AstraSec

BlackHaven
Security Audit Report

February 27, 2026

Contents

1 Introduction

1.1 About BlackHaven

1.2 Source Code

1.3 Revision History

2 Overall Assessment

3 Vulnerability Summary

3.1 Overview

3.2 Security Level Reference

3.3 Vulnerability Details

4 Appendix

4.1 About AstraSec

4.2 Disclaimer

4.3 Contact

1 Introduction

1.1 About BlackHaven

BlackHaven is a decentralized protocol that issues RBT (Reserve Backed Token), a token backed by multiple reserve assets. The protocol maintains a 1:1 backing ratio and captures premium when the market value exceeds the backing value (mNAV > 1x).

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/blackhaven-xyz/blackhaven-factory.git
- Commit: bb1cdb7

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

- https://github.com/blackhaven-xyz/blackhaven-factory.git
- Commit: bf8be82

Please note this audit covers the RBT.sol, Minter.sol, BAM.sol BackingCalculator.sol, Bond.sol, RBTNote.sol and LiquidityManager.sol contracts.
This audit focuses on the security of the contracts themselves. The correctness and reasoning of the economic model itself are not within the scope of this audit. Moreover, we assume the price oracle is robust and reliable, and that asset prices are provided in a timely manner. The implementation of the price oracle itself is not included in the scope of this audit.

1.3 Revision History

Version	Date	Description
v1.0	February 27, 2026	Initial Audit

2 Overall Assessment

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

3 Vulnerability Summary

3.1 Overview

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

M-1	Potential Risks Associated with Centralization
L-1	Potential Sandwich/MEV Attack for increaseLiquidityCurrentRange()
~~L-2~~	Integration of Non-Standard ERC20 Tokens
L-3	Transferable Note NFT Allows Unlimited Bond Discount Sharing

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 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

3.3.1 [M-1] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	High	Low	Acknowledged

The BlackHaven 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:

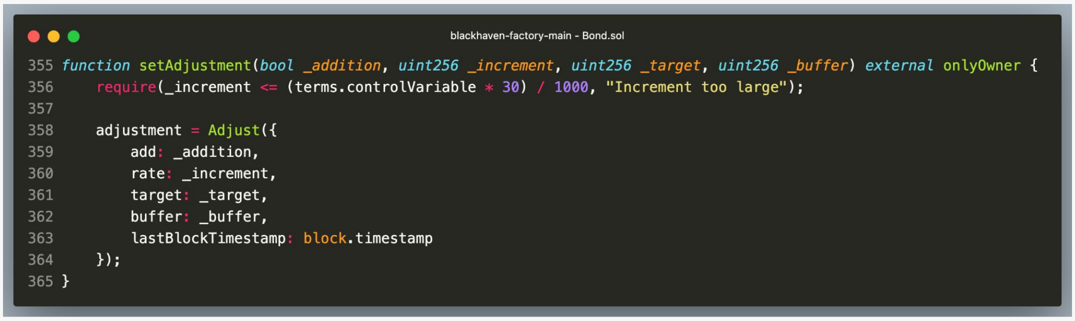
➔ **RBTBonding owner:** Controls bond pricing and limits, discount configuration, distribution of extra minted RBT between backing/POL/team/rewards, and key addresses such as rbtNote, liquidityManager, teamWallet, and rewardWallet.

➔ **Minter owner:** Whitelists and de-whitelists contracts that are allowed to mint RBT via allowMinter() / disallowMinter(), indirectly controlling all RBT minting through the Minter contract.

