AstraSec

Cakepie
Security Audit Report

February 13, 2025

Contents

1 Introduction

1.1 About Cakepie

1.2 Source Code

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 Cakepie

Cakepie is an advanced SubDAO created by the Magpie Kitchen to enhance the long-term sustainability of PancakeSwap's veCAKE design. The primary objective of Cakepie is to accumulate CAKE tokens and lock them as veCAKE, helping to decrease its circulating supply. This allows Cakepie to capitalize on PancakeSwap's structure, optimizing governance power and offering passive income opportunities for DeFi users.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/cakepie_contract.git
- CommitID: a9fcfc9

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

- https://github.com/magpiexyz/cakepie_contract.git
- CommitID: 81ca6be

Please note that the scope of this audit is as follows:

- PancakeStakingBaseUpg.sol, PancakeStakingBNBChain.sol
- PancakeStakingSideChain.sol, PancakeStakingLib.sol
- PancakeV3Helper.sol, PancakeV2LPHelper.sol
- PancakeAMLHelper.sol, RewardDistributor.sol
- MasterCakepie.sol, SmartCakeConvertor.sol
- mCakeConvertorBaseUpg.sol, mCakeConvertorBNBChain.sol
- CakepieBribeManager.sol, PancakeVoteManager.sol
- VLCakepie.sol, mCakeSV.sol
- BaseRewardPoolV3.sol, StreamRewarder.sol
- vlStreamRewarder.sol, Cakepie.sol, CakepieCCIPBridge.sol

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the Cakepie protocol. Throughout this audit, we identified a total of 6 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	2	1	–	1
Low	4	–	–	4
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~~	Improper initializer() Use in __PancakeStakingBaseUpg_init()
M-2	Potential Risks Associated with Centralization
~~L-1~~	Improper lastHarvestTime Update in _depositTokens()
~~L-2~~	Revisited Logic of _updateVoteAndCheck()
~~L-3~~	Improper Fee Claim in manualClaimFees()
~~L-4~~	Revisited Excess ETH Refund in tokenTransfer()

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:

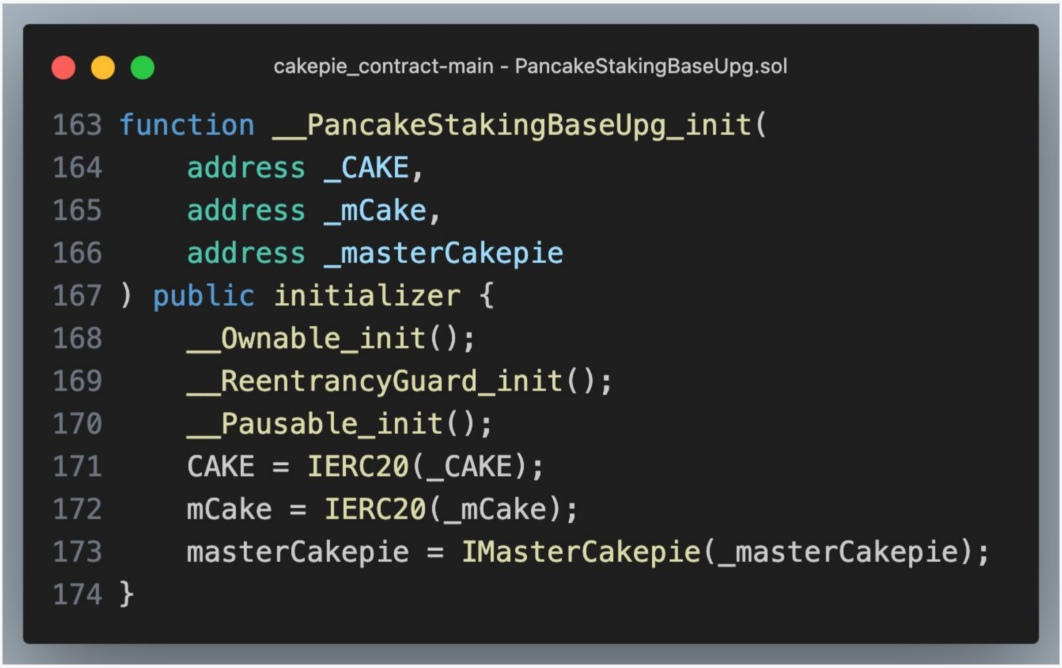
Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security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

