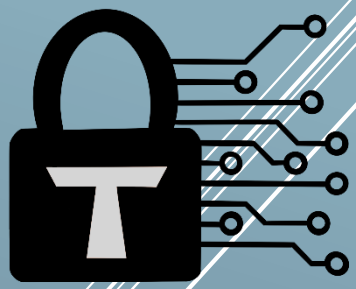
Trust Security



Smart Contract Audit

Merit Systems - Escrow

23/06/25

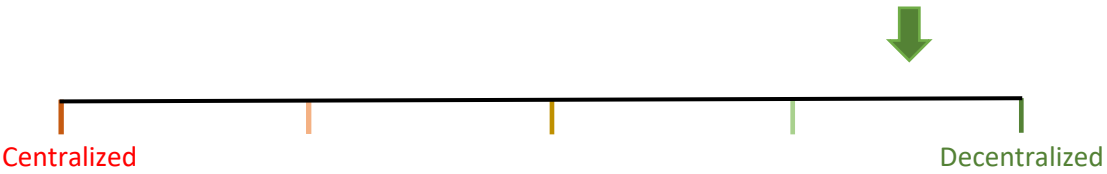
Executive summary

Category	Funding Infrastructure
Audited file count	2
Lines of Code	594
Auditor	Trust
Time period	20/06/25 – 23/06/25

Findings

Severity	Total	Fixed	Acknowledged
High	-	-	-
Medium	1	1	-
Low	-	-	-

Centralization score



Signature

Digitally signed by Trust
Date: 2025.06.23 22:49:07 +03'00'

Document properties

Versioning

Version	Date	Description
0.1	23/06/25	Client report
0.2	23/06/25	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

- Escrow.sol
- Deploy.Base.s.sol

Repository details

- **Repository URL:** https://github.com/Merit-Systems/ledger
- **Commit hash:** ba01987f98a4813fd911d8229d4024c7d0adb85e
- **Fix commit hashes:**
  - M-1: edc552fd1b128372b4e55cf4bc69ad254a5364e6
  - R-4: 0e98e4a4663901f0c49295dbd1820c9282eb8b9d
  - R-1: 194d84b01a9e4df08fd5129042af81af11083854

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Since its inception it has safeguarded over 30 clients through private services and over 30 additional projects through bug bounty submissions.

About the Auditors

Trust has established a dominating presence in the smart contract security ecosystem since 2022. He is a resident on the Immunefi, Sherlock and C4 leaderboards and is now focused in auditing and managing audit teams under Trust Security. When taking time off auditing & bug hunting, he enjoys sharing knowledge and experience with aspiring auditors through X or the Trust Security blog.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Excellent**	Project kept code as simple as possible, reducing attack risks
Documentation	**Excellent**	Project is mostly very well documented.
Best practices	**Excellent**	Project consistently adheres to industry standards.
Centralization risks	**Excellent**	Project does not introduce significant unnecessary centralization risks.

Findings

Medium severity findings

TRST-M-1 Creation of repos may fail due to nonce collisions

- **Category:** Logical flaws
- **Source:** Escrow.sol
- **Status:** Fixed

Description

The *initRepo()* function receives an owner-generated signature and creates an accounting entry. It uses a global nonce value called **setAdminNonce** to ensure replay protection. An issue occurs because the back-end must commit to a nonce allocation in the provided signature without being certain that the signature will be processed by the contract before the next request comes in. Essentially, the global nature of the nonce creates a race condition which could be triggered if multiple signature requests come in close proximity, or if a receiver chooses or accidentally does not provide it to the on-chain system.

Recommended mitigation

For each **repoId** (assuming that sufficiently identifies a developer entity), use a separate nonce mapping.

Team response

Fixed.

Mitigation review

Issue was addressed by making the nonce local to each **(repoId, instanceId)** mapping.

Additional recommendations

TRST-R-1 Remove unreachable code

During creation of distributions, it is ensured the total amount is larger than the fee amount:

```
// Validate that after fees, recipient will receive at least 1
wei
uint feeAmount = distribution.amount.mulDivUp(fee, 10_000);
require(distribution.amount > feeAmount, Errors.INVALID_AMOUNT);
```

Therefore, the code below during claiming cannot be reached:

```
// Cap fee to ensure recipient gets at least 1 wei
if (feeAmount >= distribution.amount) {
    feeAmount = distribution.amount - 1;
}
```

TRST-R-2 Improve code validations

The *onlyRepoAdmin()* and *onlyRepoAdminOrDistributor()* modifiers check the account mapping, but it is only valid to check if it exists. There should be an existence check similar to other functions.

TRST-R-3 Avoid untrusted external calls

In *distributeFromSender()*, the user-supplied **distribution.token** is called before checking it is in the whitelist (done in *_createDistribution()* after the transfer). For abundance of caution, it is recommended to first check it is in the whitelist.

TRST-R-4 Optimize setting of hasDistributions

In *distributeFromRepo()*, the **hasDistributions** is set in every loop iteration:

```
account.hasDistributions = true;
```

Since it is known distribution length is non-zero, the line above should just be called once, outside the loop.

Centralization risks

TRST-CR-1 The owner may be able to censor or block claims

The Escrow contract is designed so that the owner cannot revoke a provided claim infrastructure. However, it can effectively block claims by setting **batchLimit** to a value lower than the distribution count of a claim signature. Note that this is only possible when there is more than one distribution in the Claim.

Another way owner can block claims is by simply switching the **signer** address using *setSigner()*.