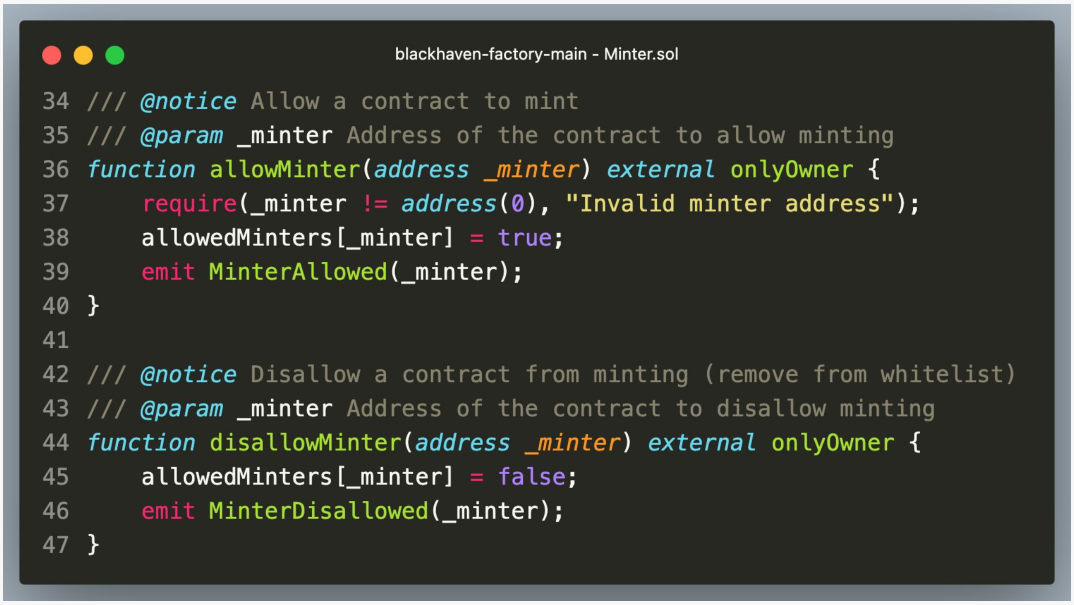
➔ **BackingCalculator owner:** Selects the backing storage address and oracles for RBT and backing tokens, thereby determining reported NAV/mNAV and backing accounting.

➔ **LiquidityManager owner / feeAddress:** Decides how protocol-owned Uniswap V3 liquidity is positioned or withdrawn, where trading fees are sent, and can move ERC20 tokens held by the manager.

➔ **RBTNote owner:** Governs the fixed-term note system that determines user yield and bond discount eligibility, and can withdraw non-RBT tokens from the Note contract.

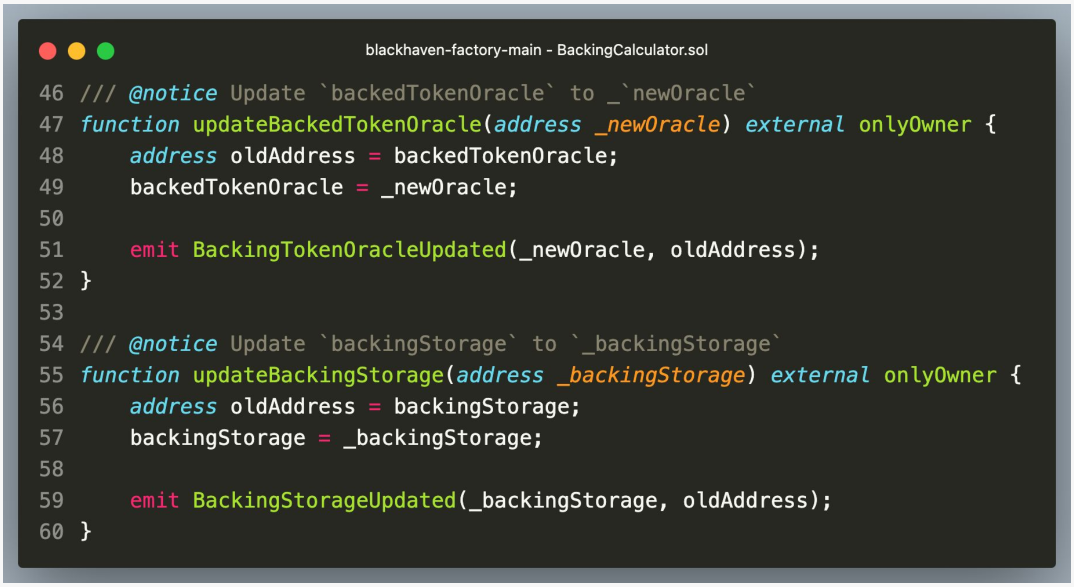

blackhaven-factory-main - Bond.sol

```solidity
function setAdjustment(bool _addition, uint256 _increment, uint256 _target, uint256 _buffer) external onlyOwner {
    require(_increment <= (terms.controlVariable * 30) / 1000, "Increment too large");

    adjustment = Adjust({
        add: _addition,
        rate: _increment,
        target: _target,
        buffer: _buffer,
        lastBlockTimestamp: block.timestamp
    });
}
```



blackhaven-factory-main - Minter.sol

```
/// @notice Allow a contract to mint
/// @param _minter Address of the contract to allow minting
function allowMinter(address _minter) external onlyOwner {
    require(_minter != address(0), "Invalid minter address");
    allowedMinters[_minter] = true;
    emit MinterAllowed(_minter);
}

/// @notice Disallow a contract from minting (remove from whitelist)
/// @param _minter Address of the contract to disallow minting
function disallowMinter(address _minter) external onlyOwner {
    allowedMinters[_minter] = false;
    emit MinterDisallowed(_minter);
}
```



blackhaven-factory-main - BackingCalculator.sol

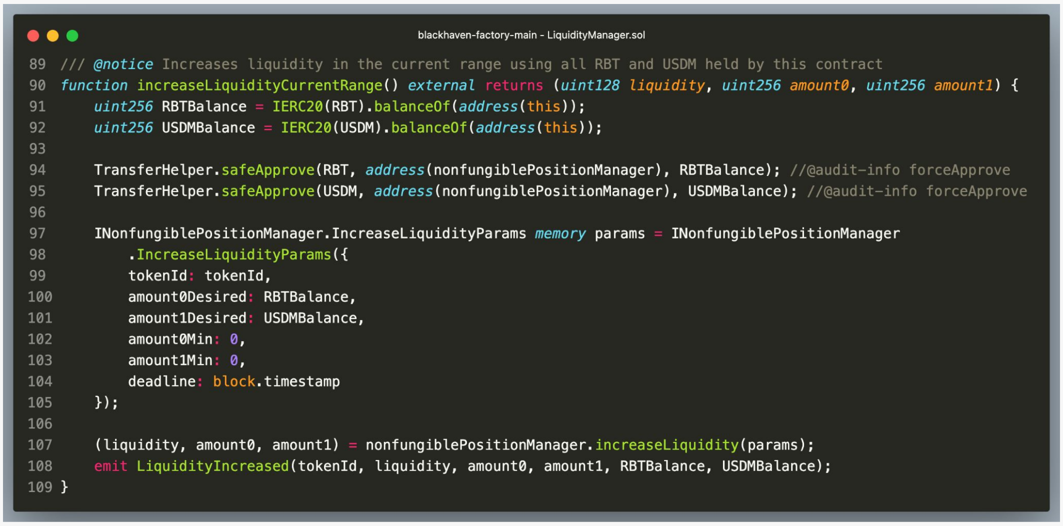
```
/// @notice Update `backedTokenOracle` to _`newOracle`
function updateBackedTokenOracle(address _newOracle) external onlyOwner {
    address oldAddress = backedTokenOracle;
    backedTokenOracle = _newOracle;

    emit BackingTokenOracleUpdated(_newOracle, oldAddress);
}

/// @notice Update `backingStorage` to `_backingStorage`
function updateBackingStorage(address _backingStorage) external onlyOwner {
    address oldAddress = backingStorage;
    backingStorage = _backingStorage;

    emit BackingStorageUpdated(_backingStorage, oldAddress);
}
```

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 confirmed by the team.