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
PancakeStakingBaseUpg.sol StreamRewarder.sol	Business Logic	Medium	Medium	Addressed

While reviewing the PancakeStakingBaseUpg contract, we identify a vulnerability in the __PancakeStakingBaseUpg_init() function due to the incorrect use of the initializer modifier. Starting from OpenZeppelin 4.4, initializer is intended only for top-level initialization, and sub-level initializers like __PancakeStakingBaseUpg_init() should use onlyInitializing. This mistake can cause failures during deployment or upgrades. Furthermore, consistent use of the same OpenZeppelin version throughout the protocol is crucial to avoid storage conflicts and initialization issues. Additionally, the StreamRewarder::__StreamRewarder_init() function shares a similar issue.



cakepie_contract-main - PancakeStakingBaseUpg.sol

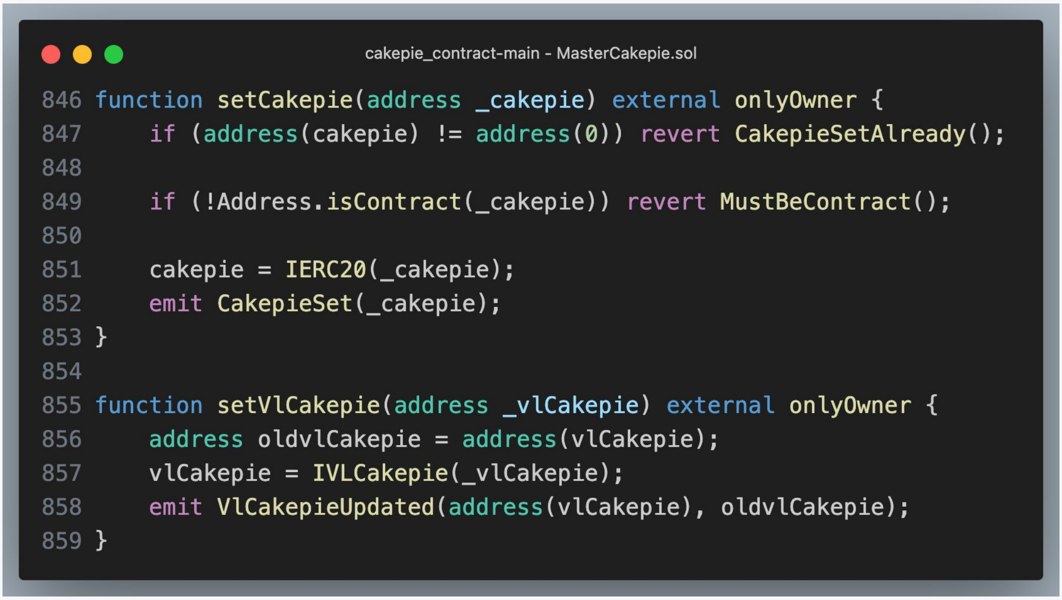
```
function __PancakeStakingBaseUpg_init(
    address _CAKE,
    address _mCake,
    address _masterCakepie
) public initializer {
    __Ownable_init();
    __ReentrancyGuard_init();
    __Pausable_init();
    CAKE = IERC20(_CAKE);
    mCake = IERC20(_mCake);
    masterCakepie = IMasterCakepie(_masterCakepie);
}
```

Remediation Replace initializer with onlyInitializing in sub-initialization functions.

3.3.2 [M-2] Potential Risks Associated with Centralization

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	High	Low	Acknowledged

In the Cakepie protocol, the existence of a series of privileged accounts introduces centralization risks, as they hold significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the privileged accounts.



```
function setCakepie(address _cakepie) external onlyOwner {
    if (address(cakepie) != address(0)) revert CakepieSetAlready();

    if (!Address.isContract(_cakepie)) revert MustBeContract();

    cakepie = IERC20(_cakepie);
    emit CakepieSet(_cakepie);
}

function setVlCakepie(address _vlCakepie) external onlyOwner {
    address oldvlCakepie = address(vlCakepie);
    vlCakepie = IVLCakepie(_vlCakepie);
    emit VlCakepieUpdated(address(vlCakepie), oldvlCakepie);
}
```

