



Smart Contract Audit Report

April, 2024



DEFIMOON PROJECT

Audit and
Development

CONTACTS

defimoon.org
audit@defimoon.org
• defimoon.org
• [defimoonorg](https://defimoon.org)
• [defimoon](https://defimoon.org)
• [defimoonorg](https://defimoon.org)

Contents

1. Disclaimer	3
2. Overview	4
3. Project centralization risks	8
4. Found issues	9
5. Contracts	13
6. Conclusion	29
Appendix A. Issues severity classification	30
Appendix B. Issue status description	31
Appendix C. List of examined issue types	32
Appendix D. Centralization risks classification	33

Disclaimer

This audit is not financial, investment, or any other kind of advice and could be used for informational purposes only. This report is not a substitute for doing your own research and due diligence should always be paid in full to any project. Defimoon is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Defimoon has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Defimoon is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. Defimoon has no connection to the project other than the conduction of this audit and has no obligations other than to publish an objective report. Defimoon will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Defimoon assumes that the provided information and materials were not altered, suppressed, or misleading. This report is published by Defimoon, and Defimoon has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Defimoon. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

Audit Information

Defimoon utilizes both manual and automated auditing approach to cover the most ground possible. We begin with generic static analysis automated tools to quickly assess the overall state of the contract. We then move to a comprehensive manual code analysis, which enables us to find security flaws that automated tools would miss. Finally, we conduct an extensive unit testing to make sure contract behaves as expected under stress conditions.

In our decision making process we rely on finding located via the manual code inspection and testing. If an automated tool raises a possible vulnerability, we always investigate it further manually to make a final verdict. All our tests are run in a special test environment which matches the "real world" situations and we utilize exact copies of the published or provided contracts.

While conducting the audit, the Defimoon security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Defimoon assesses the risks and assigns a risk level to each section together with an explanatory comment.

2. Overview

HashEx was commissioned by the **Zakat** team to perform an audit of their smart contract. The audit was conducted between **2024-04-09** and **2024-04-16**.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code was provided directly in .sol files.

SHA-1 hashes of the audited files are:

./interfaces/IPool.sol **9b257f83499e8a56833f35043550823c92d16273**

./interfaces/IConfiguration.sol **10dc2ddd23eb566eb080dcffd7f5da22883c6701**

./interfaces/IVault.sol **1adec7ab6daffab284f9e745e6c4036efd0a9a6e**

./interfaces/IPoolManager.sol **d8b77723bdfc817851228adce07b2f3d352a92a8**

./interfaces/IMint.sol **5325b6e20bf981f2f42345993535700f85317153**

./interfaces/IRegistry.sol **333d94e8dd3dfe5991a33fe3bffe683f8df36bee**

./libraries/Data.sol **7d152e6b742ef543e830d7d699b350c71b2dfe11**

./libraries/Errors.sol **ca1cf4a190432431321a66e634459e2faefbb0b3**

./libraries/Roles.sol **b694bfaca1a1742d7d1d70a42be7dda04f61e7ac**

./libraries/svg/JSON.sol **f317eca37c63ae4bce510f6c4742c5161ed104c1**

./libraries/svg/SVG.sol **6e32f3f3bcc16bb00dc08e06a6dab73fe9fb2f12**
./libraries/svg/HexStrings.sol **a54c2ca3610c887ed20e37a4fead72f87cd1140a**
./libraries/svg/Colors.sol **aa855ef8daceb94613f1a60826b4f33f90fa104d**
./libraries/logic/FeeLogic.sol **7d325461c5b617a37597eb2d4b2b5890ae97b40d**
./libraries/logic/VaultLogic.sol **a7a8f4fed1ea7c26fb534ec2aa1d2363cb6025d9**
./platform/access-control/Authority.sol **ff5156b65d89459ff6836b03d9d1ad5877470347**
./platform/nft/Descriptor.sol **5071311fd87665b908d565ea805f7ed9ec0b75b**
./platform/nft/Mint.sol **f9ca2865fd91b6d6517d5a9c86e746571d8b9fa0**
./platform/Configuration.sol **a511054de3e2c541a486d1152296179c5ca66d82**
./platform/pool/Vault.sol **83bf6e40ebc1861f78ac395f9b0a357103f930e2**
./platform/pool/Pool.sol **b59a7cc5538f21e0f5ad549eb8e8b60423fa4b18**
./platform/pool/PoolManager.sol **5281b3cd1271b5540407939145292b3eef6da458**
./platform/registry/RegisteredUpgradeable.sol **626a0fd6c1552450124908d991fad699e0d924f8**
./platform/registry/Registered.sol **829af0366c42a4b3a561a521744d635594ef3b64**
./platform/registry/Registry.sol **aa1ee9245b4b7776fb243dadc5b7ca06813d842d**
./platform/request/RequestManager.sol **6668f1eac83aabfb01d1df55ca708eec1e1a60f5**

Update. The team has responded to this report. The updated contracts are available in the [@kozyilmaz/zakat GitHub repository](#) in the v1.1.0 tree.

