State variables written after the call(s):
        - RSD.roscaBalance = RSD.RD.token.balanceOf(address(this)) (Rosca
            ↪ .sol#290)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
            ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
            ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
    Reentrancy in Rosca.contributeToPot(uint256) (Rosca.sol#164-195):
        External calls:
        - require(bool,string)(RSD.RD.token.transferFrom(msg.sender,
            ↪ address(this),_amount),Transfer failed) (Rosca.sol
            ↪ #177-180)
        State variables written after the call(s):
        - RSD.currentPotBalance = RSD.currentPotBalance.add(_amount) (
            ↪ Rosca.sol#181)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
            ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
            ↪ #115-133)
```

```
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.roscaBalance = RSD.RD.token.balanceOf(address(this)) (Rosca
           ↪ .sol#182)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.PS = PotState.isClosed (Rosca.sol#192)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
```

```
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
Reentrancy in Rosca.payOutPot() (Rosca.sol#198-213):
        External calls:
        - require(bool,string)(RSD.RD.token.transfer(currentPD.potOwner,
            ↪ currentPD.potBalance),Transfer failed) (Rosca.sol#202-205)
        State variables written after the call(s):
        - RSD.currentPotBalance = 0 (Rosca.sol#206)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
            ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
            ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.roscaBalance = RSD.RD.token.balanceOf(address(this)) (Rosca
            ↪ .sol#207)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
            ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
            ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
```

```
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.PS = PotState.isPayedOut (Rosca.sol#208)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.members[memberIndex[currentPD.potOwner].sub(1)].isPotted =
           ↪ true (Rosca.sol#209)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
```

```
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - _createPot() (Rosca.sol#212)
              - RSD.RS = RoscaState.isLive (Rosca.sol#301)
              - RSD.members[i].isPotted = false (Rosca.sol#307)
              - RSD.currentPotId = currentPD.potId (Rosca.sol#323)
              - RSD.currentPotBalance = currentPD.potBalance (Rosca.sol
                 ↪ #324)
              - RSD.PS = PotState.isOpen (Rosca.sol#325)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - delete currentPD.contributions (Rosca.sol#210)
        Rosca.currentPD (Rosca.sol#88) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getCurrentPotDetails() (Rosca.sol#344-346)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - _createPot() (Rosca.sol#212)
              - currentPD.potId = 1 (Rosca.sol#297)
```

```
                - currentPD.potOwner = RSD.members[memberIndex[RSD.creator
                    ↪ ].sub(1)].memberAddress (Rosca.sol#298-300)
                - currentPD.potId = 0 (Rosca.sol#304)
                - currentPD.potId = currentPD.potId + 1 (Rosca.sol#310)
                - currentPD.potOwner = RSD.members[currentPD.potId - 1].
                    ↪ memberAddress (Rosca.sol#311)
                - currentPD.potAmount = RSD.RD.goalAmount (Rosca.sol#313)
                - currentPD.potBalance = 0 (Rosca.sol#314)
                - currentPD.payoutDate = CalcTime._nextDayAndTime(RSD.RD.
                    ↪ disbDay,RSD.RD.occurrence) (Rosca.sol#315-318)
                - currentPD.deadline = CalcTime._nextDayAndTime(RSD.RD.
                    ↪ ctbDay,RSD.RD.occurrence) (Rosca.sol#319-322)
        Rosca.currentPD (Rosca.sol#88) can be used in cross function
            ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getCurrentPotDetails() (Rosca.sol#344-346)
        - Rosca.payOutPot() (Rosca.sol#198-213)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-1
INFO:Detectors:
CalcTime._daysToDate(uint256).L (CalcTime.sol#122) is written in both
        L = L - (1461 * _year) / 4 + 31 (CalcTime.sol#126)
        L = _month / 11 (CalcTime.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #write-after-write
INFO:Detectors:
Reentrancy in Rosca.contributeToPot(uint256) (Rosca.sol#164-195):
        External calls:
        - require(bool,string)(RSD.RD.token.transferFrom(msg.sender,
            ↪ address(this),_amount),Transfer failed) (Rosca.sol
            ↪ #177-180)
        State variables written after the call(s):
```