3.3.2 [L-1] Potential Sandwich/MEV Attack for increaseLiquidityCurrentRange()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LiquidityManager.sol	Time and State	Low	Medium	Acknowledged

The increaseLiquidityCurrentRange() function adds all RBT and USDM held by the contract as Uniswap V3 liquidity while setting amount0Min and amount1Min to 0. This removes any slippage protection: the trade will succeed even if the pool price is heavily manipulated. An attacker or MEV bot can front-run this call, push the price to an extreme level, let the contract add liquidity at a very poor rate, and then back-run to restore the price and capture the profit, causing potentially significant losses to the contract's funds.


blackhaven-factory-main - LiquidityManager.sol

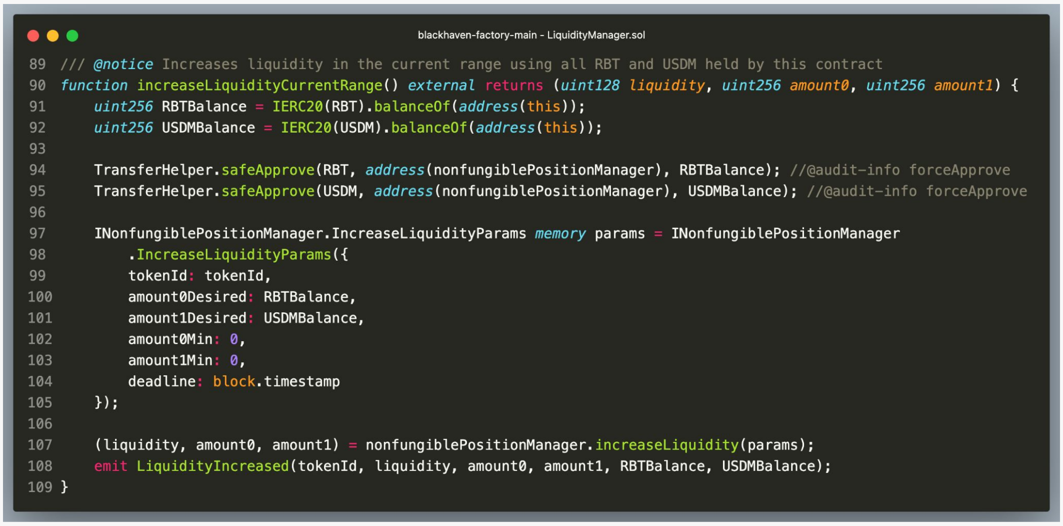
```
/// @notice Increases liquidity in the current range using all RBT and USDM held by this contract
function increaseLiquidityCurrentRange() external returns (uint128 liquidity, uint256 amount0, uint256 amount1) {
    uint256 RBTBalance = IERC20(RBT).balanceOf(address(this));
    uint256 USDMBalance = IERC20(USDM).balanceOf(address(this));

    TransferHelper.safeApprove(RBT, address(nonfungiblePositionManager), RBTBalance); //@audit-info forceApprove
    TransferHelper.safeApprove(USDM, address(nonfungiblePositionManager), USDMBalance); //@audit-info forceApprove

    INonfungiblePositionManager.IncreaseLiquidityParams memory params = INonfungiblePositionManager
        .IncreaseLiquidityParams({
        tokenId: tokenId,
        amount0Desired: RBTBalance,
        amount1Desired: USDMBalance,
        amount0Min: 0,
        amount1Min: 0,
        deadline: block.timestamp
    });

    (liquidity, amount0, amount1) = nonfungiblePositionManager.increaseLiquidity(params);
    emit LiquidityIncreased(tokenId, liquidity, amount0, amount1, RBTBalance, USDMBalance);
}
```

Remediation Add appropriate slippage protection when providing liquidity to prevent unfavorable execution.

