MiloTruck

EPOCH Island (ITO)

Security Review

December 12, 2023

Contents

1 Introduction

1.1 About MiloTruck

MiloTruck is an independent security researcher who specializes in smart contract audits. Currently, he works as a Senior Auditor at Trust Security and Associate Security Researcher at Spearbit. He is also one of the top wardens on Code4rena.

For private audits or security consulting, please reach out to him on:

- Twitter - *@milotruck*
- Discord - *@milotruck*

You can also request a quote on Code4rena or Cantina to engage them as an intermediary.

1.2 Disclaimer

A smart contract security review **can never prove the complete absence of vulnerabilities**. Security reviews are a time, resource and expertise bound effort to find as many vulnerabilities as possible. However, they cannot guarantee the absolute security of the protocol in any way.

2 Risk Classification

Severity Level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2.1 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/availability.
- Low - Funds are **not** at risk.

2.2 Likelihood

- High - Highly likely to occur.
- Medium - Might occur under specific conditions.
- Low - Unlikely to occur.

3 Executive Summary

3.1 About EPOCH Island

Epoch Island aims to become a Network State for crypto builders.

This codebase consists of one contract to facilitate the protocol's initial time offering for their native token, EPOCH.

3.2 Overview

Project Name	Epoch Island (ITO)
Project Type	Swaps
Language	Solidity
Repository	upside-contracts
Commit Hash	9a0c5310c53a6c424749d215ce7c2de4079cc96b

3.3 Scope of Work

contracts/ITOProtocolV1.sol (187 SLOC)

3.4 Issues Found

High	0
Medium	2
Low	2
Informational	0
Gas Optimization	1

4 Findings

4.1 Medium Risk

4.1.1 `protocolFeeBp` and `d.lpFeeBp` can exceed 100%

Context:

- ITOProtocolV1.sol#L34
- ITOProtocolV1.sol#L245-L247
- ITOProtocolV1.sol#L81-L83
- ITOProtocolV1.sol#L189
- ITOProtocolV1.sol#L196

Description: `MAX_FEE_BP`, which is the maximum value for all fees, is set to 75%:

```
uint16 constant MAX_FEE_BP = 7500;
```

Both `protocolFeeBp` and `lpFeeBp` are checked to be below `MAX_FEE_BP` in `setProtocolFee()` and `supply()` respectively:

```
if(_protocolFeeBp > MAX_FEE_BP) {
    revert FeeTooHigh();
}
```

```
if(_feeBp > MAX_FEE_BP) {
    revert FeeTooHigh();
}
```

However, the contract does not check that the protocol and LP fee combined are less than 100%. In fact, `protocolFeeBp + lpFeeBp` can be up to 150%.

This could create positions where `take()` cannot be called. If `protocolFeeBp + d.lpFeeBp` is larger than 100%, the following line will revert with an arithmetic underflow as `protocolFee + lpFee > _maxDownsideTokenIn`:

```
uint256 _downsideTokenBalanceAfterFees = _maxDownsideTokenIn - protocolFee - lpFee;
```

Alternatively, if `protocolFee + lpFee` is exactly 100%, the following line will revert as `BP_MULTIPLIER == d.lpFeeBp + protocolFeeBp`, resulting in a division by zero error:

```
_maxDownsideTokenIn = (_downsideTokenBalanceAfterFees * BP_MULTIPLIER) / (BP_MULTIPLIER - (d.lpFeeBp +
↪  protocolFeeBp));
```

There are two scenarios where `protocolFeeBp + d.lpFeeBp` might end up becoming 100% or more:

- A position is created where both fees add up to 100% or more. For example:
  - `protocolFeeBp` is set to 25%, and `supply()` is called to create a position with `lpFeeBp` as 75%.
- `protocolFeeBp` is increased by the owner. For example:
  - A current position has `lpFeeBp` set to 50%.
  - The owner calls `setProtocolFee()` to increase `protocolFeeBp` from 20% to 60%.
  - This will cause `take()` to always revert when called for the position.

Recommendation: Consider decreasing `MAX_FEE_BP` to `4999` instead. By making the maximum limit for both fees slightly less than 50%, `protocolFeeBp + d.lpFeeBp` will never be equal to or larger than 100%.

EPOCH Island: Fixed in commit `1dc6be7` by setting `MAX_FEE_BP` to `4999`.