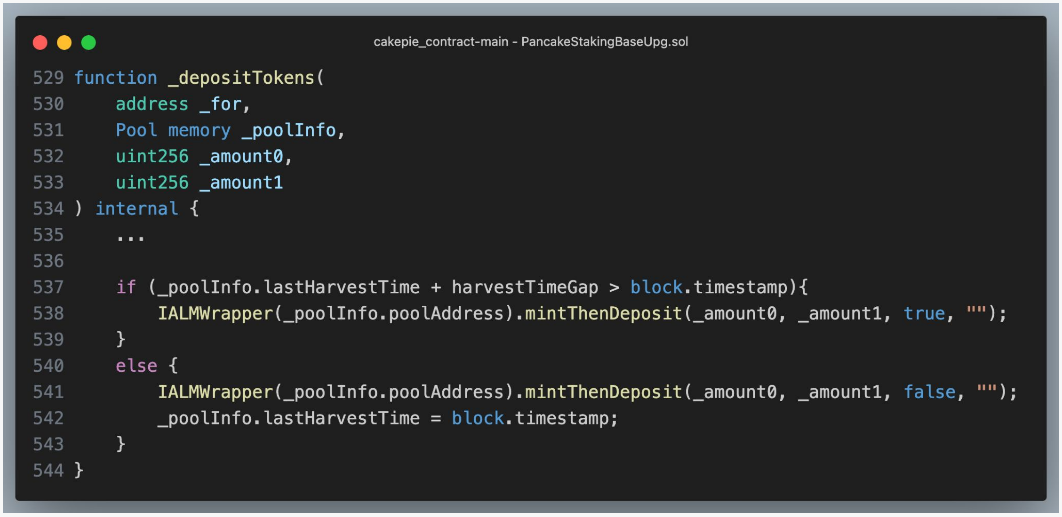
Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged accounts. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

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

3.3.3 [L-1] Improper lastHarvestTime Update in _depositTokens()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
PancakeStakingBaseUpg.sol	Business Logic	Low	Low	Addressed

While reviewing the _depositTokens() function, we identify a flaw with the handling of the lastHarvestTime parameter. The _poolInfo variable is passed as a memory variable, which means updates to _poolInfo.lastHarvestTime (line 542) within the function do not persist in the contract's state. As a result, the harvestTimeGap restriction (line 537), designed to limit the frequency of harvesting, fails to function as intended.



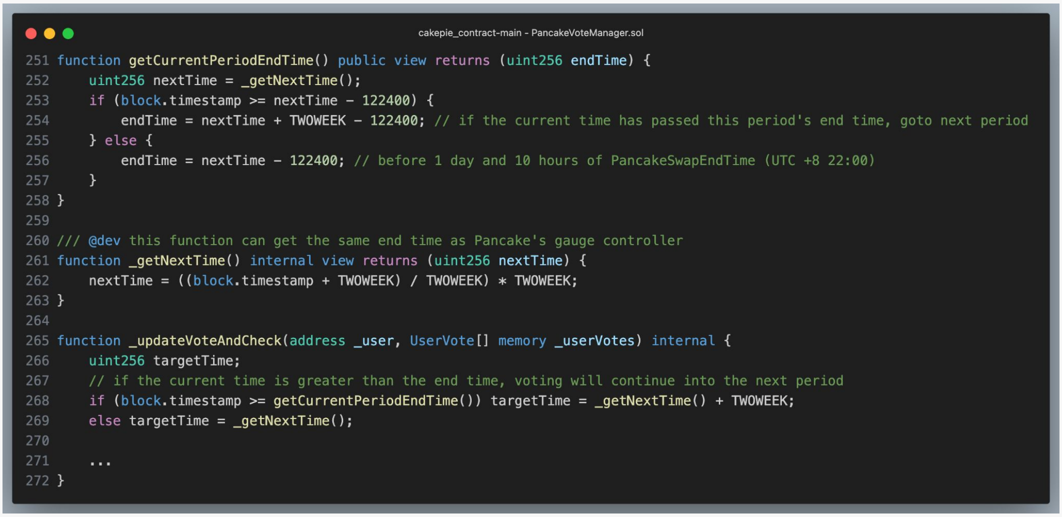
```
529 function _depositTokens(
530     address _for,
531     Pool memory _poolInfo,
532     uint256 _amount0,
533     uint256 _amount1
534 ) internal {
535     ...
536
537     if (_poolInfo.lastHarvestTime + harvestTimeGap > block.timestamp){
538         IALMWrapper(_poolInfo.poolAddress).mintThenDeposit(_amount0, _amount1, true, "");
539     }
540     else {
541         IALMWrapper(_poolInfo.poolAddress).mintThenDeposit(_amount0, _amount1, false, "");
542         _poolInfo.lastHarvestTime = block.timestamp;
543     }
544 }
```

Remediation Pass _poolInfo as a storage reference instead of memory to ensure that update to lastHarvestTime persist in the contract's state.

