



Epoch Security Assessment

Dec 8, 2023

Prepared for

Epoch

Prepared by:

Oxfoobar

Table of Contents

Table of Contents	2
Summary	4
Overview	4
Project Scope	4
Severity Classification	4
Summary of Findings	4
Disclaimer	6
Key Findings and Recommendations	7
1. Fee-on-transfer or other malicious tokens can break accounting	7
Description	7
Impact	7
Recommendation	7
Response	8
2. Users can get more than their share of rewards by waiting to withdraw	9
Description	9
Impact	9
Recommendation	9
Response	9
3. Onchain decimals() query is unnecessary	10
Description	10
Impact	10
Recommendation	10
Response	10
4. Array tracking all LP Positions can become stale and sparse	11
Description	11
Impact	11
Recommendation	11
Response	11
5. Unify max protocol fee value, declare hardcoded values as constants	12
Description	12
Impact	12
Recommendation	12
Response	12
6. Use named mappings	13
Description	13
Impact	13
Recommendation	13
Response	13

7. Fix Natspec	14
Description	14
Impact	14
Recommendation	14
Response	14
8. Use latest compiler version and associated features	16
Description	16
Impact	16
Recommendation	16
Response	16
9. Miscellaneous gas optimizations and comment fixes	17
Description	17
Response	17

Summary

Overview

Epoch is an ERC20 token offering vote escrowed locks and upside/downside exposure trading via its ITOProtocol.

Project Scope

We reviewed the core smart contracts and test suite contained within commit hash `83cdd638b6a923cb726fa453e86d0eb38fcb0bc6` on branch `develop` at <https://github.com/Moai-Labs/vepoch-contracts>, and commit hash `5ce8b7566de7ab4282fcff652230ea646a7495c6` on branch `master` at <https://github.com/Moai-Labs/upside-contracts>. Fixes were reviewed at commit hash `e3c002298ae2af68b7bcb647a87a610c1a8c6e2c` in vepoch-contracts and `9a0c5310c53a6c424749d215ce7c2de4079cc96b` in upside-contracts.

Severity Classification

High - Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).

Medium - Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.

Low - Assets are not at risk: state handling, function incorrect as to spec, issues with comments.

Informational - Code style, clarity, syntax, versioning, off-chain monitoring (events, etc)

Summary of Findings

Severity	Findings	Resolved
High	1	1
Medium	2	1
Low	2	1
Informational	4	3

Disclaimer

This security assessment should not be used as investment advice.

We do not provide any guarantees on eliminating all possible security issues. foostudio recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

We did not assess the following areas that were outside the scope of this engagement:

- Website frontend components
- Offchain order management infrastructure
- Multisig or EOA private key custody
- Metadata generation

Key Findings and Recommendations

1. Fee-on-transfer or other malicious tokens can break accounting

Severity: **High**

Files: ITOProtocolV1.sol

Description

The supply() method makes optimistic transfer calls to deposit potentially untrusted ERC20s into the protocol. This could break on nonstandard ERC20s such as fee-on-transfer tokens.

Impact

Broken accounting and potentially bricked withdrawals for specific LP positions.

Recommendation

Best practice is to check balance before and after making the transfer, and update the actual upsideTokenBalance based on the differential here, like so:

```
uint256 prevBalance = IERC20(_upsideToken).balanceOf(address(this));
IERC20(_upsideToken).safeTransferFrom(msg.sender, address(this),
_upsideTokenAmount);
_upsideTokenAmount = IERC20(_upsideToken).balanceOf(address(this)) -
prevBalance;
lpPositions.push(LPPosition(
    msg.sender,
    _feeBp,
    _startDate,
    _endDate,
    IERC20(_downsideToken),
    IERC20(_upsideToken),
    _upsideTokenAmount,
    _exchangeRate,
    0
));
```

Response
Done.

2. Users can get more than their share of rewards by waiting to withdraw

Severity: **Medium**

Files: Vepoch.sol

Description

Users can claim more than their share of rewards by simply not withdrawing once the lock expires. `calculateRewards()` has no notion of lock expiry time, only current balance. So waiting to poke this will increase what users are eligible for. More generally, the math should ensure that users get no benefits once the lock has expired.

Impact

Value leakage to non-locked users.

