AstraSec

Horizon Protocol

Security Audit Report

May 20, 2024

Contents

1 Introduction

1.1 About Horizon Protocol

Horizon Protocol is a DeFi platform that facilitates the on-chain trading of synthetic assets that represent the real economy. Horizon Protocol seeks to provide exposure to real-world assets risk/return profiles via smart contracts on the blockchain. Forked from Synthetix, Horizon Protocol will leverage the time-tested derivative liquidity protocol and bring interoperability, scalability and a whole new array of tradable, real-world derivative products to the DeFi ecosystem.

1.2 Source Code

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit. Note this audit only covers the Account.sol, AccountProxy.sol, Events.sol, Factory.sol, and Settings.sol contracts.

- https://github.com/Horizon-Protocol/horizon-perps-margin
- CommitID: 6f7d8b0

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/Horizon-Protocol/horizon-perps-margin
- CommitID: 16353b5

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Horizon` protocol. Throughout this audit, we identified a total of 3 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	1	-	-	1
Low	2	-	-	2
Informational	-	-	-	-
Undetermined	-	-	-	-

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~M-1~~ Revisit Fee Charge Logic in Account::executeConditionalOrder()

~~L-1~~ Integration of Non-Standard ERC20 Tokens

~~L-2~~ Potential Risks Associated with Centralization

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[M-1] Revisit Fee Charge Logic in Account::executeConditionalOrder()

Target	Category	IMPACT	LIKELIHOOD	STATUS
Account.sol	Business Logic	Medium	Medium	Addressed

In the `Account` contract, the `executeConditionalOrder()` function allows the function caller to execute conditional order for a given `_conditionalOrderId`. While examining its logic,we notice the current fee distribution logic is not correct.

To elaborate, we show below the related code snippet. When executing a conditional order, a certain amount of fees will be charged and the fees will be different depending on the executor. Specifically, the calling of `_getFeeDetails()` will always return a non-zero `fee` and non-zero `feeToken` when called by a `Gelato` executor. But the returned values for `fee` and `feeToken` will be zero if being called by a non-`Gelato` executor. This will cause those calls initiated by non-`Gelato` executor to fail, as executing `SafeERC20.safeTransfer(IERC20(address(0)), feeCollector, _fee)` will always revert.

Account::executeConditionalOrder()

```
    function executeConditionalOrder(uint256 _conditionalOrderId)
        external
        override
        nonReentrant
        isAccountExecutionEnabled
    {
        ...
        // remove gelato task from their accounting
        /// @dev will revert if task id does not exist {Automate.cancelTask: Task not found}
        /// @dev if executor is not Gelato, the task will still be cancelled
        automate.cancelTask({taskId: conditionalOrder.gelatoTaskId});

        // impose and record fee paid to executor
        uint256 fee = _payExecutorFee();

        // define Horizon Protocol PerpsV2 market
        IPerpsV2MarketConsolidated market =
            _getPerpsV2Market(conditionalOrder.marketKey);
        ...
    }

    /// @notice pay fee for conditional order execution
    /// @dev fee will be different depending on executor
    /// @return fee amount paid
    function _payExecutorFee() internal returns (uint256 fee) {
        address feeToken;
        (fee, feeToken) = _getFeeDetails();
        _transfer(fee, feeToken);
    }
```

Remediation When charging the execution fees, the non-`Gelato` executor should be considered.

[L-1] Integration of Non-Standard ERC20 Tokens

Target	Category	IMPACT	LIKELIHOOD	STATUS
Account.sol	Business Logic	Low	Low	Addressed

Inside the `Account::_modifyAccountMargin()` function, the statements of `MARGIN_ASSET.transferFrom(msg.sender, address(this), _abs(_amount))` (line 464) and `MARGIN_ASSET.transfer(msg.sender, _abs(_amount))` (line 472) are employed to transfer the user's asset into the `Account` contract or transfer the asset to user from the `Account` contract. However, in the case of `USDT-like` token whose transfer()/transferFrom() lack a return value, it would lead to a revert. Given this, we recommend employing the widely-used `SafeERC20` library (which serves as a wrapper for ERC20 operations while accommodating a diverse range of non-standard ERC20 tokens) to address this case.

Account::_modifyAccountMargin()

```
458     /// @notice deposit/withdraw margin to/from this margin account
459     /// @param _amount: amount of margin to deposit/withdraw
460     function _modifyAccountMargin(int256 _amount) internal {
461         // if amount is positive, deposit
462         if (_amount > 0) {
463             /// @dev failed Horizon Protocol asset transfer will revert and not
                    return false if unsuccessful
464             MARGIN_ASSET.transferFrom(msg.sender, address(this), _abs(_amount));

466             EVENTS.emitDeposit({user: msg.sender, amount: _abs(_amount)});
467         } else if (_amount < 0) {
468             // if amount is negative, withdraw
469             _sufficientMargin(_amount);

471             /// @dev failed Horizon Protocol asset transfer will revert and not
                    return false if unsuccessful
472             MARGIN_ASSET.transfer(msg.sender, _abs(_amount));

474             EVENTS.emitWithdraw({user: msg.sender, amount: _abs(_amount)});
475         }
476     }
```

Remediation Replace `transfer()`/`transferFrom()` with `safeTransfer()`/`safeTransferFrom()`.

[L-2] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Low	Low	Addressed

In the `Horizon` protocol, the existence of a privileged `owner` account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative functions potentially affected by the privileges associated with the privileged account.

Example Privileged Operations in `Horizon Protocol`

```
    /// @inheritdoc IFactory
    function upgradeAccountImplementation(address _implementation)
        external
        override
        onlyOwner
    {
        if (!canUpgrade) revert CannotUpgrade();
        implementation = _implementation;
        emit AccountImplementationUpgraded({implementation: _implementation});
    }

    /// @inheritdoc IFactory
    function removeUpgradability() external override onlyOwner {
        canUpgrade = false;
    }

    /// @inheritdoc ISettings
    function setAccountExecutionEnabled(bool _enabled)
        external
        override
        onlyOwner
    {
        accountExecutionEnabled = _enabled;

        emit AccountExecutionEnabledSet(_enabled);
    }

    /// @inheritdoc ISettings
    function setExecutorFee(uint256 _executorFee) external override onlyOwner {
        executorFee = _executorFee;

        emit ExecutorFeeSet(_executorFee);
    }
    /// @inheritdoc ISettings
    function setTokenWhitelistStatus(address _token, bool _isWhitelisted)
        external
        override
        onlyOwner
    {
        _whitelistedTokens[_token] = _isWhitelisted;

        emit TokenWhitelistStatusUpdated(_token, _isWhitelisted);
    }
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been resolved as the team confirms that the `owner` will be behind the protocol DAO multi-sig wallet.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 176 2267 4194
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI