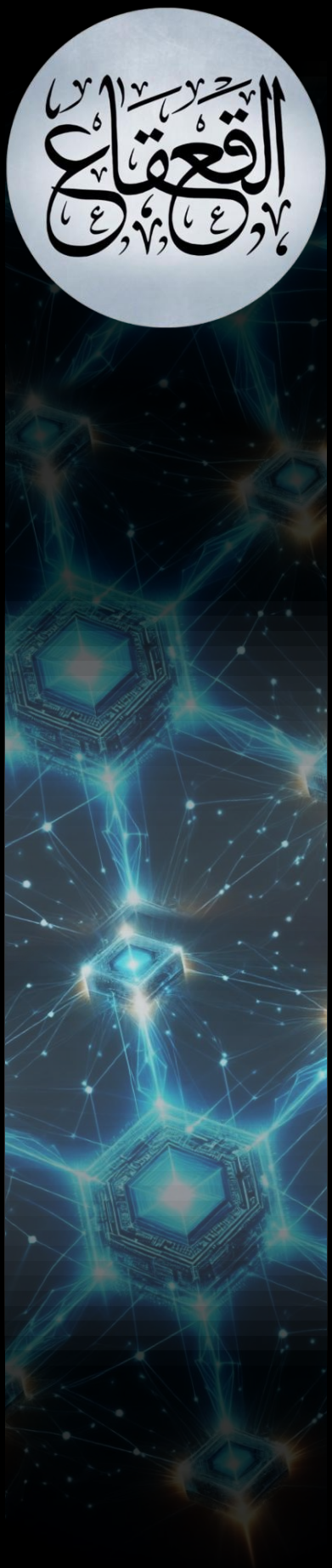
Al-Qa'qa’



Merit Systems
Auditing Report

Auditor: Al-Qa'qa’

10 October 2025

Table of Contents

1 Introduction

1.1 About Al-Qa'qa'

Al-Qa'qa' is an independent Web3 security researcher specializing in smart contract audits. Success in placing top 5 in multiple contests on Cantina and Sherlock. In addition to smart contract audits, he has moderate experience in core EVM architecture, geth.

For security consulting, reach out to him on Twitter - @Al_Qa_qa

1.2 About Merit Systems

Merit Systems enable direct monetization of GitHub repos. They create repo-owned bank accounts, simple financial tools for monetization, and automatic impact-weighted payouts to contributors.

1.3 Disclaimer

Security review cannot guarantee `100%` the protocol's safety. In the Auditing process, we try to identify all possible issues, and we cannot be sure if we missed something.

Al-Qa'qa' is not responsible for any misbehavior, bugs, or exploits affecting the audited code or any part of the deployment phase.

And change to the code after the mitigation process, puts the protocol at risk, and should be audited again.

1.4 Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.4.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

1.4.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align or little-to-no incentive

2 Executive Summary

2.1 Overview

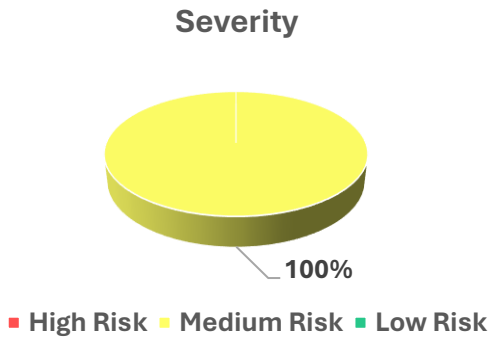
Project	Merit Systems
Repository	x402-fund-contracts (Private)
Commit Hash	20b7f2f243e91f03215c39d6e2d57a3392836589
Mitigation Hash	20b7f2f243e91f03215c39d6e2d57a3392836589
Audit Timeline	8 Oct 2025