```
        - currentPD.potBalance = currentPD.potBalance.add(_amount) (Rosca
            ↪ .sol#183)
        - currentPD.contributions.push(Contribution(msg.sender,_amount,
            ↪ block.timestamp)) (Rosca.sol#184-190)
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
        ↪ #reentrancy-vulnerabilities-2
    INFO:Detectors:
    Reentrancy in Rosca._withdrawFromRosca(uint256,address) (Rosca.sol
        ↪ #280-292):
        External calls:
        - require(bool,string)(RSD.RD.token.transfer(_member,_amount),
            ↪ Transfer failed) (Rosca.sol#289)
        Event emitted after the call(s):
        - WithdrawalExecuted(_member,_amount,block.timestamp) (Rosca.sol
            ↪ #291)
    Reentrancy in Rosca.contributeToPot(uint256) (Rosca.sol#164-195):
        External calls:
        - require(bool,string)(RSD.RD.token.transferFrom(msg.sender,
            ↪ address(this),_amount),Transfer failed) (Rosca.sol
            ↪ #177-180)
        Event emitted after the call(s):
        - PotFunded(msg.sender,_amount) (Rosca.sol#194)
    Reentrancy in Rosca.payOutPot() (Rosca.sol#198-213):
        External calls:
        - require(bool,string)(RSD.RD.token.transfer(currentPD.potOwner,
            ↪ currentPD.potBalance),Transfer failed) (Rosca.sol#202-205)
        Event emitted after the call(s):
        - CreatedPot(currentPD.potOwner,currentPD.deadline,currentPD.
            ↪ payoutDate) (Rosca.sol#326-330)
                - _createPot() (Rosca.sol#212)
        - PotPayedOut(currentPD.potOwner,dueAmount) (Rosca.sol#211)
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
        ↪ #reentrancy-vulnerabilities-3
    INFO:Detectors:
```

```
CalcTime._nextDayAndTime(string,string) (CalcTime.sol#20-55) uses
    ↪ timestamp for comparisons
        Dangerous comparisons:
        - nextTimeStamp <= block.timestamp (CalcTime.sol#40)
        - nextTimeStamp <= block.timestamp (CalcTime.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #block-timestamp
INFO:Detectors:
Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274) compares to
    ↪ a boolean constant:
        -require(bool,string)(transactions[_requestIdx].isExecuted ==
            ↪ false,Transaction already executed) (Rosca.sol#252-255)
Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274) compares to
    ↪ a boolean constant:
        -require(bool,string)(approvals[_requestIdx][msg.sender] == false
            ↪ ,You have already approved this transaction) (Rosca.sol
            ↪ #256-259)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #boolean-equality
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.19']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20
            ↪ .sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/
            ↪ SafeMath.sol#4)
        - ^0.8.19 (CalcTime.sol#2)
        - ^0.8.19 (Rosca.sol#8)
        - ^0.8.19 (RoscaSpaces.sol#8)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used
INFO:Detectors:
SafeMath.div(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#135-137) is never used and should be
```

```
    ↪ removed
SafeMath.div(uint256,uint256,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/math/SafeMath.sol#187-192) is never used and
    ↪ should be removed
SafeMath.mod(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#151-153) is never used and should be
    ↪ removed
SafeMath.mod(uint256,uint256,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/math/SafeMath.sol#209-214) is never used and
    ↪ should be removed
SafeMath.mul(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#121-123) is never used and should be
    ↪ removed
SafeMath.sub(uint256,uint256,string) (node_modules/@openzeppelin/
    ↪ contracts/utils/math/SafeMath.sol#168-173) is never used and
    ↪ should be removed
SafeMath.tryAdd(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#22-28) is never used and should be
    ↪ removed
SafeMath.tryDiv(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#64-69) is never used and should be
    ↪ removed
SafeMath.tryMod(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#76-81) is never used and should be
    ↪ removed
SafeMath.tryMul(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#47-57) is never used and should be
    ↪ removed
SafeMath.trySub(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#35-40) is never used and should be
    ↪ removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code
INFO:Detectors:
```

