yAudit Asymmetry veASF Review

Review Resources:

- Code repository

Auditors:

- HHK
- Panda

Table of Contents

Review Summary

asymmetry veASF

Asymmetry veASF provides token locking and fee distributor contracts for the ASF token.

The contracts of the asymmetry veASF Repo were reviewed over 5 days. The code review was performed by 2 auditors between 27 May and 31 May 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit 969dd707ac096737e69061fda84951c8518d2601 for the veASF repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src
├── AsfToken.sol
├── FeeDistributor.sol
├── TokenLocker.sol
├── interfaces
│   └── ITokenLocker.sol
└── utils
    ├── DelegatedOps.sol
    └── SystemStart.sol
```

After the findings were presented to the asymmetry team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, asymmetry veASF and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Proper access control mechanisms ensure only authorized accounts can execute privileged functions.
Mathematics	Good	The mathematical operations are implemented correctly, ensuring accuracy and preventing overflow/underflow errors.
Complexity	Average	The uint256 manipulations and bitwise operations add complexity that requires careful handling and understanding.
Libraries	Good	Appropriate libraries are used, and external dependencies are managed effectively.
Decentralization	Good	The contracts promote decentralization by minimizing the need for centralized control and ensuring fair governance mechanisms.
Code stability	Good	The code is stable

Category	Mark	Description
Documentation	Good	The code is well-documented, providing clear explanations of the functionality and usage of various components.
Monitoring	Good	Adequate monitoring mechanisms are in place.
Testing and verification	Average	While there is a reasonable level of testing, additional tests could be implemented to cover edge cases and improve overall coverage.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

Gas Saving Findings

1. Gas - Use unsafe math

Unsafe math can be used to increase gas efficiency.

Technical Details

```
File: src/TokenLocker.sol

517 | uint256 increase = (_newWeeks - _weeks) * _amount;

655 | increasedWeight += (newWeeks - oldWeeks) * amount;

822 | accountData.unlocked = uint32((unlocked - amountToWithdraw) / lockToTokenRatio);
```

src/TokenLocker.sol#L517, src/TokenLocker.sol#L655, src/TokenLocker.sol#L822

```
File: src/FeeDistributor.sol

122 | uint256 amount = (weeklyFeeAmounts[token][i] * weight) / totalWeight;

135 | uint256 amount = (weeklyFeeAmounts[token][i] * weight) / totalWeight;

284 | claimableAmount += (feeAmount * weight) / totalWeight;
```

src/FeeDistributor.sol#L122, src/FeeDistributor.sol#L135, src/FeeDistributor.sol#L284

Impact

Gas savings.

Recommendation

Use `unchecked {}` for subtraction and divisions.

Developer Response

Will not implement.

2. Gas - Consider using a right-shift operator instead of a division

`<x> / 256` can be replaced with `<x> >> 8` for equivalent results but a lower gas cost.

Technical Details

There are a lot of divisions by 265 in the codebase. 256 is equal to 2**8; the following change could, for example, be made:

```
File: src/TokenLocker.sol


- 182 | bitfield = accountData.updateWeeks[accountWeek / 256];
+ 182 | bitfield = accountData.updateWeeks[accountWeek >> 8];
```

Impact

Gas savings.

Recommendation

Use bit-shifting instead of divisions.

Developer Response

implemented at https://github.com/asymmetryfinance/veASF/pull/3/commits/535755b1876b2d1f560b71d5635af1c8c20b010f

3. Gas - Cache `currentWeek` in `withdrawWithPenalty()`

Technical Details

In the function `withdrawWithPenalty()` the function `getWeek()` is called twice.

Caching the first call in a `currentWeek` variable and using it at the end instead of calling `getWeek()` again would save some gas.

Impact

Gas.

Recommendation

Save roughly 100 gas by caching `getWeek()` instead of calling it twice.

L833:

```
uint256 currentWeek = getWeek();
uint256 systemWeek = currentWeek;
```

L890:

```
accountWeeklyWeights[msg.sender][currentWeek] = uint40(weight - decreasedWeight);
totalWeeklyWeights[currentWeek] = uint40(getTotalWeightWrite() - decreasedWeight);
```

Developer Response

Implemented at https://github.com/asymmetryfinance/veASF/pull/3/commits/52c9fb7cde83323b35ec872cf0ffa89101217112

Informational Findings

1. Informational - if statement can be converted to a ternary

The code can be made more compact while also increasing readability by converting the following `if`-statements to ternaries (e.g., `foo += (x > y) ? a : b`)

Technical Details

File: src/TokenLocker.sol

```
181 | if (accountWeek % 256 == 0) {
182 |                     bitfield = accountData.updateWeeks[accountWeek / 256];
183 |                 } else {
184 |                     bitfield = bitfield >> 1;
185 |                 }

226 | if (accountWeek % 256 == 0) {
227 |                 bitfield = accountData.updateWeeks[accountWeek / 256];
228 |             } else {
229 |                 bitfield = bitfield >> 1;
230 |             }

270 | if (currentWeek % 256 == 0) {
271 |                     bitfield = accountData.updateWeeks[currentWeek / 256];
272 |                 } else {
273 |                     bitfield = bitfield >> 1;
274 |                 }

938 | if (accountWeek % 256 == 0) {
939 |                 bitfield = accountData.updateWeeks[accountWeek / 256];
940 |             } else {
941 |                 bitfield = bitfield >> 1;
942 |             }
```

src/TokenLocker.sol#L181, src/TokenLocker.sol#L226, src/TokenLocker.sol#L270, src/TokenLocker.sol#L938

Impact

Informational

Recommendation

Use ternary to improve code.

Developer Response

implemented at https://github.com/asymmetryfinance/veASF/pull/3/commits/98df937fefb74f1da230c48c195d76921e2a3863

2. Informational - Avoid redundant return statements

When a function includes a named return variable, a return statement becomes redundant and should not be included.

Technical Details

```
File: src/FeeDistributor.sol

92 | return amounts;

224 | return claimedAmounts;

268 | return claimedAmounts;

286 | return claimableAmount;
```

src/FeeDistributor.sol#L92, src/FeeDistributor.sol#L224, src/FeeDistributor.sol#L268, src/FeeDistributor.sol#L286

```
File: src/TokenLocker.sol

194 | return (locked, unlocked);

286 | return (lockData, frozenAmount);

361 | return (amountToWithdraw, penaltyTotal);

958 | return weight;
```

src/TokenLocker.sol#L194, src/TokenLocker.sol#L286, src/TokenLocker.sol#L361, src/TokenLocker.sol#L958

Impact

Informational

Recommendation

Implemented some at https://github.com/asymmetryfinance/veASF/pull/3/commits/b257342ba2067929cf3cf6138a4f4b3c429fd52b

3. Informational - Missing check for address(0x0) when adding a new token to FeeDistributor

`registerNewFeeToken()` should not accept the address zero as a parameter.

Technical Details

```
File: src/FeeDistributor.sol

155 | feeTokenData[token] = FeeTokenData({ isRegistered: true, firstWeek:
uint16(getWeek()), lastWeek: 0 });
```

src/FeeDistributor.sol#L155

Impact

Infoamtional

Recommendation

Make sure the token is not the zero address.

Developer Response

Implemented at https://github.com/asymmetryfinance/veASF/pull/3/commits/cc1a98dfadb73051b3d004b01379fd5e57ed91e6

4. Informational - Make sure `feeReceiver` can't be set to zero

The ASF token implementation will revert if a token transfer is made to the zero address. The constructor and setFeeReceiver function are not preventing the `feeReceiver` from being set to the zero address. Setting it to the zero address will break `withdrawWithPenalty()`.

Technical Details

```
153 |    function setFeeReceiver(address _receiver) external onlyOwner returns (bool) {
154 |        feeReceiver = _receiver;
155 |        return true;
156 |    }
```

TokenLocker.sol#L153-L156

Impact

Informational

Recommendation

Add a check to ensure the admin isn't setting `feeReceiver` to the zero address.

Developer Response

Implemented at https://github.com/asymmetryfinance/veASF/pull/3/commits/89dc09dce62cacc8b9d74bb380892947926457e4

5. Informational - Enforce `startTime * 1 weeks == 0` in `SystemStart`

Technical Details

In the contract `SystemStart`, the `startTime` is supposed to be a timestamp on Thursday at 00:00:00 GMT.

This is because the UNIX timestamp starts at this time this way the `TokenLocker` contract can enforce locks to be 2 weeks when locking in the last 3 days of the week.

It is crucial that the timestamp passed is correct, as it could otherwise allow users to lock for one week less than three days before the end of the week.

Impact

Informational.

Recommendation

Add a check `startTime * 1 weeks == 0` in the constructor and revert if `false`.

Developer Response

Implemented at https://github.com/asymmetryfinance/veASF/pull/3/commits/832aba854a15287ef3459944acf8c3a807976d06

6. Informational - Users may be locked longer than expected

Technical Details

When a user lock his tokens for 1 week during the last 3 days of the epoch his lock is directly set to 2 weeks instead in the function `_lock()`.

This results in users expecting to be locked for 3 days or less but being locked for 7 to 10 days.

Impact

Informational.

Recommendation

Consider reverting when a user shouldn't be able to lock for 1 week instead of setting the lock to 2 weeks.

Developer Response

Will not implement

Final remarks

In conclusion, the audit of the asymmetry veASF contracts identified several areas for improvement. The changes made to the original codebase from which the project was forked didn't result in any important vulnerabilities. The recommendations primarily focused on enhancing gas efficiency, improving code readability, and adding additional safeguards to prevent potential issues.