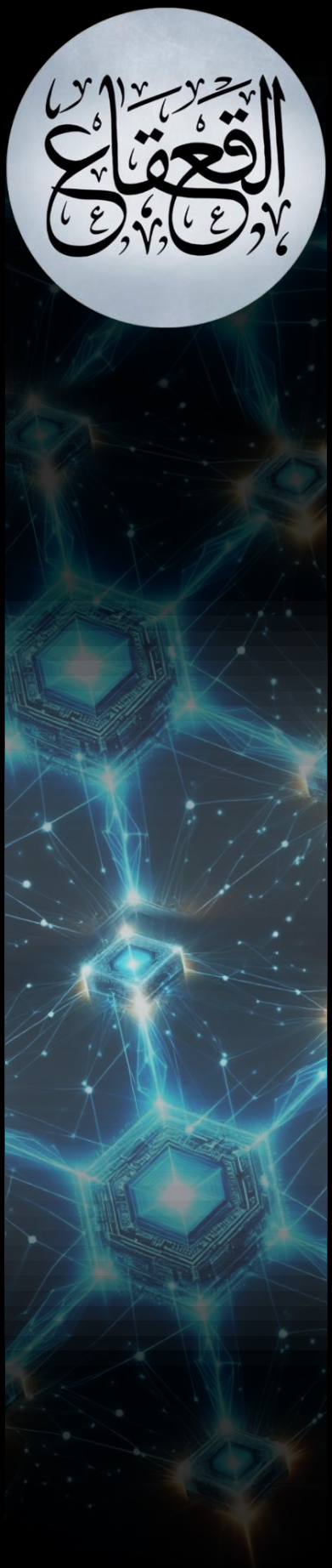
Al-Qa'qa’



Merit Systems

Auditing Report

Auditor: Al-Qa'qa’

19 March 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 code4rena 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

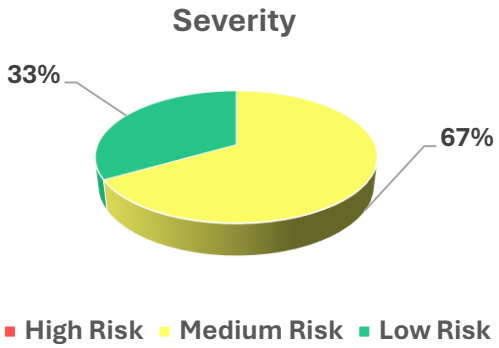
Project	Venice
Repository	Ledger (Private)
Commit Hash	87407971342432c1cf52cf378bdfa8355f612807
Mitigation Hash	c96cb43de22ab74eb80c5b4302244927b0a14722
Audit Timeline	16 Mar 2025 and 17 Mar 2025

2.2 Scope

- src/Payments/Escrow.sol

2.3 Issues Found

Severity	Count
High Risk	0
Medium Risk	2
Low Risk	1



3 Findings Summary

ID	Title	Status
M-01	The Owner of the contract can't be a Smart Contract	Acknowledge
M-02	Deadline check will always pass	Fixed
L-01	Signatures don't implement a deadline	Fixed
I-01	Incorrect format when constructing hash in `_computeClaimDomainSeparator()`	Fixed
I-02	No check if the depositId is valid or not	Fixed

4 Findings

4.1 Medium Risk

4.1.1 Accumulate Reward Rate is not synced with utilization rate

Description:
After users lock their tokens, the contract owner can allow a given `recipient` to claim tokens by signing a message verifying it.

In `setCanClaim` anyone can verify the signature and allow a given recipient from claim tokens, But the Signature is verified using `v,r,s` method

src/Payments/SplitWithLockup.sol#setCanClaim()

```
    function setCanClaim(
        address recipient,
        bool    status,
        uint8   v,
        bytes32 r,
        bytes32 s
    ) public {
        if (canClaim[recipient] == status) return;

        ...
>>      address signer = ECDSA.recover(digest, v, r, s);
        require(signer == owner, Errors.INVALID_SIGNATURE);

        ...
    }
```

The Signature should be made by the `owner` of the contract. But as the signature is passed in `v,r,s` format. This means this will only support EOA wallets, which have private keys, but in case the owner is a Smart Contract wallet like Multi Signature wallet. Then he will not be able to sign messages or recover the address.

The Owner of the Contract is a Smart Contract Multi-Signature wallet in most cases, where this is the contract's owner, and is a critical rule. The current implementation will prevent verifying against Smart Contract wallets, which can cause issues especially for contract owners, which are Multi Sig wallets in most cases.