```
Pragma version^0.8.19 (CalcTime.sol#2) necessitates a version too recent
   ↪  to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.19 (Rosca.sol#8) necessitates a version too recent to
   ↪  be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.19 (RoscaSpaces.sol#8) necessitates a version too
   ↪ recent to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
   ↪ IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/
   ↪ SafeMath.sol#4) allows old versions
solc-0.8.21 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #incorrect-versions-of-solidity
INFO:Detectors:
Parameter Rosca.joinRosca(string)._authCode (Rosca.sol#138) is not in
   ↪ mixedCase
Parameter Rosca.contributeToPot(uint256)._amount (Rosca.sol#164) is not
   ↪ in mixedCase
Parameter Rosca.withdrawalRequest(address,uint256)._member (Rosca.sol
   ↪ #218) is not in mixedCase
Parameter Rosca.withdrawalRequest(address,uint256)._amount (Rosca.sol
   ↪ #218) is not in mixedCase
Parameter Rosca.approveWithdrawalRequest(uint256)._requestIdx (Rosca.sol
   ↪ #246) is not in mixedCase
Variable Rosca.RSD (Rosca.sol#86) is not in mixedCase
Parameter RoscaSpaces.createRoscaSpace(RoscaDetails,string)._RD (
   ↪ RoscaSpaces.sol#30) is not in mixedCase
Parameter RoscaSpaces.createRoscaSpace(RoscaDetails,string)._aCode (
   ↪ RoscaSpaces.sol#31) is not in mixedCase
Parameter RoscaSpaces.getRoscaSpacesByOwner(address)._owner (RoscaSpaces
   ↪ .sol#48) is not in mixedCase
Parameter RoscaSpaces.getRoscaSpaceByOwnernAddress(address,address).
   ↪ _owner (RoscaSpaces.sol#54) is not in mixedCase
```

```
Parameter RoscaSpaces.getRoscaSpaceByOwnernAddress(address,address).
    ↪ _roscaAddress (RoscaSpaces.sol#55) is not in mixedCase
Parameter RoscaSpaces.getRoscaSpaceByAddress(address)._roscaAddress (
    ↪ RoscaSpaces.sol#61) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #conformance-to-solidity-naming-conventions
INFO:Detectors:
CalcTime.SECONDS_PER_HOUR (CalcTime.sol#6) is never used in CalcTime (
    ↪ CalcTime.sol#4-137)
CalcTime.SECONDS_PER_MINUTE (CalcTime.sol#7) is never used in CalcTime (
    ↪ CalcTime.sol#4-137)
CalcTime.DOW_MON (CalcTime.sol#10) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_TUE (CalcTime.sol#11) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_WED (CalcTime.sol#12) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_THU (CalcTime.sol#13) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_FRI (CalcTime.sol#14) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_SAT (CalcTime.sol#15) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_SUN (CalcTime.sol#16) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-state-variable
INFO:Detectors:
CalcTime._nextDayAndTime(string,string) (CalcTime.sol#20-55) uses a weak
    ↪  PRNG: "dayOfWeek = ((_days + 3) % 7) + 1 (CalcTime.sol#33)"
CalcTime._nextDayAndTime(string,string) (CalcTime.sol#20-55) uses a weak
    ↪  PRNG: "nextDay_scope_0 = day + ((28 + _day - dayOfWeek) % 28) (
    ↪ CalcTime.sol#45)"
```