3.3.3 [L-2] Integration of Non-Standard ERC20 Tokens

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Business Logic	Medium	Low	Addressed

In LiquidityManager.increaseLiquidityCurrentRange(), the contract calls safeApprove() for RBT and USDM with the full token balances before interacting with nonfungiblePositionManager. For USDT-like tokens that require allowance to be set to zero before being updated, invoking safeApprove() when a non-zero allowance already exists will revert, causing increaseLiquidityCurrentRange() to fail even when the operation would otherwise be valid. To ensure compatibility with such non-standard ERC20 tokens and avoid unintended reverts, these calls should be replaced with forceApprove(), and, more generally, raw transfer()/transferFrom() usages in the codebase should be replaced with safeTransfer()/safeTransferFrom() where applicable.


blackhaven-factory-main - LiquidityManager.sol

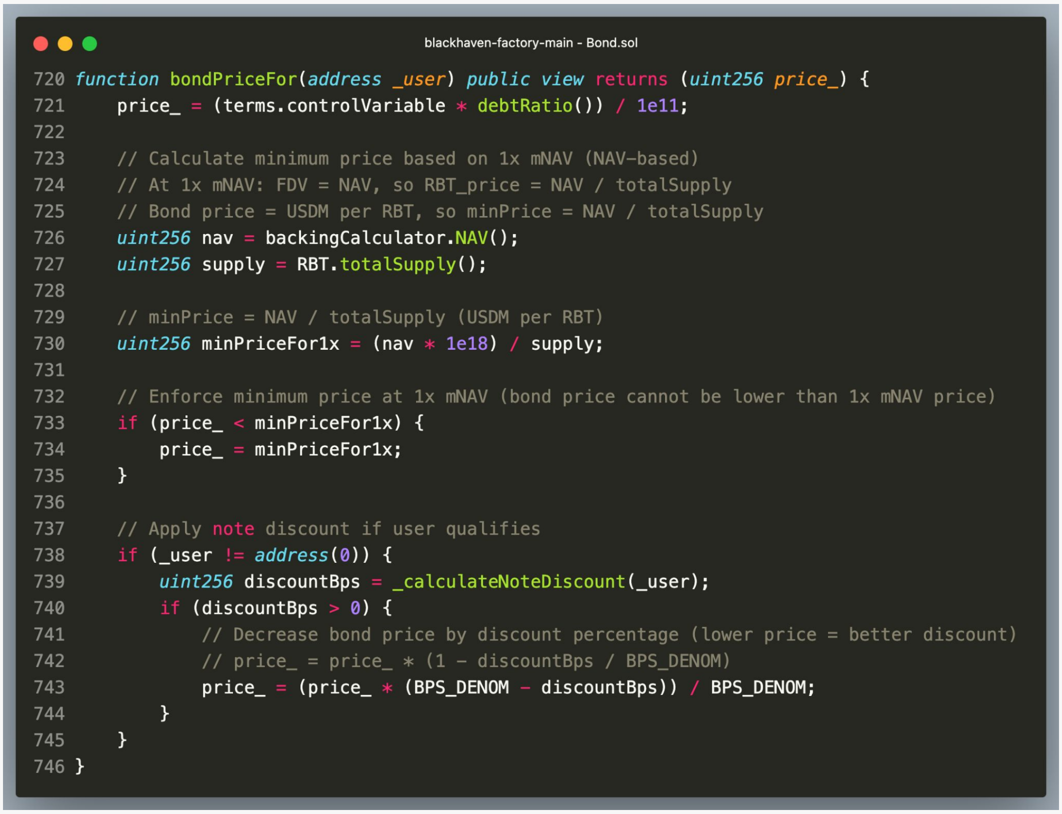
```solidity
/// @notice Increases liquidity in the current range using all RBT and USDM held by this contract
function increaseLiquidityCurrentRange() external returns (uint128 liquidity, uint256 amount0, uint256 amount1) {
    uint256 RBTBalance = IERC20(RBT).balanceOf(address(this));
    uint256 USDMBalance = IERC20(USDM).balanceOf(address(this));

    TransferHelper.safeApprove(RBT, address(nonfungiblePositionManager), RBTBalance); //@audit-info forceApprove
    TransferHelper.safeApprove(USDM, address(nonfungiblePositionManager), USDMBalance); //@audit-info forceApprove

    INonfungiblePositionManager.IncreaseLiquidityParams memory params = INonfungiblePositionManager
        .IncreaseLiquidityParams({
        tokenId: tokenId,
        amount0Desired: RBTBalance,
        amount1Desired: USDMBalance,
        amount0Min: 0,
        amount1Min: 0,
        deadline: block.timestamp
    });

    (liquidity, amount0, amount1) = nonfungiblePositionManager.increaseLiquidity(params);
    emit LiquidityIncreased(tokenId, liquidity, amount0, amount1, RBTBalance, USDMBalance);
}
```