2.2 Scope

- src/RepoFactory.sol
- src/Repo.sol
- script/Deploy.Base.s.sol

2.3 Issues Found

Severity	Count
High Risk	0
Medium Risk	1
Low Risk	0



3 Findings Summary

ID	Title	Status
M-01	A given Repo funds can't be reclaimed once it funded	Acknowledge
I-01	`Repo::fundRepo()` is not checking the validity of `amount`	Acknowledge

4 Findings

4.1 Medium Risk

4.1.1 Accumulate Reward Rate is not synced with utilization rate

Description:

Repo contracts are built to deploy a given BeaconProxy contract for a given repoId/instanceId Once deploying it. There is only one function that allows funding that repoId/instanceId by calling `fundRepo()` in the `Repo` contract

Repo::fundRepo

```
    function fundRepo(bytes calldata data) external {
        uint amount = IERC20(token).balanceOf(address(this)); // transfer whole
balance
        IERC20(token).approve(address(escrow), amount);
>>      IEscrow(escrow).fundRepo(
            repoIdentifier.repoId,
            repoIdentifier.instanceId,
            token,
            amount,
            data
        );
    }
```

The problem is that Escrow Contract has a reclaiming functionality. where in case there are no distributions happened to that repo/instance, the funder can reclaim its funds.

Escrow::reclaimRepoFunds()

```
    function reclaimRepoFunds( ... ) ... {
        require(whitelistedTokens.contains(token),
Errors.INVALID_TOKEN);
        require(amount > 0,
Errors.INVALID_AMOUNT);
>>      require(!accounts[repoId][instanceId].hasDistributions,
Errors.REPO_HAS_DISTRIBUTIONS);

        ...
    }
```

The only way a given repo has `hasDistribution` to true, which means the funders can't reclaim their funds, is in case the Repo Admin/Distributor has called `distributeFromRepo()` at least one time.

Escrow::distributeFromRepo()

```
    function distributeFromRepo( ... )
        external
        onlyRepoAdminOrDistributor(repoId, instanceId)
        returns (uint[] memory distributionIds)
```

```
    {
        ...
>>      account.hasDistributions = true;

        ...
    }
```

The function `distributeFromRepo()` is restricted to admins only, so in case a given Repo was made but is not intended to distribute rewards, or was relying `distributeFromSender` only, or get closed, etc... and there are still funds in the contract. The Funders can take there funds back by calling `reclaimRepoFunds`, but since Repo contract instance has only funding mechanism. these funds will be stucked forever, and the only way to get it back is by upgrading the Beacon implementation.

Recommendations
We should make sure the repo/instance we are funding has `hasDistribution` set to true, so that we make sure that funding contracts are non-reclaimable and the repo is working.

Status: Acknowledge

4.2 Informational Findings

4.2.1 `Repo::fundRepo()` is not checking the validity of `amount`

Description:

When funding the repo, we take all the balance of the USDC held by the contract and fund it to that repo/instance in Escrow contract. there is no check for the amount to be greater than zero, even `0` is accepted to be funded by `Repo::fundRepo()` But this is no the same as in `Escrow::fundRepo()` where it rejects the `0` transfer. besides it also will reject small values (some wei of tokens), because of fees amount should exceed fees check.

Repo::fundRepo

```
    function fundRepo(bytes calldata data) external {
        uint amount = IERC20(token).balanceOf(address(this)); // transfer whole
balance
        IERC20(token).approve(address(escrow), amount);
        IEscrow(escrow).fundRepo( ...);
    }
```

Escrow::fundRepo()

```
    function fundRepo( ... ) external {
        require(whitelistedTokens.contains(address(token)), Errors.INVALID_TOKEN);
>>      require(amount > 0,
Errors.INVALID_AMOUNT);

        token.safeTransferFrom(msg.sender, address(this), amount);

        uint feeAmount = amount.mulDivUp(feeOnFund, 10_000);
```

```
>>      require(amount > feeAmount, Errors.INVALID_AMOUNT);

        ...
    }
```

Recommendations:\

- We can check the value is greater than zero (but this will result in funding revert for too small values like `1 wei` ...
- Or having a minimum deposit amount

Status: Acknowledge