3.3.4 [L-2] Revisited Logic of _updateVoteAndCheck()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
PancakeVoteManager.sol	Business Logic	Low	Low	Addressed

While reviewing the _updateVoteAndCheck() function, we identify a flaw in the condition block.timestamp >= getCurrentPeriodEndTime() (line 268), which never evaluates to true. As a result, targetTime is always assigned to the current period, preventing the function from transitioning votes to the next period as intended. This misalignment disrupts the protocol's governance mechanism by misattributing votes, leading to potential inaccuracies in decision-making and governance outcomes.



cakepie_contract-main - PancakeVoteManager.sol

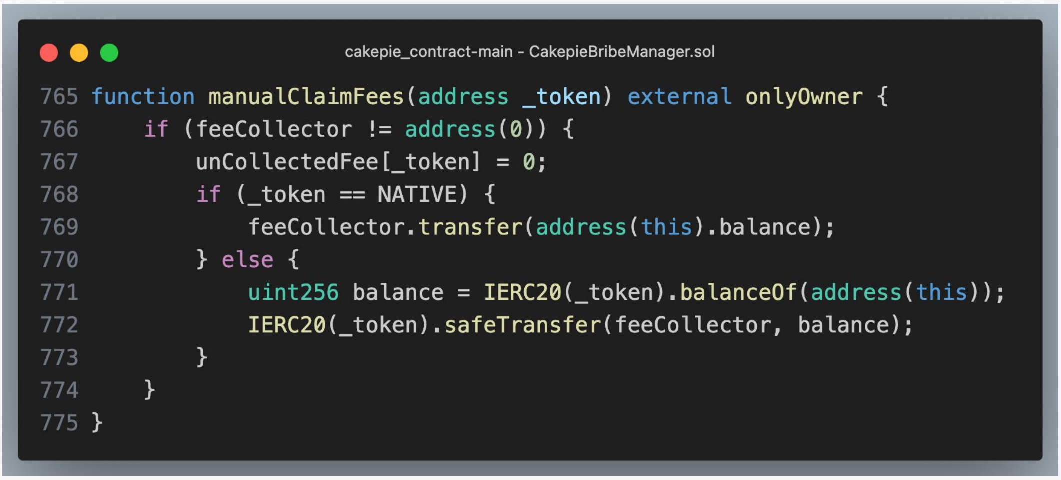
```
251 function getCurrentPeriodEndTime() public view returns (uint256 endTime) {
252     uint256 nextTime = _getNextTime();
253     if (block.timestamp >= nextTime - 122400) {
254         endTime = nextTime + TWOWEEK - 122400; // if the current time has passed this period's end time, goto next period
255     } else {
256         endTime = nextTime - 122400; // before 1 day and 10 hours of PancakeSwapEndTime (UTC +8 22:00)
257     }
258 }
259
260 /// @dev this function can get the same end time as Pancake's gauge controller
261 function _getNextTime() internal view returns (uint256 nextTime) {
262     nextTime = ((block.timestamp + TWOWEEK) / TWOWEEK) * TWOWEEK;
263 }
264
265 function _updateVoteAndCheck(address _user, UserVote[] memory _userVotes) internal {
266     uint256 targetTime;
267     // if the current time is greater than the end time, voting will continue into the next period
268     if (block.timestamp >= getCurrentPeriodEndTime()) targetTime = _getNextTime() + TWOWEEK;
269     else targetTime = _getNextTime();
270
271     ...
272 }
```

Remediation Update the condition to ensure accurate period handling: block.timestamp >= _getNextTime() - 122400 (line 268).

3.3.5 [L-3] Improper Fee Claim in manualClaimFees()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
CakepieBribeManager.sol	Business Logic	Low	Low	Addressed

The manualClaimFees() function mistakenly transfers the entire balance of the specified token from the contract, instead of using the amount recorded in unCollectedFee[_token]. This can result in unrelated tokens held in the contract being incorrectly transferred to the feeCollector, violating the intended functionality. Such behavior may lead to accidental or unauthorized transfers of funds not associated with the fee system.