Recommendations:

- Instead of using `v,r,s` format. use `bytes` format (a signature is a 65 bytes length as r-s-v sequence)
- Use OpenZeppelin SignatureChecker library instead of ECDSA---

Status: Acknowledge

4.1.2 Deadline check will always pass

context:
This issue was introduced when fixing issue L-01

Description:
The deadline checks add `block.timestamp` when checking, where instead of checking `timestamp` <= `deadline` it makes another addition to the `timestamp` to the deadline.

Escrow.sol#L202

```
    function setCanClaim( ... ) public {
        if (canClaim[recipient] == status) return;

>>      require(block.timestamp <= block.timestamp + deadline,
Errors.SIGNATURE_EXPIRED);
        ...
    }
```

This check will always end up being true as `X + positive Number` is always greater than or equal `X`

Recommendations:
remove adding `block.timestamp` to `deadline` in comparison

Status: Fixed

4.2 Low Findings

4.2.1 Signatures don't implement a deadline

Description:
The Signatures made by the owner of the contract to set recipients to either can claim assets or not. is not implementing a `deadline` parameter, where the Claim Structure only has the recipient, status, and the nonce

> src/Payments/SplitWithLockup.sol#setCanClaim

```
    function setCanClaim( ... ) public {
        if (canClaim[recipient] == status) return;

>>      bytes32 structHash = keccak256(
            abi.encode(
                CLAIM_TYPEHASH,
                recipient,
                status,
                recipientNonces[recipient]
            )
        );
    }
```

The signatures made by the owner, will be valid forever till it is consumed. This is not Good Practice for Signatures, especially for dealing with money transfereing logic.

Where the signature should have a period to be valid in it. And if the deadline passed. The Signature should be marked as invalid at this case.

Recommendations:
add `deadline` parameter and check that the deadline is < block.timestamp

- `CLAIM_TYPEHASH` should be changed to add the deadline param
- structHash will add this input (deadline) when constructing the `structHash`
- And ofc the main check will be made `deadline is < block.timestamp`

Status: Completely Fixed by resolving M-02

4.3 Informational Findings

4.3.1 Incorrect format when constructing hash in `_computeClaimDomainSeparator()`

Description:
When constructing the Domain Separator using `EIP-712` signatures, changing string format to byte format before hashing them is like a standard.

In `_computeClaimDomainSeparator()`, the version slot is not cast into `bytes` format before hashing it.

```
    function _computeClaimDomainSeparator() internal view returns (bytes32) {
        return keccak256(
            abi.encode(
                keccak256("EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)"),
                keccak256(bytes("SplitWithLockup")),
>>              keccak256("1"),
                block.chainid,
                address(this)
            )
        );
    }
```

Recommendations:
Cast the version ("1") string into bytes before hashing it by `keccak256`

```
diff --git a/src/Payments/SplitWithLockup.sol b/src/Payments/SplitWithLockup.sol
index 809ecb5..17cb1fd 100644
--- a/src/Payments/SplitWithLockup.sol
+++ b/src/Payments/SplitWithLockup.sol
@@ -225,7 +225,7 @@ contract SplitWithLockup is Owned, ISplitWithLockup {
             abi.encode(
                 keccak256("EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)"),
                 keccak256(bytes("SplitWithLockup")),
-                keccak256("1"),
+                keccak256(bytes("1")),
                 block.chainid,
                 address(this)
             )
```

Status: Fixed

4.3.2 No check if the depositId is valid or not

Description:
When claiming/reclaiming assets, there is no check for the `depositId` is valid or not. This will not introduce problems in claim as it enforces signature verification. But in reclaim this restriction does not exist.

src/Payments/SplitWithLockup.sol#_reclaim()

```
    function _reclaim(uint depositId) internal {
        Deposit storage _deposit = deposits[deposited];

>>      require(_deposit.state == Status.Deposited,       Errors.ALREADY_CLAIMED);
>>      require(block.timestamp > _deposit.claimDeadline, Errors.STILL_CLAIMABLE);

        _deposit.state = Status.Reclaimed;
        _deposit.token.safeTransfer(_deposit.sender, _deposit.amount);

        emit Reclaimed(depositId, _deposit.sender, _deposit.amount);
    }
```

- If the current deposit count is `10`, and we passed `15` as `depositId`. All the values will be the initial values.
- State check will pass, as Deposited is the first element in enum (default)
- timestamp check will also pass, as the timestamp is > zero (unit default)
- The only thing that will prevent changing the state to `Reclaimed` is the transfer lib by Solmate. Where it checks if the address contains code or not. and address(0) does not contain code, making the tx revert.

There is no security concern regarding this. But accepting any `depositId`, will technically pass all checks. and is not an allowed parameter. having an error message for it is better.

Recommendations:
Add another `require` to check that `depositId` is valid. It should be smaller than `depositCount`.

Status: Fixed