Upside.fun(Uptoken) Security Review

Reviewer
Hans

May 16, 2025

Contents

1 Executive Summary

As a security advisor of Upside.fun, Hans reviewed Uptoken.

Summary

Type of Project	AMM
Timeline	7th May, 2025 - 16th May, 2025
Methods	Manual Review

A comprehensive security review identified a total of 7 issues.

Repository	**Commit**
uptoken-contracts-experiment5	e030f7c99c66319bf985ae9480849b23125b21c3

Total Issues

High Risk	0
Medium Risk	1
Low Risk	1
Informational	5

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

Repository	**Commit**
uptoken-contracts-experiment5	262f7e679080a4af47e6fcbd8cb5dd7af345b037

2 Scope of the Audit

This audit was conducted for 2 contracts.

- UpsideMetaCoin.sol
- UpsideProtocol.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. 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

The Upside Protocol enables the tokenization of URLs via `UpsideMetaCoin` and provides trading through an AMM with the constant product formula.

6 Findings

6.1 Medium Risk

6.1.1 Possible revert during a swap

Context: UpsideProtocol.sol#L214

Description: While selling a metacoin, the `swap()` function calls `processSwapFee()` and transfers staker fees to the staking contract using `distributeRewards()`.

```
    if (_isBuy) {
        claimableProtocolFees += fee;
        feeToProtocol = fee;
    } else {
        feeToDeployer = (fee * swapDeployerFeeBp) / 10000;
        claimableDeployerFees[_metaCoinAddress][_deployer] += feeToDeployer;
        feeToStakers = fee - feeToDeployer;

        IERC20Metadata(_metaCoinAddress).approve(stakingContractAddress, feeToStakers);
        IUpsideStaking(stakingContractAddress).distributeRewards(_metaCoinAddress, feeToStakers);
        ↪  //@audit revert
    }
```

But actually, the caller deposits the metacoin at the end of the `swap()` function, and `distributeRewards()` should transfer to the staking contract using the existing metacoin balance. So if this contract has fewer tokens than the expected fee, it will revert.

Here is an example.

- Originally, the contract had 1B `Meta`.
- After some swaps, 999M `Meta` has been sold, and the contract has 1M `Meta`.
- When a user calls `swap()` to sell 200M `Meta` that he bought before, the `processSwapFee()` function transfers the staker fee to the staking contract before receiving 200M `Meta` from the user.
- If the staker fee is greater than 1M, it will revert inside `distributeRewards()` because the contract has insufficient `Meta`.

Impact: Users couldn't sell their metacoins.

Recommendation: The `swap()` function should pull the metacoin from the user before handling the fees.

Upside: Fixed at commit cde347f.

Hans: Verified.

6.2 Low Risk

6.2.1 The swap calculation is done in favor of users

Context: UpsideProtocol.sol#L254-L267

Description: During the swap, `newMetaCoinReserves` and `newLiquidityTokenReserves` are rounded down. Because `amountOut` is calculated as `oldReserves - newReserves`, `amountOut` is rounded up which is not in favor of the protocol.

```
    if(_isBuy) {
        newLiquidityTokenReserves = metaCoinInfo.liquidityTokenReserves + amountInAfterFee;

        newMetaCoinReserves = k / newLiquidityTokenReserves;
        amountOut = metaCoinInfo.metaCoinReserves - newMetaCoinReserves;

        if(amountOut > metaCoinInfo.metaCoinReserves) revert InsufficientLiquidity(); // @dev Sanity
        ↪  check
    } else {
        newMetaCoinReserves = metaCoinInfo.metaCoinReserves + amountInAfterFee;
        newLiquidityTokenReserves = k / newMetaCoinReserves;
        amountOut = metaCoinInfo.liquidityTokenReserves - newLiquidityTokenReserves;

        if(amountOut > metaCoinInfo.liquidityTokenReserves - INITIAL_LIQUIDITY_RESERVES) revert
        ↪  InsufficientLiquidity(); // @dev Sanity check
    }
```

Additionally, because `newMetaCoinReserves` and `newLiquidityTokenReserves` are rounded down, `newX * newY` would be less than `k`.

Impact: An invariant of the constant product formula is not maintained.

Recommendation: Here is a recommended mitigation.

```
if (_isBuy) {
    newLiquidityTokenReserves = metaCoinInfo.liquidityTokenReserves + amountInAfterFee;
    amountOut = (metaCoinInfo.metaCoinReserves * amountInAfterFee) / newLiquidityTokenReserves;
    newMetaCoinReserves = metaCoinInfo.metaCoinReserves - amountOut;
} else {
    newMetaCoinReserves = metaCoinInfo.metaCoinReserves + amountInAfterFee;
    amountOut = (metaCoinInfo.liquidityTokenReserves * amountInAfterFee) / newMetaCoinReserves;
    newLiquidityTokenReserves = metaCoinInfo.liquidityTokenReserves - amountOut;

    if(newLiquidityTokenReserves < INITIAL_LIQUIDITY_RESERVES) revert InsufficientLiquidity();
}
```

Upside: Fixed at commit 9ff098e.

