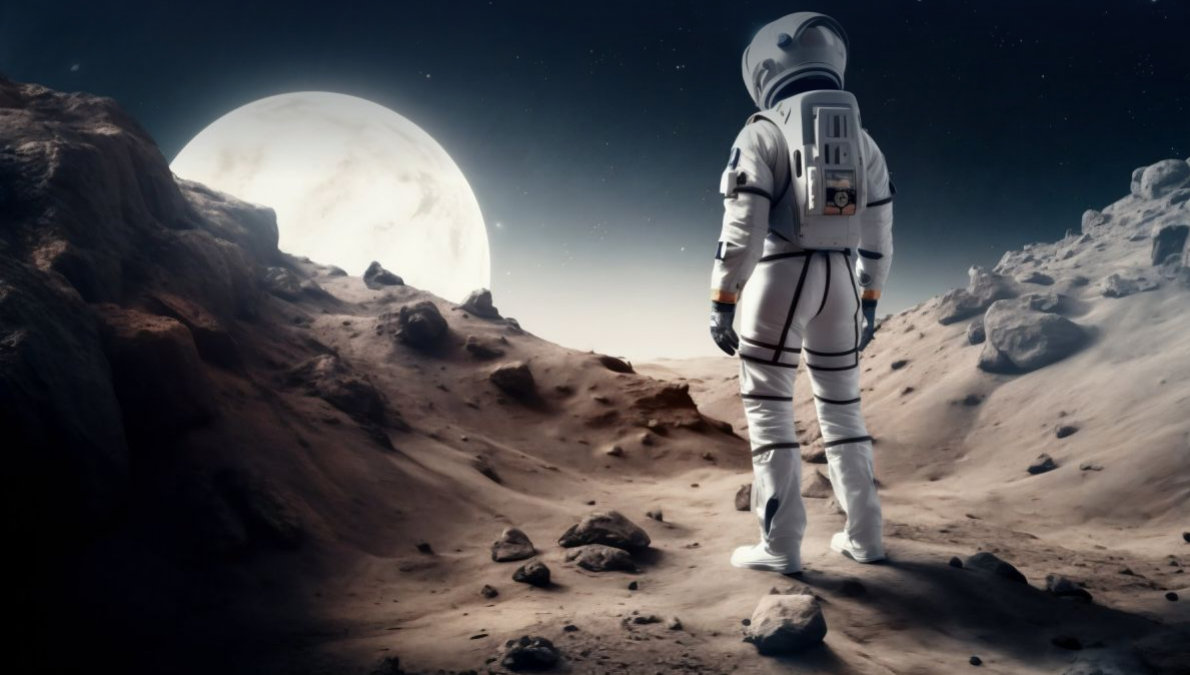
AstraSec

SpinUp
Security Audit Report

November 25, 2025

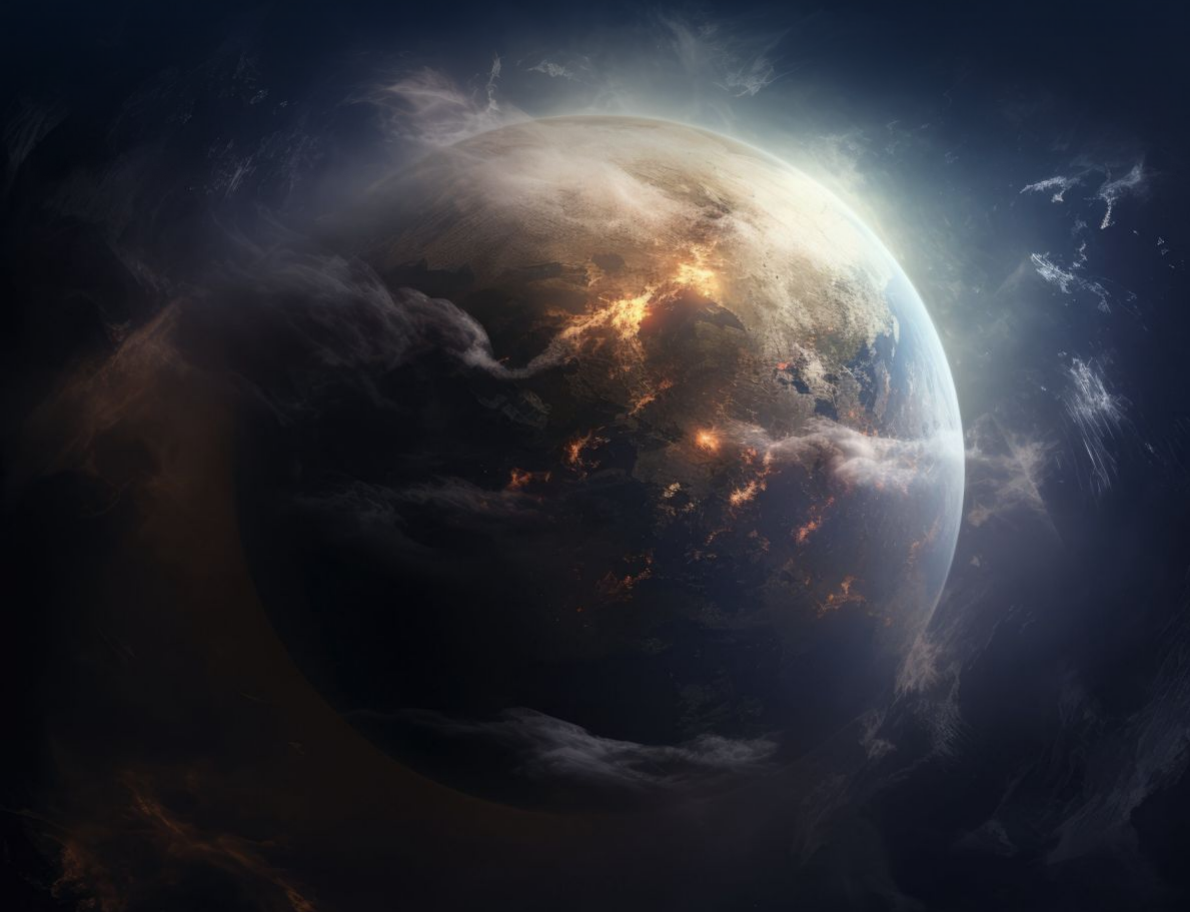


Contents

1 Introduction

1.1 About SpinUp

SpinUp is a comprehensive and innovative DeFi ecosystem built on the Hyperliquid, seamlessly integrating a Liquid Staking platform, a dynamic MEME launchpad, and a highly efficient MEME DEX. This ecosystem is designed to empower users with diverse staking opportunities, facilitate the launch and growth of meme-based tokens, and provide a decentralized exchange tailored for meme coin trading.



1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/hyperpie/tree/feat/hype-lst-staking
- CommitID: ba87f502f42df10a9275435a36ae78e6da61c2cf

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

- https://github.com/magpiexyz/hyperpie
- CommitID: a24ad960d9be67e693cc6f0f5d1afce0a4bdd264

1.3 Revision History

Version	Date	Description
v1.0	March 7, 2025	Initial Audit
v1.1	March 22, 2025	PR1
v1.2	May 28, 2025	PR5
v1.3	July 4, 2025	PR32
v1.4	July 24, 2025	PR31, PR6
v1.5	July 29, 2025	PR35
v1.6	August 21, 2025	PR37
v1.7	November 24, 2025	PR52

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the SpinUp protocol. Throughout this audit, we identified a total of 9 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	3	–	–	3
Medium	4	1	–	3
Low	2	–	–	2
Informational	–	–	–	–
Total	9	1	–	8

3 Vulnerability Summary

3.1 Overview

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

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:

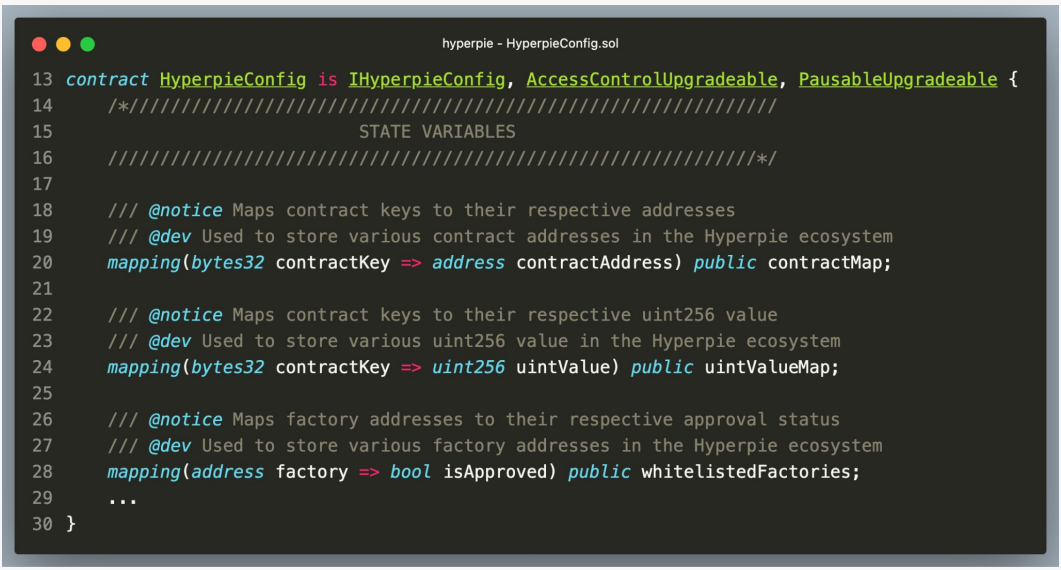
Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security 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 re-mediation. 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

3.3.1 [H-1] Storage Layout Conflict in HyperpieConfig

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
HyperpieConfig.sol	Business Logic	High	High	Addressed

The HyperpieConfig contract, designed for upgradeability, newly incorporates the PausableUpgradeable contract to introduce a pausing feature. However, this modification introduces a storage slot conflict, as the updated storage layout no longer aligns with prior versions. This misalignment poses a critical risk to contract upgradeability, potentially leading to undefined behavior in production.



Remediation Avoid introducing storage conflict during the upgrade process.

3.3.2 [H-2] Improper Access Control in HyperpieConfig::pause()/unpause()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
HyperpieConfig.sol	Business Logic	High	High	Addressed

Upon reviewing the HyperpieConfig implementation, we identify a critical flaw: the pause() function reverts for authorized pausers and permits execution by unauthorized users, inverting the intended access control. Similarly, the unpause() function exhibits the same issue, allowing unintended access and compromising the contract's security.

hyperpie - HyperpieConfig.sol

```
87  /// @notice Pauses the contract, this will pause all the contracts that are using the HyperpieConfig contract
88  function pause() external {
89      if (hasRole(HyperpieConstants.PAUSER_ROLE, msg.sender)) {
90          revert IHyperpieConfig.CallerNotHyperpiePauser();
91      }
92      _pause();
93  }
94
95  /// @notice Unpauses the contract, this will unpause all the contracts that are using the HyperpieConfig contract
96  function unpause() external {
97      if (hasRole(HyperpieConstants.DEFAULT_ADMIN_ROLE, msg.sender)) {
98          revert IHyperpieConfig.CallerNotHyperpieConfigAdmin();
99      }
100     _unpause();
101 }
```

Remediation Correct the implementation of the pause()/unpause() functions to ensure proper access control.

3.3.3 [H-3] Native/Wrapped Token Double-Spend in MEME Token Purchase

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
MEMELaunchpad.sol	Business Logic	High	High	Addressed

While examining the implementation of MEMELaunchpad, we identify a double-spend vulnerability in the native token payment flow when purchasing MEME tokens. Specifically, in the buyMemeTokenWithoutPoolCreationNative() function, the caller's native token (msg.value) is initially used for payment. However, during execution, the function further calls LaunchpadLibrary::buyMemeToken(), which incorrectly processes an additional payment using wrapped native tokens (WHYPE). This results in duplicate deductions —effectively charging users twice for a single transaction (once in native tokens and again in wrapped tokens).

Moreover, the MEMELaunchpad::createLaunch()/buyMemeTokenWithPoolCreation-AndSwapNative() functions share the same issue.

hyperpie - MEMELaunchpad.sol

```
272 function buyMemeTokenWithoutPoolCreationNative(
273     address _memeToken,
274     uint256 _minReceivedMemeTokenAmount
275 ) external payable nonReentrant whenNotPaused {
276     LaunchInfo storage launchInfo = launches[_memeToken];
277     LaunchpadLibrary.buyMemeTokenPreChecksNative(launchInfo, msg.value);
278     LaunchpadLibrary.tradePreChecks(launchInfo, _memeToken, msg.value);
279
280     uint256 wrappedNativeAmount = LaunchpadLibrary.wrapNativeToWype(msg.value);
281     _buyMemeToken(launchInfo, wrappedNativeAmount, _minReceivedMemeTokenAmount);
282 }
```

hyperpie - LaunchpadLibrary.sol

```
141 function buyMemeToken(
142     ...
143 ) external returns (uint256) {
144     ...
145     IERC20(launchInfo.depositTokenAddress).safeTransferFrom(msg.sender, address(this), _depositTokenAmountBeforeFee);
146     IERC20(launchInfo.memeToken).safeTransfer(msg.sender, memeTokenAmount);
147
148     transferTradingFees(launchInfo, _depositTokenAmountBeforeFee, _tokenCreatorFee, _feeInfos);
149
150     return memeTokenAmount;
151 }
```

Remediation Ensure the payment flow exclusively uses either native tokens or wrapped tokens, but not both.