2.1 Summary

Project name	Zakat
URL	
Platform	Haqq Network
Language	Solidity
Centralization level	● High
Centralization risk	● High

2.2 Contracts

Name	Address
PoolManager	
Pool	
Vault	
Mint	
Descriptor	
Registry	
Registered and RegisteredUpgradeable	
Configuration	
RequestManager	

Authority

Libraries

Interfaces

3. Project centralization risks

The contracts of the project are designed to be deployed via upgradable proxies, meaning that accounts with privileged access can change implementations of the contracts. We recommend to avoid leaving an unspent allowance to an upgradable contract.

4. Found issues



Critical	1 (3%)
High	1 (3%)
Medium	2 (6%)
Low	20 (61%)
Info	9 (27%)

Ceb. PoolManager

ID	Severity	Title	Status
CebI7a	● Low	Concurrent authorization models	✓ Resolved
CebI7b	● Low	Lack of input validation	✓ Resolved
CebI74	● Info	No function that returns length	✓ Resolved
CebI75	● Info	No visibility modifier	✓ Resolved

Cec. Pool

ID	Severity	Title	Status
CecI7d	● High	Lack of input validation	✓ Resolved
CecI80	● Medium	Possibly locked native coins	✓ Resolved
CecI8b	● Low	Unused variable	✓ Resolved
CecI7c	● Low	Lack of storage gap	✓ Resolved

Cec17f	● Low	Lack of events	✓ Resolved
Cec17e	● Info	Lack of NatSpec descriptions	✓ Resolved

Ced. Vault

ID	Severity	Title	Status
Ced18a	● Critical	Owner can override consensus	✓ Resolved
Ced183	● Low	Gas optimizations	⌚+ Partially fixed
Ced182	● Low	Lack of storage gap	✓ Resolved
Ced184	● Low	Limited reentrancy protection	✓ Resolved
Ced186	● Low	Lack of error message	✓ Resolved
Ced185	● Info	Lack of NatSpec descriptions	✓ Resolved
Ced181	● Info	Possibly unaddressed native coins	🕒 Acknowledged

Ce2. Mint

ID	Severity	Title	Status
Ce2I87	● Low	Gas optimizations	🕒 Acknowledged
Ce2I88	● Low	Lack of storage gap	✓ Resolved
Ce2I89	● Info	Lack of NatSpec descriptions	✓ Resolved

Ce9. Registry

ID	Severity	Title	Status
Ce9I8d	● Medium	Lack of input validation	✓ Resolved
Ce9I8c	● Low	Lack of storage gap	✓ Resolved
Ce9I71	● Low	No events	✓ Resolved
Ce9I70	● Low	Gas optimization	✓ Resolved
Ce9I72	● Info	Not used variable	✓ Resolved

Cea. Registered and RegisteredUpgradeable

ID	Severity	Title	Status
CeaI8e	● Low	Lack of storage gap	✓ Resolved
CeaI73	● Low	Gas optimizations	✓ Resolved

Cd8. RequestManager

ID	Severity	Title	Status
Cd8I8f	● Low	Gas optimizations	⌚+ Partially fixed
Cd8I90	● Low	Lack of storage gap	⌚ Acknowledged
Cd8I91	● Info	Possible DoS by gas limit	✓ Resolved
Cd8I92	● Info	Function access	✓ Resolved

Cef. Libraries

ID	Severity	Title	Status
Cefl78	● Low	Lack of input validation	✓ Resolved
Cefl77	● Low	Gas optimizations	✓ Resolved

5. Contracts

Ceb. PoolManager

Overview

A factory contract to deploy and upgrade existing Pool contracts. Only privileged accounts can interact with the PoolManager.

