Epoch Island (ITOProtocolV1) Security Review

Reviewer
Hans

December 26, 2023

Contents

1 Executive Summary

As a security advisor of Epoch Island, Hans reviewed upside-contracts.

Summary

Type of Project	Liquidity Pool
Timeline	10st Dec, 2023 - 14th Dec, 2023
Methods	Manual Review

A comprehensive security review identified a total of 4 issues.

Repository	**Initial Commit**
upside-contracts	9a0c5310c53a6c424749d215ce7c2de4079cc96b

Total Issues

High Risk	0
Medium Risk	2
Low Risk	1
Informational	1
Gas Optimization	0

The reported vulnerabilities were addressed by the team, and the mitigation underwent a review process and was verified by Hans.

Repository	**Final Commit**
Epoch Island	da1ea725df1aed0f5cdba939b62ad46317c93b08

2 Scope of the Audit

This audit was conducted for a single contract in `contracts/ITOProtocolV1.sol`.

3 About Hans

Hans is an esteemed security analyst in the realm of smart contracts, boasting a firm grounding in mathematics that has sharpened his logical abilities and critical thinking skills. These attributes have fast-tracked his journey to the peak of the Code4rena leaderboard, marking him as the number one auditor in a record span of time. In addition to his auditor role, he also serves as a judge on the same platform. Hans' innovative insight is evident in his creation of Solodit, a vital resource for navigating consolidated security reports. In addition, he is a co-founder of Cyfrin, where he is dedicated to enhancing the security of the blockchain ecosystem through continuous efforts.

4 Disclaimer

I endeavor to meticulously identify as many vulnerabilities as possible within the designated time frame; however, I must emphasize that I cannot accept liability for any findings that are not explicitly documented herein. It is essential to note that my security audit should not be construed as an endorsement of the underlying business or product. The audit was conducted within a specified timeframe, with a sole focus on evaluating the security aspects of the solidity implementation of the contracts.

While I have exerted utmost effort in this process, I must stress that I cannot guarantee absolute security. It is a well-recognized fact that no system can be deemed completely impervious to vulnerabilities, regardless of the level of scrutiny applied.

5 Protocol Summary

Epoch Island is a community-owned economy created to fuel and fund crypto builders, kind of a decentralized Silicon Valley. `ITOProtocolV1` (previously `upside`) is designed to be a liquidity pool where LPs can create positions by depositing so-called "upside tokens". Users can deposit their "downside tokens" and gets the upside tokens after paying a fee. While the protocol assumed `EPOCH` token as an upside token and `USDC` as a downside token, it is designed to be flexible to support any ERC20 tokens.

6 Additional Comments

The security assessment was carried out with a narrow focus on the contracts. Due to time limitations and the incremental nature of the reviews, the results might not be comprehensive and might not represent the complete security profile of the protocol. The audit was conducted on the contracts at commit 9a0c53.

7 Findings

7.1 Medium Risk Findings

7.1.1 Accounting can be off by a few WEIs due to rounding

Severity: Medium

Context: https://github.com/Moai-Labs/upside-contracts/blob/9a0c5310c53a6c424749d215ce7c2de4079cc96b/contracts/ITOProtocolV1.sol#L197

Description: The function `take()` is designed to take the downside tokens from the user and give the upside tokens. There has been changes in the function `take()` to handle the case where the user provides more downside tokens than the maximum available upside tokens. In an overview, the protocol pulls downside tokens from the user

and takes the protocol fee and LP fee from it and allocate the remainder in the position's downside balance. After all, the correct accounting would need to ensure that all the downside tokens pulled from the user are distributed over the protocol fee, LP fee, and the position's `downsideTokenBalance`. However, the current implementation will suffer from rounding issues and the accounting can be off by a few WEIs.

```
ITOProtocolV1.sol
190:         (uint256 protocolFee, uint256 lpFee) = computeFee(_positionId, _maxDownsideTokenIn);
191:         uint256 _downsideTokenBalanceAfterFees = _maxDownsideTokenIn - protocolFee - lpFee;
192:         uint256 upsideTokenAmount = (_downsideTokenBalanceAfterFees * d.upsidePerDownside) /
↪  MULTIPLIER;
193:
194:         if(upsideTokenAmount > d.upsideTokenBalance) {
195:             upsideTokenAmount = d.upsideTokenBalance;
196:
197:             _downsideTokenBalanceAfterFees = (upsideTokenAmount * MULTIPLIER) /
↪  d.upsidePerDownside;
198:             _maxDownsideTokenIn = (_downsideTokenBalanceAfterFees * BP_MULTIPLIER) /
↪  (BP_MULTIPLIER - (d.lpFeeBp + protocolFeeBp));
199:             (protocolFee, lpFee) = computeFee(_positionId, _maxDownsideTokenIn);//@audit-issue
↪  rounding will make the accounting inaccurate, need to handle smoothly
200:         }
201:         if(_minUpsideTokenOut > upsideTokenAmount) {
202:             revert OutputTooLow();
203:         }
204:
```