```
CalcTime._nextDayAndTime(string,string) (CalcTime.sol#20-55) uses a weak
   ↪  PRNG: "nextDay = day + ((7 + _day - dayOfWeek) % 7) (CalcTime.
   ↪ sol#36)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #weak-PRNG
INFO:Detectors:
CalcTime._daysFromDate(uint256,uint256,uint256) (CalcTime.sol#94-115)
   ↪ performs a multiplication on the result of a division:
       - __days = _day - 32075 + (1461 * (_year + 4800 + (_month - 14) /
          ↪  12)) / 4 + (367 * (_month - 2 - ((_month - 14) / 12) *
          ↪ 12)) / 12 - (3 * ((_year + 4900 + (_month - 14) / 12) /
          ↪ 100)) / 4 - OFFSET19700101 (CalcTime.sol#104-112)
CalcTime._daysToDate(uint256) (CalcTime.sol#117-136) performs a
   ↪ multiplication on the result of a division:
       - N = (4 * L) / 146097 (CalcTime.sol#123)
       - L = L - (146097 * N + 3) / 4 (CalcTime.sol#124)
CalcTime._daysToDate(uint256) (CalcTime.sol#117-136) performs a
   ↪ multiplication on the result of a division:
       - _year = (4000 * (L + 1)) / 1461001 (CalcTime.sol#125)
       - L = L - (1461 * _year) / 4 + 31 (CalcTime.sol#126)
CalcTime._daysToDate(uint256) (CalcTime.sol#117-136) performs a
   ↪ multiplication on the result of a division:
       - _month = (80 * L) / 2447 (CalcTime.sol#127)
       - _day = L - (2447 * _month) / 80 (CalcTime.sol#128)
CalcTime._daysToDate(uint256) (CalcTime.sol#117-136) performs a
   ↪ multiplication on the result of a division:
       - L = _month / 11 (CalcTime.sol#129)
       - _month = _month + 2 - 12 * L (CalcTime.sol#130)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #divide-before-multiply
INFO:Detectors:
Rosca._createPot() (Rosca.sol#295-331) uses a dangerous strict equality:
       - RSD.RS == RoscaState.isStarting (Rosca.sol#296)
Rosca._createPot() (Rosca.sol#295-331) uses a dangerous strict equality:
```

```
        - currentPD.potId == RSD.members.length (Rosca.sol#303)
Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292) uses a
    ↪ dangerous strict equality:
        - require(bool,string)(RSD.RS == RoscaState.isLive,You can only
            ↪ withdraw from a live Rosca) (Rosca.sol#281-284)
Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274) uses a
    ↪ dangerous strict equality:
        - require(bool,string)(RSD.members[memberIndex[msg.sender].sub(1)
            ↪ ].memberAddress == msg.sender,You are not a member) (Rosca
            ↪ .sol#247-251)
Rosca.contributeToPot(uint256) (Rosca.sol#164-195) uses a dangerous
    ↪ strict equality:
        - require(bool,string)(RSD.RS == RoscaState.isLive,You can only
            ↪ contribute to a live Rosca) (Rosca.sol#165-168)
Rosca.contributeToPot(uint256) (Rosca.sol#164-195) uses a dangerous
    ↪ strict equality:
        - require(bool,string)(RSD.PS == PotState.isOpen,You can only
            ↪ contribute to an open pot) (Rosca.sol#169-172)
Rosca.contributeToPot(uint256) (Rosca.sol#164-195) uses a dangerous
    ↪ strict equality:
        - currentPD.potBalance == RSD.RD.goalAmount (Rosca.sol#191)
Rosca.joinRosca(string) (Rosca.sol#138-160) uses a dangerous strict
    ↪ equality:
        - require(bool,string)(memberIndex[msg.sender] == 0,You are
            ↪ already a member of this Rosca) (Rosca.sol#144-147)
Rosca.payOutPot() (Rosca.sol#198-213) uses a dangerous strict equality:
        - require(bool,string)(RSD.RS == RoscaState.isLive,!RoscaIsLive)
            ↪ (Rosca.sol#199)
Rosca.payOutPot() (Rosca.sol#198-213) uses a dangerous strict equality:
        - require(bool,string)(currentPD.potBalance == RSD.RD.goalAmount
            ↪ ,!FullyFunded) (Rosca.sol#200)
Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242) uses a
    ↪ dangerous strict equality:
```

