Manifest Finance - Security Review

08.26.2025

Conducted by:

Kann, Lead Security Researcher

Radev_eth, Lead Security Researcher

Table of Contents

1 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

2 Risk classification

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

2.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

2.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3 Executive summary

Overview

Project Name	Manifest Finance
Repository	Private
Commit hash	Private
Resolution	Fixed
Documentation	https://docs.manifest.finance/
Methods	Manual review

Scope

/src/authmanager/AuthManager.sol
/src/authmanager/AuthManagerStorage.sol
/src/chainalysis/MockChainalysisOracle.sol
/src/chainalysis/SanctionsList.sol
/src/chainalysis/SanctionsListStorage.sol
/src/oracle/USHPriceOracle.sol
/src/oracle/USHPriceOracleStorage.sol
/src/sush/AdminControl.sol
/src/sush/Silo.sol
/src/sush/StakedUSH.sol
/src/sush/StakedUSHBase.sol
/src/ush/USAToken.sol
/src/ush/USHToken.sol
/src/ush/USHTokenStorage.sol

Issues Found

Critical risk	1
High risk	1
Medium risk	1
Low risk	2
Informational	1

4 Findings

4.1 Critical

4.1.1 First Deposit Can Result in Zero Shares Due to Direct Token Transfer

Severity: *Critical risk*

Description: The StakedUSH vault is vulnerable to zero share deposits when USH tokens are sent directly to the contract. Because the vault calculates shares based on totalSupply and totalAssets, a direct token transfer increases totalAssets without increasing totalSupply. As a result, legitimate depositor will receive 0 shares for a positive deposit, breaking fair share distribution and potentially locking user funds.

The sUSH vault calculates shares using the formula:

shares = (assets * (totalSupply + decimalsOffset)) / (totalAssets + 1)

If a user directly transfers USH to the vault contract (bypassing deposit()), the vault's totalAssets increases while totalSupply remains 0.

When the next user calls deposit(), the formula becomes:

(assets * (0 + 0)) / (totalAssets + 1) = 0

Thus, the user receives 0 shares (REVERTS), even though they tried depositing a positive amount of assets.

Resolution: Fixed

4.2 High

4.2.1 FULL-Restricted Users Can Still Deposit

Severity: *High risk*

Description: The deposit flow does not fully enforce the FULL_RESTRICTED_STAKER_ROLE check. When a user with FULL restriction calls deposit(assets, receiver), the restriction logic fails to prevent the operation, allowing restricted users to bypass intended compliance rules.

Deposit flow: deposit(ERC4626 logic) -> _deposit(override) where checks if both addresses have SOFT_RESTRICTED_STAKER_ROLE if true reverts then logic goes to _deposit(ERC4626)

```
function _deposit(address caller, address receiver, uint256 assets, uint256 shares)
     internal virtual {

        SafeERC20.safeTransferFrom(IERC20(asset()), caller, address(this), assets);
        _mint(receiver, shares);

        emit Deposit(caller, receiver, assets, shares);
    }
```

Where _mint is called.

```
function _mint(address account, uint256 value) internal {
        if (account == address(0)) {
            revert ERC20InvalidReceiver(address(0));
        }
        _update(address(0), account, value);
    }
```

See how _mint sets first parameter for _update to be address 0.

and since we have _update override it goes to

```
 function _update(address from, address to, uint256 value) internal override {
        if (hasRole(FULL_RESTRICTED_STAKER_ROLE, from) && to != address(0)) {
            revert OperationNotAllowed();
        }
        if (hasRole(FULL_RESTRICTED_STAKER_ROLE, to)) {
            revert OperationNotAllowed();
        }
        super._update(from, to, value);
    }
```

Where it fails to check if the msg.sender is restricted role

Resolution: Fixed

4.3 Medium

4.3.1 _currentDefaultAdmin Not set in SingleAdminAccessControl

Severity: *High risk*

Description: In StakedUshBase.sol, which imports and inherits AdminControl, the constructor calls:

```
_grantRole(DEFAULT_ADMIN_ROLE, _owner);
```

This sets the default admin role to _owner. However, the contract does not initialize the _currentDefaultAdmin variable, leaving it as address(0).

Consequently, in a scenario where the current default admin attempts to transfer the admin role by calling transferAdmin, followed by _pendingDefaultAdmin executing acceptAdmin, the _grantRole override logic executes:

```
_revokeRole(DEFAULT_ADMIN_ROLE, _currentDefaultAdmin);
```

Since _currentDefaultAdmin is address(0), the call to _revokeRole effectively does nothing. The function then proceeds to grant the role to the new admin via:

```
super._grantRole(role, account);
```

The net effect is that both the original _owner and the new admin retain the DEFAULT_ADMIN_ROLE, resulting in two admins instead of a proper transfer.

This issue arises from the lack of proper initialization of _currentDefaultAdmin during contract deployment and highlights a gap in the role transfer logic that lead to unintended multi-admin privileges.

Resolution: Fixed

4.4 Low

4.4.1 Pause bypass for approvals via permit

Severity: *Low risk*

Description: USHToken gates approve() with whenNotPaused, but inherits permit() from Solmate without overriding. While paused, anyone can still set/refresh allowances via permit, pre-arming drains that execute the instant the token is unpaused.

Resolution: Fixed

4.4.2 Burner role blocked by Auth Manager checks

Severity: *Low risk*

Description:

USHToken.burn(from, amount) calls _checkAuthTransfer(from, address(0)). If `from` has since become banned/sanctioned (via `AuthManager`), `checkSanctioned`/`checkBanned` will revert. This prevents the protocol (holder of `BURNER_ROLE`) from programmatically burning balances from blocked accounts a common compliance/control action.

Resolution: Fixed

4.5 Informational

4.5.1 chainalysisOracleEnabled() function logic inverted

Severity: *Informational*

Description: The view function chainalysissOracleEnabled() returns true when the oracle is unset (i.e., disabled), and false when the oracle is actually configured. This behavior contradicts the function name, which suggests it should return true when the oracle is enabled.

Recommendation

Fix logic:

```
function chainalysisOracleEnabled() public view returns (bool) {
    return KycManagerStorageLib.getV1().chainalysisOracle != address(0);
}
```

Resolution: Fixed