SHERLOCK

Security Review For Aegis

Collaborative Audit Prepared For:	**Aegis**
Lead Security Expert(s):	**0x73696d616f**
Date Audited:	**April 22 - April 23, 2025**
Final Commit:	**5add3ff**

Introduction

A comprehensive security review of the Staked YUSD contract suite, covering staking mechanics, reward distribution, and integration with Aegis's Bitcoin□backed YUSD stablecoin.

Scope

Repository: Loggy/aegis-contracts

Audited Commit: c65bce4dba50cb1064db0dffef514cc9616abe88

Final Commit: 5add3ff520a8da5a647232bdc487a635213caa5e

Files:

- contracts/sYUSD.sol

Final Commit Hash

5add3ff520a8da5a647232bdc487a635213caa5e

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
1	0	2

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Attacker can DoS user withdrawals at no cost

Source: https://github.com/sherlock-audit/2025-04-aegis-staked-yusd/issues/2

Summary

Everytime a user deposits or mints, a lockup element is added to an array. An attacker can deposit on behalf of other user and fill the array with excessive elements such that it is DoSed due to OOG.

Vulnerability Detail

`sYUSD::deposit()` is as follows:

```
function _deposit(
    address caller,
    address receiver,
    uint256 assets,
    uint256 shares
) internal virtual override {
    super._deposit(caller, receiver, assets, shares);

    // Add new locked shares entry
    userLockedShares[receiver].push(LockedShares({
        amount: shares,
        expiryTimestamp: block.timestamp + lockupPeriod
    }));
}
```

As can be seen, it pushes a new element each time it is called, via `sYUSD::deposit()` or `sYUSD::mint()`. An attacker can deposit to another user and fill their array. Then, when the user withdraws via `sYUSD::withdraw()` or `sYUSD::redeem()`, it loops through the array all at once and DoSes withdrawals due to OOG.

```
function _withdraw(
    address caller,
    address receiver,
    address owner,
    uint256 assets,
    uint256 shares
) internal virtual override {
    // First update the unlocked shares
    updateUnlockedShares(owner);

    // Ensure user has enough unlocked shares
```

```
    if (unlockedShares[owner] < shares) {
        revert InsufficientUnlockedShares(shares, unlockedShares[owner]);
    }

    // Reduce unlocked shares
    unlockedShares[owner] -= shares;

    super._withdraw(caller, receiver, owner, assets, shares);
}

...

function updateUnlockedShares(address user) public {
    LockedShares[] storage userLocks = userLockedShares[user];

    for (uint256 i = 0; i < userLocks.length; i++) {
        if (block.timestamp >= userLocks[i].expiryTimestamp && userLocks[i].amount
↪   > 0) {
            unlockedShares[user] += userLocks[i].amount;
            userLocks[i].amount = 0;
        }
    }

    // Clean up empty entries
    _cleanupLockedShares(user);
}
```

Please find a POC here.

Impact

Stuck funds at no cost for an attacker.

Code Snippet

https://github.com/sherlock-audit/2025-04-aegis-staked-yusd/blob/main/aegis-contracts/contracts/sYUSD.sol#L80-L92

Tool Used

Manual Review

Recommendation

Set a max iterations argument.

Discussion

Loggy

https://github.com/Loggy/aegis-contracts/commit/f48e7163167b569bc07c0c302ab71343590f2ef8

commit with fixes

Issue L-1: Missing maxRedeem() implementation

Source: https://github.com/sherlock-audit/2025-04-aegis-staked-yusd/issues/3

Summary

`ERC4626` implements a `maxRedeem()` function that must specify the max amount of shares a user can redeem. Due to the locked shares property, a user can't withdraw all their shares at any time, but the `maxRedeem()` still returns their balance (inherited).

Vulnerability Detail

When we look at `sYUSD::maxWithdraw()`, it correctly accounts for the locked shares:

```
function maxWithdraw(address owner) public view virtual override returns (uint256) {
    // Calculate current unlocked amount (without state changes)
    uint256 currentUnlocked = unlockedShares[owner];
    LockedShares[] storage userLocks = userLockedShares[owner];

    for (uint256 i = 0; i < userLocks.length; i++) {
        if (block.timestamp >= userLocks[i].expiryTimestamp) {
            currentUnlocked += userLocks[i].amount;
        }
    }

    // Convert unlocked shares to assets
    return convertToAssets(currentUnlocked);
}
```

However, this is not true for `maxRedeem()`, which returns the inherited unchanged `balanceOf(user)` from `ERC4626`.

Impact

Specification mismatch.

Code Snippet

https://github.com/sherlock-audit/2025-04-aegis-staked-yusd/blob/foundry-tests/aegis-contracts/contracts/sYUSD.sol#L17

Tool Used

Manual Review

Recommendation

Implement `maxRedeem()`, for example:

```
function maxRedeem(address owner) public view virtual override returns (uint256) {
    // Calculate current unlocked amount (without state changes)
    uint256 currentUnlocked = unlockedShares[owner];
    LockedShares[] storage userLocks = userLockedShares[owner];

    for (uint256 i = 0; i < userLocks.length; i++) {
        if (block.timestamp >= userLocks[i].expiryTimestamp) {
            currentUnlocked += userLocks[i].amount;
        }
    }

    return currentUnlocked;
}
```

Discussion

Loggy

https://github.com/Loggy/aegis-contracts/commit/f48e7163167b569bc07c0c302ab71343590f2ef8

Commit with fixes

Issue L-2: Donation attack possible, although unlikely, could make an initial deposit

Source: https://github.com/sherlock-audit/2025-04-aegis-staked-yusd/issues/4

Summary

ERC4626 donations attack are mitigated by default in the current Openzeppelin implementation by setting the decimals offset to 0, which is enough to dillute attacker donations as per the assets/shares conversion, here. However, this means the attack may not always be profitable for the attacker, but there is still a chance it is, and the users take losses.

Vulnerability Detail

Essentially the attack is as follows:

1. Attacker mints 1 shares.
2. Attacker donates `YUSD` to `sYUSD`.
3. User deposits and mints 0 or a rounded down amount of shares (specifying that shares minted > 0 is not enough because if it rounds down to 1, 2 or similar the user also takes losses).

This is mitigated by the current ERC4626 implementation as it results in:

1. Attacker mints 1 shares.
2. Attacker donates 10e18 assets + 1.
3. If the attacker redeems their share, they get `1 * (1e18 + 1 + 1) / (1 + 1) = 0.5e18`.

However, a user may still get caught in this: Step 4, user deposits 4e18 assets, getting `4e18 * (1 + 1) / (10e18 + 1) = 0` shares.

Impact

Attacker steals user funds

Code Snippet

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC4626.sol#L225-L227

Tool Used

Manual Review

Recommendation

Make an initial, small deposit and keep it deposited to fully prevent these attacks.

Discussion

Loggy

https://github.com/Loggy/aegis-contracts/commit/a54bf59d86344cea9f2ca0c68c0cbe406bf8c54b

added min deposit to deploy script to minimize time for potential attack

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.