As we can see, we are calculating `_maxDownsideTokenIn` reversely starting from the `d.upsideTokenBalance` and then decides the protocol fee and LP fee by the function `computeFee()`. The problem is that the function `computeFee()` will round the fee amount and `_maxDownsideTokenIn` will be a few WEIs more than `_downsideTokenBalanceAfterFees + protocolFee + lpFee`.

Impact The internal accounting will be off by a few WEIs and this can be possibly exploited by malicious users.

Recommendation: Calculate `lpFee` as `_maxDownsideTokenIn - downsideTokenBalanceAfterFees - protocolFee` rather than using `computeFee()`.

Client: Fixed in 2f02c1 following the recommendation.

Hans: Verified.

7.1.2 Rounding must be directed so as to be favorable to the protocol.

Severity: Medium

Context: https://github.com/Moai-Labs/upside-contracts/blob/9a0c5310c53a6c424749d215ce7c2de4079cc96b/contracts/ITOProtocolV1.sol#L228

Description: The function `untake()` is designed to take the upside tokens back from the user and release the downside tokens. The amount of upside tokens to be taken back is calculated by multiplying the amount of downside tokens to be released and the `upsidePerDownside` rate. In the calculation, the protocol uses the `MULTIPLIER` to avoid rounding issues but the rounding is directed downward. This is not favorable to the LP because the LP will lose the WEIs that are rounded down.

```
ITOProtocolV1.sol
223:     function untake(uint256 _positionId, uint256 _downsideTokenAmount) external {
224:         LPPosition memory d = lpPositions[_positionId];
225:
226:         if(block.timestamp > d.endDate) {
227:             revert PositionEnded();
228:         }
229:
230:         uint256 upsideTokenAmount = (_downsideTokenAmount * d.upsidePerDownside) /
↪  MULTIPLIER;//@audit-issue rounding direction must be reversed, maybe it's a good idea to receive
↪  upside as an input
231:
232:         downsideTokenAmounts[_positionId][msg.sender] -= _downsideTokenAmount;
233:         lpPositions[_positionId].downsideTokenBalance -= _downsideTokenAmount;
234:         lpPositions[_positionId].upsideTokenBalance += upsideTokenAmount;
235:
236:         d.upsideToken.safeTransferFrom(msg.sender, address(this), upsideTokenAmount);
237:         d.downsideToken.safeTransfer(msg.sender, _downsideTokenAmount);
238:
239:         emit Untake(_positionId, _downsideTokenAmount, upsideTokenAmount, msg.sender);
240:     }
241:
```

Impact The LP will lose the WEIs that are rounded down during `untake()`.

Recommendation: Round upward by adding `MULTIPLIER - 1` to the numerator.

Client: Acknowledged.

Hans: Acknowledged.

7.1.3 Incomplete support for fee-on-transfer tokens

Severity: Medium

Context: https://github.com/Moai-Labs/upside-contracts/blob/9a0c5310c53a6c424749d215ce7c2de4079cc96b/contracts/ITOProtocolV1.sol#L211C69-L211C88

Description: The protocol is designed to support any ERC20 tokens but it is not fully compatible with fee-on-transfer tokens. To support fee-on-transfer tokens, the protocol needs to calculate the amount of actually transferred tokens by checking the balance change. But in numerous places except for `supply()`, the protocol assumes the amount of transferred tokens is equal to the amount parameter of the transfer function.