Recommendation

Ensure user benefits stop once the lock expires.

Response

Getting continuous rewards after your lock duration has completed is intentional game theory design. These individuals are still providing liquidity, therefore adding economic value. If they unlock liquidity after lock complete, they will need to start over completely, so it promotes loyalty.

3. Onchain decimals() query is unnecessary

Severity: **Medium**

Files: ITOProtocolV1.sol

Description

The take() and untake() methods query downsideToken.decimals(), to counteract potential mismatches in precision between upside and downside tokens. However decimals should be primarily as an offchain display feature rather than incorporated into onchain math. For example, when a user enters to swap “100 USDC” on the Uniswap frontend, the Uniswap router and pool do not query USDC decimals. Rather, the Uniswap frontend calculates decimals offchain and converts “100 USDC” into a 100e6 uint256 that can then be operated on by Solidity. The same should apply here - decimals being used solely as frontend preprocessing, then the smart contracts operate directly on raw uint256 amount with no scaling needed.

Impact

Increased gas, complexity, and potential misparameterization of LP positions.

Recommendation

Operate solely on raw uint256 amounts without worrying about decimals.

Response

Done.

4. Array tracking all LP Positions can become stale and sparse

Severity: **Low**

Files: ITOProtocolV1.sol

Description

All LP Positions are tracked in a single array. While no specific DDoS vector is present in the codebase, the `lpPositionsCount()` method is potentially misleading because previous array elements can be deleted entirely once they've been fulfilled.

Impact

Misleading length.

Recommendation

It's more common to see a mapping(`uint256 => LPPosition`) paired with an autoincrementing `uint256` to mark the next used id key. This way deletions can be processed fully without leaving unexpected empty space in the array.

Response

Acknowledged, no changes will be implemented.

5. Unify max protocol fee value, declare hardcoded values as constants

Severity: **Low**

Files: ITOProtocolV1.sol

Description

Fee denominator is 10_000, this value is hardcoded in several methods but should be declared as a constant to ensure no extraneous or missing zeroes slip by anywhere. The protocolFeeMaxBp is 7_000 (70%) in setProtocolFee() but 7_500 (75%) in supply(). These values should be unified into one correct value and used as a constant to avoid error.

Impact

Incorrect parameterization.

Recommendation

Use constant declaration to ensure no typos.

Response

Done.


6. Use named mappings

Severity: **Informational**

Files: *.sol

Description

Solidity 0.8.18 and later support a language feature called [named mappings](#), where keys and values can be named to give devs better documentation of what's expected in each.



```
pragma solidity >=0.8.18;

contract Foo {
    mapping(address key => uint256 value) public items;
    mapping(string name => uint256 balance) public users;
}
```

Impact

Extraneous comments that could be inlined.

Recommendation

Experiment with named mappings where the storage layout otherwise requires comment explanations.

Response

None.

7. Fix Natspec

Severity: **Informational**

Files: *.sol

Description

The first word of Natspec return comments should be the name of the returned variable, instead of the first word of the description of it. Three leading slashes should be used for `@notice` and `@dev` natspec comments. Ensure all function parameters are documented in addition to methods.

Impact

Cleaner autogenerated docs.

Recommendation

Update Natspec.

Response

Fixed.

8. Use latest compiler version and associated features

Severity: **Informational**

Files: pool.sol

Description

All files currently have the solidity pragma at 0.8.17. Best practice is to target the latest 0.8.23 version. This will have three benefits: cheaper deployment costs, gas optimization and new language features. This protocol is targeting Ethereum, so there are no PUSH0 incompatibility issues. However if multichain expansion is a plan in the future, it'll be important to explicitly set the compiler's target EVM version as "paris" rather than "shanghai" to avoid PUSH0 opcode incompatibility.

Impact

Increased deployment size and gas costs.

Recommendation

Use a floating ^0.8.23 pragma everywhere, and target the paris EVM version with `evm_version = "paris"` in foundry.toml

Response

Fixed.

9. Miscellaneous gas optimizations and comment fixes

Severity: **Informational**

Files: Vepoch.sol

Description

- rewardToken can be made immutable
- calculateVeTokens comment should refer to 365 days instead of 365.24
- min lock time should be at least an hour rather than a minute

Response

Fixed.