Remediation Use forceApprove() instead of safeApprove() and replace all transfer()/transferFrom() calls with safeTransfer()/safeTransferFrom() to ensure compatibility with USDT-like non-standard ERC20 tokens and avoid unintended reverts

3.3.4 [L-3] Transferable Note NFT Allows Unlimited Bond Discount Sharing

Target	Category	IMPACT	LIKELIHOOD	STATUS
Bond.sol	Business Logic	Medium	Low	Acknowledged

The bondPriceFor() function applies a discount of up to 2.5% (250 bps) based on the caller's RBT Note holdings via _calculateNoteDiscount(_user). However, because RBT Notes are transferable ERC-721 NFTs, a qualifying holder (e.g., Alice) can transfer the NFT to another address (e.g., Bob) immediately before Bob calls deposit(), allowing Bob to receive the full discount even if he does not personally qualify. Bob can then transfer the NFT back to Alice right after the deposit. This enables a simple and repeatable "discount rental" attack that lets the maximum discount be shared across an arbitrary number of bond purchases, undermining the intended incentive for long-term Note holders and causing material revenue/backing leakage to the protocol.


blackhaven-factory-main - Bond.sol

```
function bondPriceFor(address _user) public view returns (uint256 price_) {
    price_ = (terms.controlVariable * debtRatio()) / 1e11;

    // Calculate minimum price based on 1x mNAV (NAV-based)
    // At 1x mNAV: FDV = NAV, so RBT_price = NAV / totalSupply
    // Bond price = USDM per RBT, so minPrice = NAV / totalSupply
    uint256 nav = backingCalculator.NAV();
    uint256 supply = RBT.totalSupply();

    // minPrice = NAV / totalSupply (USDM per RBT)
    uint256 minPriceFor1x = (nav * 1e18) / supply;

    // Enforce minimum price at 1x mNAV (bond price cannot be lower than 1x mNAV price)
    if (price_ < minPriceFor1x) {
        price_ = minPriceFor1x;
    }

    // Apply note discount if user qualifies
    if (_user != address(0)) {
        uint256 discountBps = _calculateNoteDiscount(_user);
        if (discountBps > 0) {
            // Decrease bond price by discount percentage (lower price = better discount)
            // price_ = price_ * (1 - discountBps / BPS_DENOM)
            price_ = (price_ * (BPS_DENOM - discountBps)) / BPS_DENOM;
        }
    }
}
```

Remediation Redesign the discount mechanism to prevent the discount eligibility checks from being bypassed.

4 Appendix

4.1 About AstraSec

AstraSec is a premier blockchain security firm dedicated to delivering high-quality auditing services for blockchain-based protocols. Composed of veteran security researchers with extensive experience across the DeFi landscape, our team maintains an unwavering commitment to excellence and precision. AstraSec’s comprehensive methodology and deep understanding of blockchain architecture enable us to ensure protocol resilience, making us a trusted partner for our clients.

4.2 Disclaimer

4.3 Contact

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