```
        - require(bool,string)(RSD.members[memberIndex[msg.sender].sub(1)
            ↪ ].memberAddress == msg.sender,You are not a member) (Rosca
            ↪ .sol#219-223)
    Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242) uses a
        ↪ dangerous strict equality:
        - require(bool,string)(RSD.members[memberIndex[_member].sub(1)].
            ↪ memberAddress == _member,They are not a member) (Rosca.sol
            ↪ #224-227)
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
        ↪ #dangerous-strict-equalities
    INFO:Detectors:
    Reentrancy in Rosca._withdrawFromRosca(uint256,address) (Rosca.sol
        ↪ #280-292):
        External calls:
        - require(bool,string)(RSD.RD.token.transfer(_member,_amount),
            ↪ Transfer failed) (Rosca.sol#289)
        State variables written after the call(s):
        - RSD.roscaBalance = RSD.RD.token.balanceOf(address(this)) (Rosca
            ↪ .sol#290)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
            ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
            ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
    Reentrancy in Rosca.contributeToPot(uint256) (Rosca.sol#164-195):
```

```
        External calls:
        - require(bool,string)(RSD.RD.token.transferFrom(msg.sender,
            ↪ address(this),_amount),Transfer failed) (Rosca.sol
            ↪ #177-180)
        State variables written after the call(s):
        - RSD.currentPotBalance = RSD.currentPotBalance.add(_amount) (
            ↪ Rosca.sol#181)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
            ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
            ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.roscaBalance = RSD.RD.token.balanceOf(address(this)) (Rosca
            ↪ .sol#182)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
            ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
            ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
```

```
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.PS = PotState.isClosed (Rosca.sol#192)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
    Reentrancy in Rosca.payOutPot() (Rosca.sol#198-213):
        External calls:
        - require(bool,string)(RSD.RD.token.transfer(currentPD.potOwner,
           ↪ currentPD.potBalance),Transfer failed) (Rosca.sol#202-205)
        State variables written after the call(s):
        - RSD.currentPotBalance = 0 (Rosca.sol#206)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
```

```
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.roscaBalance = RSD.RD.token.balanceOf(address(this)) (Rosca
           ↪ .sol#207)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.PS = PotState.isPayedOut (Rosca.sol#208)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
```

```
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - RSD.members[memberIndex[currentPD.potOwner].sub(1)].isPotted =
           ↪ true (Rosca.sol#209)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - _createPot() (Rosca.sol#212)
              - RSD.RS = RoscaState.isLive (Rosca.sol#301)
              - RSD.members[i].isPotted = false (Rosca.sol#307)
              - RSD.currentPotId = currentPD.potId (Rosca.sol#323)
              - RSD.currentPotBalance = currentPD.potBalance (Rosca.sol
                 ↪ #324)
              - RSD.PS = PotState.isOpen (Rosca.sol#325)
        Rosca.RSD (Rosca.sol#86) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca._withdrawFromRosca(uint256,address) (Rosca.sol#280-292)
        - Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274)
        - Rosca.constructor(RoscaDetails,string,address) (Rosca.sol
           ↪ #115-133)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getMembers() (Rosca.sol#349-351)
        - Rosca.getRoscaDetails() (Rosca.sol#335-341)
```

```
        - Rosca.joinRosca(string) (Rosca.sol#138-160)
        - Rosca.nextPot() (Rosca.sol#354-356)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - Rosca.withdrawalRequest(address,uint256) (Rosca.sol#218-242)
        - delete currentPD.contributions (Rosca.sol#210)
        Rosca.currentPD (Rosca.sol#88) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getCurrentPotDetails() (Rosca.sol#344-346)
        - Rosca.payOutPot() (Rosca.sol#198-213)
        - _createPot() (Rosca.sol#212)
              - currentPD.potId = 1 (Rosca.sol#297)
              - currentPD.potOwner = RSD.members[memberIndex[RSD.creator
                 ↪ ].sub(1)].memberAddress (Rosca.sol#298-300)
              - currentPD.potId = 0 (Rosca.sol#304)
              - currentPD.potId = currentPD.potId + 1 (Rosca.sol#310)
              - currentPD.potOwner = RSD.members[currentPD.potId - 1].
                 ↪ memberAddress (Rosca.sol#311)
              - currentPD.potAmount = RSD.RD.goalAmount (Rosca.sol#313)
              - currentPD.potBalance = 0 (Rosca.sol#314)
              - currentPD.payoutDate = CalcTime._nextDayAndTime(RSD.RD.
                 ↪ disbDay,RSD.RD.occurrence) (Rosca.sol#315-318)
              - currentPD.deadline = CalcTime._nextDayAndTime(RSD.RD.
                 ↪ ctbDay,RSD.RD.occurrence) (Rosca.sol#319-322)
        Rosca.currentPD (Rosca.sol#88) can be used in cross function
           ↪ reentrancies:
        - Rosca._createPot() (Rosca.sol#295-331)
        - Rosca.contributeToPot(uint256) (Rosca.sol#164-195)
        - Rosca.getCurrentPotDetails() (Rosca.sol#344-346)
        - Rosca.payOutPot() (Rosca.sol#198-213)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities-1
INFO:Detectors:
```

```
CalcTime._daysToDate(uint256).L (CalcTime.sol#122) is written in both
        L = L - (1461 * _year) / 4 + 31 (CalcTime.sol#126)
        L = _month / 11 (CalcTime.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #write-after-write
INFO:Detectors:
Reentrancy in Rosca.contributeToPot(uint256) (Rosca.sol#164-195):
        External calls:
        - require(bool,string)(RSD.RD.token.transferFrom(msg.sender,
            ↪ address(this),_amount),Transfer failed) (Rosca.sol
            ↪ #177-180)
        State variables written after the call(s):
        - currentPD.potBalance = currentPD.potBalance.add(_amount) (Rosca
            ↪ .sol#183)
        - currentPD.contributions.push(Contribution(msg.sender,_amount,
            ↪ block.timestamp)) (Rosca.sol#184-190)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Rosca._withdrawFromRosca(uint256,address) (Rosca.sol
    ↪ #280-292):
        External calls:
        - require(bool,string)(RSD.RD.token.transfer(_member,_amount),
            ↪ Transfer failed) (Rosca.sol#289)
        Event emitted after the call(s):
        - WithdrawalExecuted(_member,_amount,block.timestamp) (Rosca.sol
            ↪ #291)
Reentrancy in Rosca.contributeToPot(uint256) (Rosca.sol#164-195):
        External calls:
        - require(bool,string)(RSD.RD.token.transferFrom(msg.sender,
            ↪ address(this),_amount),Transfer failed) (Rosca.sol
            ↪ #177-180)
        Event emitted after the call(s):
        - PotFunded(msg.sender,_amount) (Rosca.sol#194)
```

```
Reentrancy in Rosca.payOutPot() (Rosca.sol#198-213):
        External calls:
        - require(bool,string)(RSD.RD.token.transfer(currentPD.potOwner,
            ↪ currentPD.potBalance),Transfer failed) (Rosca.sol#202-205)
        Event emitted after the call(s):
        - CreatedPot(currentPD.potOwner,currentPD.deadline,currentPD.
            ↪ payoutDate) (Rosca.sol#326-330)
                - _createPot() (Rosca.sol#212)
        - PotPayedOut(currentPD.potOwner,dueAmount) (Rosca.sol#211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3
INFO:Detectors:
CalcTime._nextDayAndTime(string,string) (CalcTime.sol#20-55) uses
    ↪ timestamp for comparisons
        Dangerous comparisons:
        - nextTimeStamp <= block.timestamp (CalcTime.sol#40)
        - nextTimeStamp <= block.timestamp (CalcTime.sol#49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #block-timestamp
INFO:Detectors:
Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274) compares to
    ↪ a boolean constant:
        -require(bool,string)(transactions[_requestIdx].isExecuted ==
            ↪ false,Transaction already executed) (Rosca.sol#252-255)
Rosca.approveWithdrawalRequest(uint256) (Rosca.sol#246-274) compares to
    ↪ a boolean constant:
        -require(bool,string)(approvals[_requestIdx][msg.sender] == false
            ↪ ,You have already approved this transaction) (Rosca.sol
            ↪ #256-259)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #boolean-equality
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.19']
```