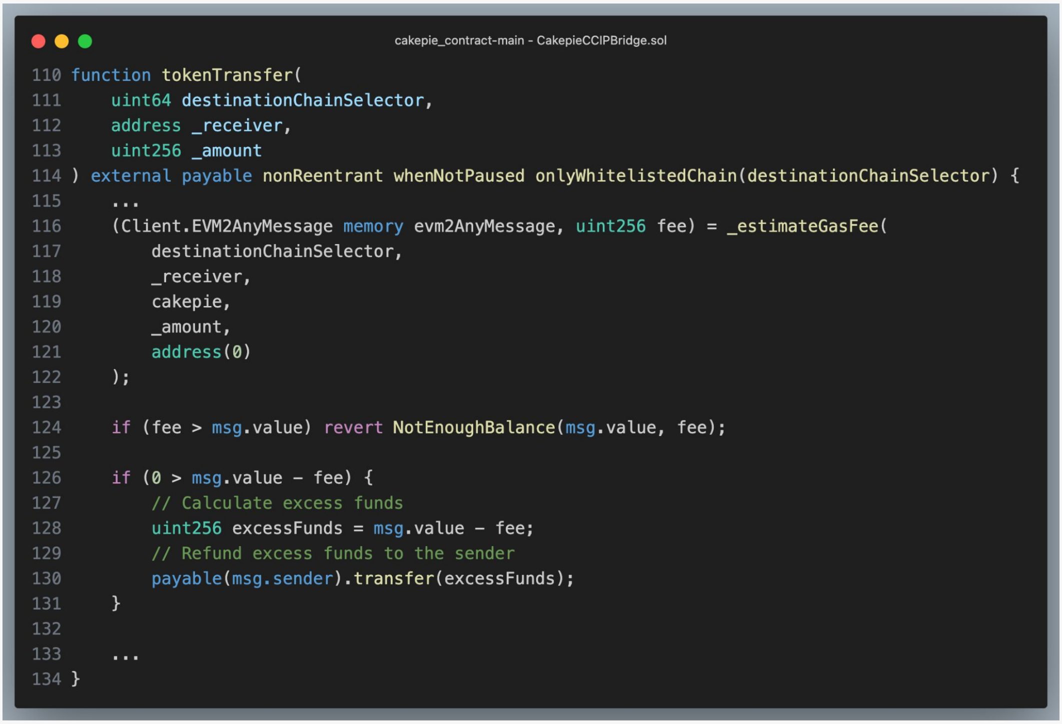
```
function manualClaimFees(address _token) external onlyOwner {
    if (feeCollector != address(0)) {
        unCollectedFee[_token] = 0;
        if (_token == NATIVE) {
            feeCollector.transfer(address(this).balance);
        } else {
            uint256 balance = IERC20(_token).balanceOf(address(this));
            IERC20(_token).safeTransfer(feeCollector, balance);
        }
    }
}
```

Remediation Improve the implementation of the manualClaimFees() function to ensure that only the amount recorded in unCollectedFee[_token] is claimed.

3.3.6 [L-4] Revisited Excess ETH Refund in tokenTransfer()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
CakepieCCIPBridge.sol	Business Logic	Low	Low	Addressed

The tokenTransfer() function is intended to refund users any excess ETH sent beyond the required bridge fee. However, the refund logic contains an incorrect condition: if (0 > msg.value - fee) (line 126). This condition will never evaluate to true because msg.value is always greater than or equal to fee (validated earlier in the function, line 124). As a result, the refund logic is bypassed, and users are not reimbursed for any excess ETH sent above the bridge fee.



```
110 function tokenTransfer(
111     uint64 destinationChainSelector,
112     address _receiver,
113     uint256 _amount
114 ) external payable nonReentrant whenNotPaused onlyWhitelistedChain(destinationChainSelector) {
115     ...
116     (Client.EVM2AnyMessage memory evm2AnyMessage, uint256 fee) = _estimateGasFee(
117         destinationChainSelector,
118         _receiver,
119         cakepie,
120         _amount,
121         address(0)
122     );
123
124     if (fee > msg.value) revert NotEnoughBalance(msg.value, fee);
125
126     if (0 > msg.value - fee) {
127         // Calculate excess funds
128         uint256 excessFunds = msg.value - fee;
129         // Refund excess funds to the sender
130         payable(msg.sender).transfer(excessFunds);
131     }
132
133     ...
134 }
```

Remediation Correct the refund logic to if (msg.value > fee) (line 126) to accurately calculate and refund excess ETH to the user.

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://twitter.com/AstraSecAI