Hans: Verified.

6.3 Informational

6.3.1 Lack of validations in `setFeeInfo()`

Context: UpsideProtocol.sol#L172, UpsideProtocol.sol#L321

Description: During the time fee calculation, it could revert if the fee parameters are set incorrectly, potentially causing a DoS.

```
uint256 intervalsElapsed = secondsPassed / fee.swapFeeDecayInterval;
uint256 feeReduction = intervalsElapsed * fee.swapFeeDecayBp;

if (feeReduction >= (fee.swapFeeStartingBp - fee.swapFeeFinalBp)) {
    swapFeeBp = fee.swapFeeFinalBp;
} else {
    swapFeeBp = fee.swapFeeStartingBp - feeReduction;
}
```

Recommendation: Recommend adding more validations in the `setFeeInfo()` function.

```
if(
    _newFeeInfo.tokenizeFeeEnabled &&
    (_newFeeInfo.swapFeeDeployerBp > 10000 ||
    _newFeeInfo.swapFeeDecayBp > 10000 ||
    _newFeeInfo.swapFeeFinalBp > 10000 ||
    _newFeeInfo.swapFeeStartingBp > 10000 ||
    _newFeeInfo.tokenizeFeeDestinationAddress == address(0) ||
    _newFeeInfo.swapFeeStartingBp < _newFeeInfo.swapFeeFinalBp ||
    _newFeeInfo.swapFeeDecayInterval == 0)
    ) {
        revert InvalidSetting();
}
```

Upside: Fixed at commit 9ff098e.

Hans: Verified.

6.3.2 Incorrect event in `processSwapFee()`

Context: UpsideProtocol.sol#L228

Description: After introducing a static fee on sell, the `SwapFeeProcessed` event is emitted with the wrong fee percentage.

```
        if (_isBuy) {
            // @dev On buy, the dynamic time fee is used
            fee = (_tokenAmount * swapFeeBp) / 10000;
            tokenAmountAfterFee = _tokenAmount - fee;
            ...
        } else {
            // @dev On sell, a static percentage bp is used (impl could be significantly improved to
            ↪  save gas)
            fee = (_tokenAmount * feeInfo.swapFeeSellBp) / 10000;
            tokenAmountAfterFee = _tokenAmount - fee;
            ...
        }

        emit SwapFeeProcessed(
            _metaCoinAddress,
            _isBuy,
            secondsPassed,
            swapFeeBp, //@audit incorrect on sell
            fee,
            feeToProtocol,
            feeToDeployer,
            feeToStakers
        );
```

Recommendation: Recommend emitting `feeInfo.swapFeeSellBp` on sell.

Upside: Fixed at commit cde347f.

Hans: Verified.

6.3.3 Useless validation in `swap()`

Context: UpsideProtocol.sol#L267

Description: The below validation is useless because the above line will revert if the revert condition is met.

```
    newMetaCoinReserves = metaCoinInfo.metaCoinReserves - amountOut;

    if(amountOut > metaCoinInfo.metaCoinReserves) revert InsufficientLiquidity(); // @dev Sanity check
```

Recommendation: Recommend removing the validation.

Upside: Fixed at commit cde347f.

Hans: Verified.

6.3.4 `ERC20Permit` should be initialized using a token name

Context: UpsideMetaCoin.sol#L34

Description: Openzeppelin's ERC20Permit implementation recommends using a token name.

```
/**
 * @dev Initializes the {EIP712} domain separator using the `name` parameter, and setting `version`
 ↪  to `"1"`.
 *
 * It's a good idea to use the same `name` that is defined as the ERC-20 token name.
 */
constructor(string memory name) EIP712(name, "1") {}
```

The `UpsideMetaCoin` contract uses `_symbol`, but it has no impact because the metacoin's name and symbol are same.

```
constructor(
    string memory _name,
    string memory _symbol,
    uint256 _totalSupply,
    address _upsideProtocol
) ERC20(_name, _symbol) Ownable(msg.sender) ERC20Permit(_symbol)
```

Recommendation: Recommend using `_name` instead of `_symbol`.

Upside: Fixed at commit 262f7e6.

Hans: Verified.

6.3.5 Lack of events after updating a token name and symbol.

Context: UpsideMetaCoin.sol#L52

Description: After updating a token name and symbol in `setNameAndSymbol()`, it doesn't emit an event.

```
function setNameAndSymbol(string memory _name, string memory _symbol) external onlyOwner {
    _customName = _name;
    _customSymbol = _symbol;
}
```

Recommendation: Recommend emitting an event after an update.

Upside: Fixed at commit 262f7e6.

Hans: Verified.