```
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20
          ↪ .sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/
          ↪ SafeMath.sol#4)
        - ^0.8.19 (CalcTime.sol#2)
        - ^0.8.19 (Rosca.sol#8)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #different-pragma-directives-are-used
INFO:Detectors:
SafeMath.div(uint256,uint256) (node_modules/@openzeppelin/contracts/
   ↪ utils/math/SafeMath.sol#135-137) is never used and should be
   ↪ removed
SafeMath.div(uint256,uint256,string) (node_modules/@openzeppelin/
   ↪ contracts/utils/math/SafeMath.sol#187-192) is never used and
   ↪ should be removed
SafeMath.mod(uint256,uint256) (node_modules/@openzeppelin/contracts/
   ↪ utils/math/SafeMath.sol#151-153) is never used and should be
   ↪ removed
SafeMath.mod(uint256,uint256,string) (node_modules/@openzeppelin/
   ↪ contracts/utils/math/SafeMath.sol#209-214) is never used and
   ↪ should be removed
SafeMath.mul(uint256,uint256) (node_modules/@openzeppelin/contracts/
   ↪ utils/math/SafeMath.sol#121-123) is never used and should be
   ↪ removed
SafeMath.sub(uint256,uint256,string) (node_modules/@openzeppelin/
   ↪ contracts/utils/math/SafeMath.sol#168-173) is never used and
   ↪ should be removed
SafeMath.tryAdd(uint256,uint256) (node_modules/@openzeppelin/contracts/
   ↪ utils/math/SafeMath.sol#22-28) is never used and should be
   ↪ removed
SafeMath.tryDiv(uint256,uint256) (node_modules/@openzeppelin/contracts/
   ↪ utils/math/SafeMath.sol#64-69) is never used and should be
   ↪ removed
```

```
SafeMath.tryMod(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#76-81) is never used and should be
    ↪ removed
SafeMath.tryMul(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#47-57) is never used and should be
    ↪ removed
SafeMath.trySub(uint256,uint256) (node_modules/@openzeppelin/contracts/
    ↪ utils/math/SafeMath.sol#35-40) is never used and should be
    ↪ removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code
INFO:Detectors:
Pragma version^0.8.19 (CalcTime.sol#2) necessitates a version too recent
    ↪  to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.19 (Rosca.sol#8) necessitates a version too recent to
    ↪  be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/
    ↪ SafeMath.sol#4) allows old versions
solc-0.8.21 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity
INFO:Detectors:
Parameter Rosca.joinRosca(string)._authCode (Rosca.sol#138) is not in
    ↪ mixedCase
Parameter Rosca.contributeToPot(uint256)._amount (Rosca.sol#164) is not
    ↪ in mixedCase
Parameter Rosca.withdrawalRequest(address,uint256)._member (Rosca.sol
    ↪ #218) is not in mixedCase
Parameter Rosca.withdrawalRequest(address,uint256)._amount (Rosca.sol
    ↪ #218) is not in mixedCase
Parameter Rosca.approveWithdrawalRequest(uint256)._requestIdx (Rosca.sol
    ↪ #246) is not in mixedCase
```

```
Variable Rosca.RSD (Rosca.sol#86) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #conformance-to-solidity-naming-conventions
INFO:Detectors:
CalcTime.SECONDS_PER_HOUR (CalcTime.sol#6) is never used in CalcTime (
    ↪ CalcTime.sol#4-137)
CalcTime.SECONDS_PER_MINUTE (CalcTime.sol#7) is never used in CalcTime (
    ↪ CalcTime.sol#4-137)
CalcTime.DOW_MON (CalcTime.sol#10) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_TUE (CalcTime.sol#11) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_WED (CalcTime.sol#12) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_THU (CalcTime.sol#13) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_FRI (CalcTime.sol#14) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_SAT (CalcTime.sol#15) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
CalcTime.DOW_SUN (CalcTime.sol#16) is never used in CalcTime (CalcTime.
    ↪ sol#4-137)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-state-variable
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

In this audit, we examined the design and implementation of Clixpesa contract and discovered several issues of varying severity. Clixpesa Solutions Ltd team addressed issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Blockhat auditors advised Clixpesa Solutions Ltd Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

BLOCKHAT
SECURITY

For a Smart Contract Audit, contact us at contact@blockhat.io