3.3.4 [M-1] Properly Update totalTradingFee/totalPoolCreationFee in setFeeInfo()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
MEMELaunchpad.sol	Business Logic	High	Low	Addressed

The setFeeInfo() function contains a logical error in fee calculation update: it computes the new totalFeePercent in local variable by adjusting it with _feePercent and subtracting the old feeInfo.feePercent, but fails to update the corresponding storage variables (totalTradingFee or totalPoolCreationFee). This oversight leads to inaccurate fee tracking, as the contract's global fee totals remain outdated despite individual feeInfo entries being updated.

hyperpie - MEMELaunchpad.sol

```
368 function setFeeInfo(
369     uint256 _index,
370     uint256 _feePercent,
371     address _feeDestination,
372     bool _isTradingFee
373 )
374     external
375     onlyDefaultAdmin
376 {
377     ...
378
379     FeeInfo storage feeInfo = _isTradingFee ? tradingFeeInfos[_index] : poolCreationFeeInfos[_index];
380
381     uint256 totalFeePercent = _isTradingFee ? totalTradingFee : totalPoolCreationFee;
382     totalFeePercent += _feePercent;
383     totalFeePercent -= feeInfo.feePercent;
384     if (totalFeePercent > HyperpieConstants.DENOMINATOR) {
385         revert FeeTooHigh();
386     }
387
388     feeInfo.feePercent = _feePercent;
389     feeInfo.feeDestination = _feeDestination;
390
391     emit FeePercentSet(_index, _feePercent, _feeDestination, _isTradingFee);
392 }
```

Remediation Properly store the newly calculated totalFeePercent to the appropriate storage variables (totalTradingFee or totalPoolCreationFee).

3.3.5 [M-2] Potential DoS Attack for HyperpieRouter::addLiquidity()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
HyperpieRouter.sol HyperpiePair.sol	Business Logic	High	Low	Addressed

The addLiquidity() function in HyperpieRouter contains a potential Denial-of-Service (DoS) vulnerability due to the possible manipulated liquidity pools. The potential attack vector is as below:

- An attacker can pre-create a [MEME, WHYPE] pair and donate a minimal amount (1 wei) of WHYPE to the pair.
- By calling sync(), the attacker forces the pool to update its reserves to (0, 1)
- When adding liquidity, the _quoteliquidity() function reverts due to invalid reserve ratios (when either reserveA or reserveB is 0), effectively blocking liquidity additions.

Notably, this vulnerability impacts the MEMELaunchpad contract, leading to failed liquidity additions during MEME launch termination and ultimately causing MEME launch failures.

hyperpie - HyperpieRouter.sol

```
314 function _quoteLiquidity(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB) {
315     if (amountA == 0) revert InsufficientAmount();
316     if (reserveA == 0 || reserveB == 0) revert InsufficientLiquidity();
317     amountB = (amountA * reserveB) / reserveA;
318 }
```

hyperpie - HyperpiePair.sol

```
267 function sync() external nonReentrant {
268     _update(IERC20(token0).balanceOf(address(this)), IERC20(token1).balanceOf(address(this)), reserve0, reserve1);
269 }
```

Remediation Prevent the HyperpiePair::sync() function from being called to manipulate pool reserves until initial liquidity is added.

3.3.6 [M-3] Sandwich Attack on updateHyperpiePrice()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
PriceProvider.sol	Business Logic	Medium	Low	Mitigated

The *_updatemHypePrice()* function, which allows a price oracle to periodically update the *hHYPE/HYPE* exchange rate, is susceptible to a sandwich attack. This vulnerability stems from the fact that staked *HYPE* within SpinUp is used for staking on Hyperliquid L1, and the rewards generated from this staking activity generally cause the hHYPE/HYPE exchange rate to gradually increase over time. A malicious actor can exploit this predictable price increase by strategically inserting transactions immediately before and after the oracle's call to _updatemHypePrice() within the same block. The attack sequence is as follows:

- Stake *HYPE*: The attacker stakes *HYPE* into SpinUp just before the *_updatemHypePrice()* function is called.
- Oracle Update: The price oracle calls *_updatemHypePrice()* to update the *hHYPE/HYPE* exchange rate.
- Withdraw *HYPE*: The attacker immediately withdraws their staked HYPE after the update.

