Upside Protocol Audit Report

Jamil Hallack

May 26, 2025

Contents

Upside Protocol Initial Audit Report

Version 0.1

jamillhallak.com

June 2, 2025

Upside Protocol Audit Report

Jamil Hallack

May 26, 2025

Upside Protocol Audit Report

Prepared by **Jamil**

Lead Auditors - Jamil Hallack

Assisting Auditors - None

Table of contents

See table

About Jamil

Jamil Hallack is a Smart Contract Security Engineer with 4+ years of experience auditing and building secure DeFi, GameFi, and cross-chain protocols. He specializes in Solidity, Yul, Huff, and formal verification (Certora, Halmos), with deep EVM knowledge and a strong track record in uncovering high-impact vulnerabilities via Code4rena, Sherlock, and Cyfrin. Jamil delivers gas-optimized, audit-grade code and rigorously tested systems aligned with best security practices.

Disclaimer

The Jamil Hallack team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

The findings described in this document correspond to the following commit hash

`470800e6f2262ed44d38258f627eaea1c02283d6`

Scope

`./contracts/UpsideProtocol.sol`
`./contracts/UpsideMetaCoin.sol`

Protocol Summary

Upside is a social prediction market (think Polymarket + reddit) where you win money by being early on viral tweets, Spotify songs, YouTube videos, etc.

Roles

- Owner: Sets fee parameters; Sets staking contract address; Claims fees; Changes name / symbol of a MetaCoin; Disables whitelist; Withdraws liquidity; Sets tokenization fees; Whitelists addresses for transfers

Executive Summary

Issues found

Severity	Number of issues found
High	0
Medium	0
Low	0
Info	7
Total	7

Findings

[I-1] Missing Array Length Consistency Check in `setMetaCoinWhitelist()`

Summary
The `setMetaCoinWhitelist()` function assumes the input arrays are of equal length without enforcing it.

Vulnerability Details
If the arrays differ in length, this could cause out-of-bounds errors or result in incomplete whitelist updates, potentially leading to unexpected contract behavior.

Proof of Concept

```
function setMetaCoinWhitelist(
    address[] calldata _metaCoinAddresses,
    address[] calldata _walletAddresses,
    bool[] calldata _isWhitelisted
) external onlyOwner {
    for (uint256 i; i < _metaCoinAddresses.length; ) {
        metaCoinWhitelistMap[_metaCoinAddresses[i]][_walletAddresses[i]] = _isWhiteliste
        unchecked { ++i; }
    }
}
```

Remediation Add a validation to ensure all input arrays are the same length before proceeding.

```
require(
    _metaCoinAddresses.length == _walletAddresses.length &&
    _walletAddresses.length == _isWhitelisted.length,
    "Length mismatch"
);
```

Reference:
UpsideProtocol.sol#L321-L333

[I-2] Misleading Timestamp Emitted in `WithdrawLiquidityTimerStarted` Event

Summary
The `WithdrawLiquidityTimerStarted` event in the `withdrawLiquidity()` function emits the unlock timestamp instead of the actual time the cooldown was triggered.

Vulnerability Details
This may mislead off-chain systems expecting the real-time start of cooldown periods.

Proof of Concept

```
emit WithdrawLiquidityTimerStarted(block.timestamp + WITHDRAW_LIQUIDITY_COOLDOWN); // mi
```

Remediation Emit the current block timestamp:

```
emit WithdrawLiquidityTimerStarted(block.timestamp);
```

Reference:
UpsideProtocol.sol#L364

[I-3] Missing Zero Address Validation in `claimDeployerFees()` function

Summary
The function `claimDeployerFees()` can transfer tokens to the zero address, potentially resulting in irreversible token loss.

Vulnerability Details
This function does not check if the `_recipient` is non-zero, which may lead to accidental fund burns by deployers or dApps interacting with this method.

Proof of Concept

```
IERC20Metadata(_metaCoinAddress).safeTransfer(_recipient, fees); // no validation
```

Remediation

Define a custom error (e.g., `error InvalidRecipient();`) and add a zero address check before proceeding:

```
if (_recipient == address(0)) revert InvalidRecipient();
```

Reference:
UpsideProtocol.sol#L301-L307

[I-4] Missing Zero Address Validation in `claimProtocolFees()`

Summary
The `claimProtocolFees()` function allows the owner to transfer protocol fees to the zero address.

Vulnerability Details
Since the zero address is not a valid recipient, sending fees there results in unrecoverable loss of funds.

Proof of Concept

```
IERC20Metadata(liquidityTokenAddress).safeTransfer(_recipient, fees); // unsafe
```

Remediation

Define a custom error (e.g., `error InvalidRecipient();`) and add a zero address check before proceeding:

```
if (_recipient == address(0)) revert InvalidRecipient();
```

Reference:
UpsideProtocol.sol#L417-L423

[I-5] Missing Zero Address Validation in `setStakingContractAddress()`

Summary
The `setStakingContractAddress()` function allows setting the staking contract to a zero address.

Vulnerability Details
Setting the staking contract to the zero address can cause any funds sent there to be lost and unrecoverable.

Proof of Concept

```
stakingContractAddress = _newStakingContractAddress; // no check
```

Remediation

Define a custom error (e.g., `error InvalidRecipient();`) and add a zero address check before proceeding:

```
if (_newStakingContractAddress == address(0)) revert InvalidStakingContract();
```

Reference:
UpsideProtocol.sol#L427-L431

[I-6] Missing Zero Address Validation in `swap()`

Summary
The `_recipient` parameter in `swap()` function is not validated, which could result in tokens being permanently lost if a zero address is passed.

Vulnerability Details
If `_recipient == address(0)`, the contract transfers tokens to the zero address, effectively burning them. This introduces an unnecessary risk of user funds being lost permanently.

Proof of Concept
In `UpsideProtocol.sol`, the following line in the `swap()` function transfers tokens without validating `_recipient`:

```
IERC20Metadata(_metaCoinAddress).safeTransfer(_recipient, amountOut); // unsafe if _rec
```

Remediation

Define a custom error (e.g., `error InvalidRecipient();`) and add a zero address check before proceeding:

```
if (_recipient == address(0)) revert InvalidRecipient();
```

Reference:
UpsideProtocol.sol#L291-L296

[I-7] Missing Zero Address Check in `setTokenizeFee()`

Summary
The `setTokenizeFee()` function allows inserting a zero address as a fee token address .

Vulnerability Details
If the fee token address is set to the zero address, any fees sent will be lost.

Proof of Concept

```
function setTokenizeFee(address _tokenizeFeeAddress, uint256 _tokenizeFeeAmount) externa
        tokenizeFeeMap[_tokenizeFeeAddress] = _tokenizeFeeAmount; // no input validatio
        emit TokenizeFeeSet(_tokenizeFeeAddress, _tokenizeFeeAmount);
    }
```

Remediation

Define a custom error (e.g., `error  InvalidFeeToken();`) and add a zero address check before proceeding:

```
if (_tokenizeFeeAddress == address(0)) revert InvalidFeeToken();
```

Reference:
UpsideProtocol.sol#L435-L438