ITOProtocolV1.sol

```
223:     function untake(uint256 _positionId, uint256 _downsideTokenAmount) external {
224:         LPPosition memory d = lpPositions[_positionId];
225:
226:         if(block.timestamp > d.endDate) {
227:             revert PositionEnded();
228:         }
229:
230:         uint256 upsideTokenAmount = (_downsideTokenAmount * d.upsidePerDownside) / MULTIPLIER;
231:
232:         downsideTokenAmounts[_positionId][msg.sender] -= _downsideTokenAmount;
233:         lpPositions[_positionId].downsideTokenBalance -= _downsideTokenAmount;
234:         lpPositions[_positionId].upsideTokenBalance += upsideTokenAmount;
235:
236:         d.upsideToken.safeTransferFrom(msg.sender, address(this),
↪  upsideTokenAmount);//@audit-issue not compatible with fee-on-transfer tokens
237:         d.downsideToken.safeTransfer(msg.sender, _downsideTokenAmount);
238:
239:         emit Untake(_positionId, _downsideTokenAmount, upsideTokenAmount, msg.sender);
240:     }
241:
```

Impact The protocol is not fully compatible with fee-on-transfer tokens.

Recommendation: Implement the balance change check in all places where the protocol pulls tokens from the user.

Client: Acknowledged. The protocol decided to not support fee-on-transfer tokens.

Hans: Acknowledged.

7.2 Low Risk Findings

7.2.1 Do not allow new positions with zero upside token amount

The function `supply()` is designed to create a new position with the given amount of upside tokens. But there is no check to prevent the user from creating a new position with zero upside tokens. Positions with zero upside tokens are useless and can be exploited by malicious users in a complicated way.

```
ITOProtocolV1.sol
74:     function supply(
75:         address _downsideToken,
76:         address _upsideToken,
77:         uint256 _upsideTokenAmount,
78:         uint256 _exchangeRate,
79:         uint32 _startDate,
80:         uint32 _endDate,
81:         uint16 _feeBp
82:     ) external {
83:         if(_feeBp > MAX_FEE_BP) {
84:             revert FeeTooHigh();
85:         }
86:         if(block.timestamp > _startDate || _startDate > _endDate) {
87:             revert InvalidDuration();
88:         }
89:
90:         uint256 previousBalance = IERC20Metadata(_upsideToken).balanceOf(address(this));
91:         IERC20Metadata(_upsideToken).safeTransferFrom(msg.sender, address(this),
↪  _upsideTokenAmount);
92:         _upsideTokenAmount = IERC20Metadata(_upsideToken).balanceOf(address(this)) -
↪  previousBalance;
93:
94:         lpPositions.push(LPPosition(
95:             msg.sender,
96:             _feeBp,
97:             _startDate,
98:             _endDate,
99:             IERC20Metadata(_downsideToken),
100:             IERC20Metadata(_upsideToken),
101:             _upsideTokenAmount,//@audit-issue zero amount check
102:             _exchangeRate,
103:             0
104:         ));
105:         emit Supply(lpPositions.length - 1);
106:     }
```

Client: Acknowledged.

Hans: Acknowledged.

7.3 Informational Findings

7.3.1 Incorrect comments for the `_exchangeRate` parameter

From the usage, it is obvious the `_exchangeRate` parameter is the number of upside tokens to pay per downside token and it should be in decimals of `18 + D{upside} - D{downside}`. But the example in the comment says the opposite and it is confusing. The comment should be fixed and it is recommended to add more real world examples to clarify the decimals.

ITOProtocolV1.sol

```
65:     /// @notice Allows LPs to create a position
66:     /// @param _downsideToken Address of the downside token
67:     /// @param _upsideToken Address of the upside token
68:     /// @param _upsideTokenAmount The number of upside tokens to provide to this position
69:     /// @param _exchangeRate The number of upside tokens to pay per downside token
70:     /// @dev _exchangeRate should be computed like so: ((downsideAmountWei / upsideAmountWei) *
↪  10**18)//@audit-info Wrong comment. The rate must be in decimals of 18 + D{upside} - D{downside}
↪  from the usage.
71:     /// @param _startDate Unix timestamp of when this position should begin
72:     /// @param _endDate Unix timestamp of when this position should end
73:     /// @param _feeBp The maximum liquidity provider fee to charge
```

Client: Fixed in d19dcc.

Hans: Verified.