4.1.2 Support for fee-on-transfer tokens is inconsistent

Context:

- ITOProtocolV1.sol#L88-L90
- ITOProtocolV1.sol#L211
- ITOProtocolV1.sol#L234

Description: Support for fee-on-transfer tokens was added to `supply()`:

```
uint256 previousBalance = IERC20Metadata(_upsideToken).balanceOf(address(this));
IERC20Metadata(_upsideToken).safeTransferFrom(msg.sender, address(this), _upsideTokenAmount);
_upsideTokenAmount = IERC20Metadata(_upsideToken).balanceOf(address(this)) - previousBalance;
```

However, other functions in the contract are still not compatible with fee-on-transfer tokens. Most notably, when transferring downside tokens into the contract in `take()`:

```
d.downsideToken.safeTransferFrom(msg.sender, address(this), _maxDownsideTokenIn);
```

Additionally, when transferring upside tokens into the contract in `untake()`:

```
d.upsideToken.safeTransferFrom(msg.sender, address(this), upsideTokenAmount);
```

Recommendation: Consider refactoring `take()` and `untake()` to support fee-on-transfer tokens by checking if the amount received is sufficient. For example, in `take()`:

```
uint256 balanceBefore = d.downsideToken.balanceOf(address(this));
d.downsideToken.safeTransferFrom(msg.sender, address(this), _maxDownsideTokenIn);
uint256 amountReceived = d.downsideToken.balanceOf(address(this)) - balanceBefore;
if (balanceBefore < _maxDownsideTokenIn) revert InsufficientTokensTransferred();
```

The same pattern can be used for `untake()` as well.

Alternatively, consider not supporting fee-on-transfer tokens if it isn't required.

EPOCH Island: Fixed in commit `52e6377` by removing support for fee-on-transfer tokens.

4.2 Low Risk

4.2.1 Ensure `protocolFeeBp` is zero when `protocolFeeRecipientAddress` is the zero address

Context:

- ITOProtocolV1.sol#L244-L250

Description: `setProtocolFee()` allows the owner to set `protocolFeeBp` and `protocolFeeRecipientAddress`:

```
function setProtocolFee(address _protocolFeeRecipientAddress, uint16 _protocolFeeBp) external onlyOwner
↪ {
    if(_protocolFeeBp > MAX_FEE_BP) {
        revert FeeTooHigh();
    }

    protocolFeeRecipientAddress = _protocolFeeRecipientAddress;
    protocolFeeBp = _protocolFeeBp;
```

However, there is no check to ensure that `_protocolFeeBp` is zero when `_protocolFeeRecipientAddress` is the zero address.

Recommendation: Consider adding the following check in `setProtocolFee()`:

```
  function setProtocolFee(address _protocolFeeRecipientAddress, uint16 _protocolFeeBp) external
↪  onlyOwner {
      if(_protocolFeeBp > MAX_FEE_BP) {
          revert FeeTooHigh();
      }

+     if(_protocolFeeRecipientAddress == address(0) && _protocolFeeBp != 0) {
+         revert InvalidFeeConfiguration();
+     }
```

This ensures that the protocol fee cannot be non-zero while the fee recipient is an invalid address, which prevents protocol fees from accruing to no one.

EPOCH Island: Fixed in commit `1dc6be7` as recommended.

4.2.2 Avoid initializing `protocolFeeBp` on deployment

Context:

- ITOProtocolV1.sol#L31-L32

Description: When the contract is first deployed, `protocolFeeBp` is set to 1%, while `protocolFeeRecipientAddress` is `address(0)`.

```
uint16 public protocolFeeBp = 100;
address public protocolFeeRecipientAddress;
```

Note that `protocolFeeRecipientAddress` is not set in the constructor.

If the owner does not call `setProtocolFee()` to set the fee recipient, fees from `take()` will accrue to the zero address.

Recommendation: Consider not initializing `protocolFeeBp` to `100` on deployment:

```
- uint16 public protocolFeeBp = 100;
+ uint16 public protocolFeeBp;
  address public protocolFeeRecipientAddress;
```

This forces the owner to call `setProtocolFee()` to activate the protocol fee, which ensures that `protocolFeeRecipientAddress` will be set to a valid address.

The alternative would be to set `protocolFeeRecipientAddress` in the constructor.

EPOCH Island: Fixed in commit `1dc6be7` by initializing `protocolFeeRecipientAddress` in the constructor.

4.3 Gas Optimization

4.3.1 Gas optimizations for `computeFee()`

`computeFee()` can be optimized to use less gas by:

- Avoid copying the whole `LPPosition` struct from storage into memory.
- Cache `protocolFeeBp` and `d.lpFeeBp` to avoid reading from storage multiple times.

```
function computeFee(uint256 _positionId, uint256 _downsideTokenAmount) public view returns(
    uint256 protocolFeeTokens,
    uint256 lpFeeTokens
) {
    uint256 _protocolFeeBp = protocolFeeBp;
    if(_protocolFeeBp != 0) {
        protocolFeeTokens = (_downsideTokenAmount * _protocolFeeBp) / BP_MULTIPLIER;
    }

    uint256 _lpFeeBp = lpPositions[_positionId].lpFeeBp;
    if(_lpFeeBp != 0) {
        lpFeeTokens = (_downsideTokenAmount * _lpFeeBp) / BP_MULTIPLIER;
    }

    return (protocolFeeTokens, lpFeeTokens);
}
```

EPOCH Island: Fixed in commit `1dc6be7` as recommended.