Issues

CebI7a Concurrent authorization models

Low

Resolved

The PoolManager contract uses the Beacon proxy pattern for upgrading deployed Pool contracts. The `upgradeTo()` function requires both external AccessManager and local Ownable authorization, unnecessary complicating the upgrade process.

```
contract PoolManager is
    IPoolManager,
    AccessManaged,
    Pausable,
    Registered,
    UpgradeableBeacon
{
    ...
    function upgradeTo(address _implementation) public override restricted {
        super.upgradeTo(_implementation);
        version++;
    }
}

contract UpgradeableBeacon is IBeacon, Ownable {
    ...
    function upgradeTo(address newImplementation) public virtual onlyOwner {
        _setImplementation(newImplementation);
    }
}
```

CebI7b Lack of input validation

Low

Resolved

There is no protection for the same name pools in the `deployPool()` function. While the current implementation of the Registry contract denies duplicated pools, future upgrades may cause the `REGISTRY.registerPool()` call to overwrite pool address.

```

function deployPool(
    PoolParameters memory _parameters
) external restricted whenNotPaused {
    ...
    bytes32 id = _calculateId(version, _parameters.name);
    ...
    REGISTRY.registerPool(id, pool, _parameters);
}

function _calculateId(
    uint256 _version,
    string memory _name
) internal view returns (bytes32) {
    return keccak256(abi.encode(address(this), _version, _name));
}

```

CebI74 No function that returns length

Info

Resolved

There are no functions that return the length of the `poolAddresses` and the `poolIds` EnumerableSets.

CebI75 No visibility modifier

Info

Resolved

Global variable `vaultImplementation` don't have a visibility modifier. Default visibility is `internal`.

Cec. Pool

Overview

A contract for user interactions. The Pool allows users to donate their funds in ERC20 tokens or in native EVM coins in exchange for NFTs. Collected funds are stored in the Pool's individual Vault contract and may be transferred out by the Pool's owner's decision.

Issues

CecI7d Lack of input validation

● High

✓ Resolved

The ERC20 payment token address is defined by the user and isn't validated in any kind, although the Pool contract contains an explicit list of supported assets and has privileged functions to update this list.

```
function depositERC20WithPermit(
    address _token,
    uint256 _value,
    uint256 _deadline,
    uint8 _permitV,
    bytes32 _permitR,
    bytes32 _permits
) external {
    vault.depositERC20WithPermit(
        _token,
        msg.sender,
        _value,
        _deadline,
        _permitV,
        _permitR,
        _permits
    );

    donations[_token] += _value;
    donationsOf[_token][msg.sender] += _value;
```

```

    _mint(msg.sender, _token, _value);
}

function depositERC20(address _token, uint256 _value) external {
    vault.depositERC20(_token, msg.sender, _value);

    donations[_token] += _value;
    donationsOf[_token][msg.sender] += _value;

    _mint(msg.sender, _token, _value);
}

```

Recommendation

Include validation of `require(isAssetEnabled(_token))` or similar to the `depositERC20()` and `depositERC20WithPermit()` functions.

CecI80 Possibly locked native coins

● Medium

✓ Resolved

The contract contains empty function to receive native EVM coins with no clear use case. Any sent coins will remain locked in the contract's balance.

```
receive() external payable {}
```

Recommendation

Remove the `receive()` function.

CecI8b Unused variable

● Low

✓ Resolved

The `factory` address is not initialized and not used anywhere in the Pool contract.

CecI7c Lack of storage gap

● Low

✓ Resolved

The Pool contract is designed to be deployed via upgradable proxy but lacks reserved storage slots for possible future upgrades.

Cecl7f Lack of events

Low

Resolved

The functions `enableAsset()`, `disableAsset()`, `submitTX()`, `signTX()`, `executeTX()`, `_mint()` change important variable in the contract storage, but no event is emitted.

We recommend adding events to these functions to make it easier to track their changes offline.

Cecl7e Lack of NatSpec descriptions

Info

Resolved

Several functions lack descriptions. We recommend writing documentation using [NatSpec Format](#). This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Ced. Vault

Overview

A vault contract to be deployed by the Pool contract that stores collected funds in form of ERC20 tokens and native EVM coins. Stored funds can be transferred out to an arbitrary address by a consensus decision of owners, defined during the contract creation.

Issues

Cedl8a Owner can override consensus

Critical

Resolved

The owners need a consensus decision to submit and execute a transfer-out transaction, but any of the owners can upgrade the implementation and override the logic.

```
function _executeTX(
    uint256 _index,
    address _executor,
```

```

    uint24 _feeRatio,
    address[] memory _feeTakers
) internal {
    ...
    if (transaction.signatureCount < threshold)
        revert NotEnoughSignatures();
}

function _authorizeUpgrade(
    address _implementation
) internal override onlyOwners {}

```

Recommendation

The upgradability should either be a consensus decision or approved via the Authority.