As a direct result of the increased exchange rate, the attacker receives more *HYPE* upon withdrawal than they would have received before the update. This excess *HYPE* effectively comes from the staking rewards that should be distributed among other stakers.

hyperpie - PriceProvider.sol

```
66  function _updatemHypePrice(uint256 exchangeRate) internal {
67      address mHYPE = hyperpieConfig.getAddress(HyperpieConstants.MHYPE_TOKEN);
68      ImHYPE(mHYPE).updateExchangeRateToUnderlying(exchangeRate);
69      emit ExchangeRateUpdate(HyperpieConstants.PLATFORM_TOKEN_ADDRESS, mHYPE, exchangeRate);
70  }
```

Remediation To mitigate the issue, it is recommended that the newly updated exchange rate from *_updatemHypePrice()* only takes effect starting from the subsequent block. This delay would prevent attackers from sandwiching the *_updatemHypePrice()* transaction with their stake and unstake transactions within the same block, effectively neutralizing the exploit.

3.3.7 [M-4] Potential Risks Associated with Centralization

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	High	Low	Acknowledged

The SpinUp protocol relies on multiple privileged accounts that possess extensive control over critical operations, introducing notable centralization risks. These accounts, assigned distinct roles, can unilaterally influence the protocol's functionality and integrity. Below are key examples of privileged functions and their associated roles:

- Admin Role: Can grant additional privileged roles, configure essential protocol parameters, and withdraw all *HYPE* tokens from the protocol.
- Oracle Role: Controls the exchange rate between *mHYPE* and *HYPE*, directly impacting token economics.
- Minter Role: Authorized to mint *hHYPE* tokens, affecting token supply.
- Burner Role: Permitted to burn *hHYPE* tokens, influencing circulating supply.

This concentration of power in privileged accounts creates a dependency on their security and trustworthiness.

hyperpie - HyperpieWithdrawManager.sol

```
350  function emergencyWithdraw(uint256 amount, address recipient) external onlyDefaultAdmin {
351      if (recipient == address(0)) revert InvalidDestination();
352      (bool success,) = payable(recipient).call{ value: amount }("");
353      if (!success) revert TransferFailed();
354      emit EmergencyWithdraw(amount, recipient);
355  }
```

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

Response By Team This issue has been acknowledged by the team.

3.3.8 [L-1] Emission of UpdatedHyperpieConfig() Event in initialize()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Coding Practices	Low	Low	Addressed

The SpinUp protocol defines an *UpdatedHyperpieConfig()* event to log changes to the *hyperpieConfig* state variable, promoting transparency and auditability of configuration updates. However, this event is not emitted in the *initialize()* functions of the PriceProvider and HyperpieWithdrawManager contracts.

The omission of *UpdatedHyperpieConfig()* events in these initialization functions diminishes the protocol's transparency and auditability. Off-chain systems or external monitors depending on these events to track *hyperpieConfig* updates may miss or fail to verify configuration changes during initialization, which could result in inconsistencies or undetected misconfigurations.

hyperpie - PriceProvider.sol

```
26  function initialize(address hyperpieConfigAddr) external initializer    {
27      UtilLib.checkNonZeroAddress(hyperpieConfigAddr);
28
29      hyperpieConfig = IHyperpieConfig(hyperpieConfigAddr);
30
31      rateIncreaseLimit = 100; // 1% limit
32      rateChangeWindowLimit = 2 hours;
33  }
```

Remediation Ensure the *UpdatedHyperpieConfig()* event is emitted after successfully setting the *hyperpieConfig* state variable in the *initialize()* functions of both the PriceProvider and HyperpieWithdrawManager contracts.

3.3.9 [L-2] Inconsistent Role Usage in mHYPE::pause()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
mHYPE.sol	Coding Practices	Low	Low	Addressed

The mHYPE contract employs a *Manager* role to control the *pause()* function, enabling the pausing of contract operations. However, other contracts within the SpinUp ecosystem, such as the HyperpieStaking contract, utilize a *Pauser* role for the same purpose. This inconsistency in role design, as shown in the provided code snippet, may introduce unnecessary complexity and management challenges for protocol governance and access control.

Specifically, the *mHYPE::pause()* function is restricted by the *onlyHyperpieManager* modifier, whereas a standardized *onlyPauser* modifier is used in other contracts for pausing functionality.

hyperpie - mHYPE.sol

```
103  function pause() external onlyHyperpieManager {
104      _pause();
105  }
```

hyperpie - HyperpieStaking.sol

```
130  function pause() external onlyPauser {
131      _pause();
132  }
```

Remediation Refactor the mHYPE contract to align with the ecosystem's standard by replacing the *onlyHyperpieManager* modifier with the *onlyPauser* modifier for the *pause()* function.

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	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://x.com/AstraSecAI