Cedl83 Gas optimizations

 Low

 Partially fixed

1. Zero effect code in the `depositNative()` function:

`Address.sendValue(payable(address(this)), msg.value).`

2. Excessive stored data: `transactions[id].id = id.`

3. Unnecessary reads from storage in the `_submitTX()` function: `transactions[_id]` variable.

4. Unnecessary reads from storage in the `_signTX()` function: `transaction.signatureCount`, `transaction.id`, `transaction` variables.

5. Unnecessary reads from storage in the `_executeTX()` function: `transaction.token`, `transaction` variables.

6. Unnecessary reads from storage in the `_setOwners()` function: `owners[i]` variables.

Cedl82 Lack of storage gap

 Low

 Resolved

The contract is designed to be deployed via upgradable proxy but lacks reserved storage slots for possible future upgrades.

Cedl84 Limited reentrancy protection

Low

Resolved

The **nonReentrant** modifier prevents reentrancy only for limited number of internal functions, specifically **_executeNative()**, **_executeERC20()**, **_adjustBalance()**. We recommend protect all external and public functions explicitly to prevent cross-function reentrancy.

Cedl86 Lack of error message

Low

Resolved

The **depositERC20WithPermit()** function contains possible silent failure on **IERC20Permit(_token).permit()** external call.

```
function depositERC20WithPermit(
    address _token,
    address _from,
    uint256 _value,
    uint256 _deadline,
    uint8 _permitV,
    bytes32 _permitR,
    bytes32 _permitsS
) external onlyPool {
    try
        IERC20Permit(_token).permit(
            _from,
            address(this),
            _value,
            _deadline,
            _permitV,
            _permitR,
            _permitsS
        )
    {} catch {}

    _depositERC20(_token, _from, _value);
}
```

Recommendation

Include a meaningful revert in the **catch** section.

Cedl85 Lack of NatSpec descriptions

● Info

✓ Resolved

Several functions lack descriptions. We recommend writing documentation using [NatSpec Format](#). This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Cedl81 Possibly unaddressed native coins

● Info

⌚ Acknowledged

The contract contains empty function to receive native EVM coins with no clear use case. Any sent coins will remain locked in the contract's balance without accounting via [_adjustBalance\(\)](#). In addition, small number of native coins is locked as the division remainder in the [_executeTX\(\)](#) function.

```
/// @notice Fallback function to receive native.  
receive() external payable virtual {}
```

Ce2. Mint

Overview

This is an ERC721 token that can be minted by any Pool contract (or any other granted account) to a user that made an asset donation. Newly minted token will represent his asset donation.

Issues

Ce2I87 Gas optimizations

 Low

 Acknowledged

1. The AccessManager authorization model can replace the Ownable contract inherited by the Mint contract.

Ce2I88 Lack of storage gap

 Low

 Resolved

The contract is designed to be deployed via upgradable proxy but lacks reserved storage slots for possible future upgrades.

Ce2I89 Lack of NatSpec descriptions

 Info

 Resolved

Several functions lack descriptions. We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Cf1. Descriptor

Overview

A contract to encode SVG data for the Mint NFT contract.

Ce9. Registry

Overview

A contract allowing tracking of deployed pools and donations.

Issues

Ce9I8d Lack of input validation

● Medium

✓ Resolved

A donation can be registered for a non-existent pool in the `registerDonation()` function.

There's a possible length's mismatch in the `batchRegister()` and `batchUpdate()` functions.

Recommendation

Include explicit validation of the input data.

Ce9I8c Lack of storage gap

● Low

✓ Resolved

The contract is designed to be deployed via upgradable proxy but lacks reserved storage slots for possible future upgrades.

Ce9I71 No events

● Low

✓ Resolved

These functions don't have any events:

1. `register()`
2. `batchRegister()`
3. `update()`
4. `batchUpdate()`

Ce9I70 Gas optimization

● Low

✓ Resolved

1. To the `registerPool()` function has arguments that are not used anywhere:
`_parameters.feeRatio`, `_parameters.permittedAssets`, and
`_parameters.vaultParameters`.
2. The AccessManager authorization model can replace the Ownable contract inherited by the Registry contract.
3. The `generateAddressedId()` is called twice in the `registerDonation()` function.

Ce9I72 Not used variable

● Info

✓ Resolved

Global variable `MINT_CONTRACT` isn't used anywhere.

Cea. Registered and RegisteredUpgradeable

Overview

A contract modules to store address of the Registry contract.

Issues

Ceal8e Lack of storage gap

● Low

✓ Resolved

The RegisteredUpgradeable contract is designed to be deployed via upgradable proxy but lacks reserved storage slots for possible future upgrades.

Ceal73 Gas optimizations

● Low

✓ Resolved

Global variables `REGISTRY_ADDRESS` and `REGISTRY` hold the same value. One of the two variables can be deleted.

The non-upgradable Registered contract may store the Registry address in immutable form to reduce gas consumption.

Cd7. Configuration

Overview

A contract that holds information about fee takers. It will be used in the Pool contract.

Cd8. RequestManager

Overview

An entry point for users and pool owners to create, approve, and execute requests. The contract stores information about requests that were created by users.

Issues

Cd8I8f Gas optimizations

 Low

 Partially fixed

1. The AccessManager auth model can replace the Ownable contracts inherited by the RequestManager contract.
2. Excessive stored data: `requests[id].id = id`.
3. Multiple reads from storage in the `_bumpStatus()` function: `_request.status` variable.

Cd8I90 Lack of storage gap

 Low

 Acknowledged

The contract is designed to be deployed via upgradable proxy but lacks reserved storage slots for possible future upgrades.

Cd8I91 Possible DoS by gas limit

 Info

 Resolved

The `getRequests()` function returns full array of requests and may suffer from DoS due to gas limit since `requests[]` array is never purged.

Cd8I92 Function access

Info

Resolved

The `create()` function is marked as restricted, but in fact it's a public function.

```
// -----
// Public API (Restricted)
// -----  
  
function create(  
    bytes32 _poolId,  
    address _recipient,  
    string calldata _description,  
    address _token,  
    uint256 _amount  
) external returns (uint256) {  
    uint256 id = _create(  
        _poolId,  
        _recipient,  
        _description,  
        _token,  
        _amount  
    );  
  
    return id;  
}
```

Recommendation

Clarify the desired accessibility of the `create()` function.

Cd9. Authority

Overview

Implements [AccessManager](#) functionality from OpenZeppelin library.

Cef. Libraries

Overview

Various libraries contracts for the project contracts:

- related to SVG encoding: Colors, HexStrings, JSON, SVG;
- shared structs and errors: Data, Errors, Roles;
- related to fee logic: FeeLogic, VaultLogic.

Issues

Cefl78 Lack of input validation

● Low

✓ Resolved

The SVG contract's function `formatted()` lacks validation of input parameters `_decimals` `>= _precision`. Without this safety check the `formatted()` function may experience underflow error.

```
function formatted(
    uint256 _amount,
    uint256 _decimals,
    uint256 _precision
) internal pure returns (string memory) {
    uint256 lExponent = 10 ** _decimals;
    uint256 left = (_amount / lExponent);

    uint256 rExponent = 10 ** (_decimals - _precision);
    uint256 right = (_amount / rExponent) % mod;

    string memory leftStr = left.toString();
```

```

    string memory rightStr = right.toString();

    uint256 fill = _precision - bytes(rightStr).length;

    if (fill == 4) return leftStr;

    unchecked {
        for (uint256 i; i < fill; i++) {
            rightStr = string(abi.encodePacked("0", rightStr));
        }
    }

    return string(abi.encodePacked(leftStr, ".", rightStr));
}

```

Cefl77 Gas optimizations

● Low

✓ Resolved

Data contract excessive storage usage:

1. the Transaction and Request structs contain duplicated data in **signatureCount** and **signatures.length** fields.

VaultLogic contract redundant code:

1. In the **checkThreshold()** function the **_parameters.threshold <= 0** check can be switched to **_parameters.threshold == 0** since the **uint256 threshold** can't be negative.

FeeLogic contract redundant code:

1. In the **calculateTotalFee()** function **_ratio <= 0** check can be replaced with **_ratio == 0**, it will be more gas-optimized and there is no such case when the **_ratio** variable is negative.
2. In the **clampFee()** function there is an excessive check **if (_value < 0)** that can be omitted, the variable **_value** can't be negative.
3. The function **clampFee()** isn't used anywhere.

Cf0. Interfaces

Overview

Various interface contracts for the project

contracts: IConfiguration, IMint, IPool, IPoolManager, IRegistry, IVault.

6. Conclusion

1 critical, 1 high, 2 medium, 20 low severity issues were found during the audit. 1 critical, 1 high, 2 medium, 16 low issues were resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Info.** Issues that do not impact the contract operation. Usually, info severity issues are related to code best practices, e.g. style guide.

Appendix B. Issue status description

- ✓ **Resolved.** The issue has been completely fixed.
- ✗ **Partially fixed.** Parts of the issue have been fixed but the issue is not completely resolved.
- **Acknowledged.** The team has been notified of the issue, no action has been taken.
- ⌚ **Open.** The issue remains unresolved.

Appendix C. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

Appendix D. Centralization risks classification

Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.