BINENET

Smart Contract Audit Report

BNB Smart Chain

CaryPact Project

V 1.0

SC.5475811c8fe6

Nov. 30th, 2025

Table Of Content

1 Report Overview ..... - 2 -
2 Asset Management Security Assessment ..... - 3 -
3 Audit Overview ..... - 4 -
3.1 Project Information ..... - 4 -
3.2 Audit Information ..... - 5 -
3.3 External Visibility Analysis ..... - 5 -
3.4 Audit Process ..... - 6 -
4 Audit Categories ..... - 7 -
5 Security Finding Details ..... - 9 -
5.1 Missing Zero Address Check: setReceiveTokenAccount function does not validate address validity - 9 -
5.2 Logic Error: buyVip function allows vip=4 but undefined corresponding price and reward rate ..... - 9 -
5.3 Profit Withdrawal Function in drawProfit ..... - 10 -
5.4 Invite Code Registration Function in registerWithInviteCode ..... - 11 -
5.5 Admin Invite Code Management Function in setUserInviteCode ..... - 12 -
5.6 Backward Compatible Inviter Registration Function in addMemberInviter ..... - 12 -
6 Explanation of Vulnerability Rating ..... - 14 -
7 Statement ..... - 16 -
8 About Binenet ..... - 17 -

BINENET

1 Report Overview

Binenet security audit has identified 0 security vulnerabilities across the CaryPact smart contracts.

Contract Code	Function	Security Level	Status	Fix Result
---	---	---	---	---

***Risk Description:** No obvious security issues found based on business logic.

BINENET

2 Asset Management Security Assessment

Asset Type	Function	Security Level
User Mortgage Token Assets	---	---
Users Mortgage Platform Currency Assets	---	---

***Description:** Check the management security of digital currency assets transferred by users in the contract business logic. Observe whether there are security risks that may cause the loss of customer funds, such as the digital currency assets transferred into the contract are incorrectly recorded or transferred out by mistake.

BINENET

3 Audit Overview

3.1 Project Information

CaryPact is a blockchain-based node pool and membership platform built on smart contracts that enables users to participate in a tiered VIP membership system using USDT (Tether). The platform operates through a decentralized NodePool contract that manages user registrations, VIP level purchases, and reward distributions. Users can purchase VIP memberships at multiple tier levels, with each tier providing access to the platform's node pool services and profit-sharing mechanisms. The system implements a whitelist-based access control mechanism, ensuring that only approved addresses can participate in the platform's core functions, providing an additional layer of security and community management.

The platform features a comprehensive referral and reward system that incentivizes user growth through a unique invitation code mechanism. Each registered user receives a unique alphanumeric invitation code that can be shared with others to build an invitation network. When new users register using an invitation code, they establish an inviter-invitee relationship, and the inviter receives a direct referral reward based on a percentage of the new member's VIP purchase amount. This reward is automatically credited to the inviter's profit pool, which can be withdrawn at any time. The system tracks various user statistics including total USDT spent, VIP level, accumulated profits, and the number of successful referrals, creating a transparent and traceable reward ecosystem.

The platform's smart contract architecture includes sophisticated administrative controls and fund management capabilities. Contract owners can configure VIP prices, reward rates, withdrawal fees, and manage the whitelist system through dedicated management functions. The system supports profit withdrawals with configurable fee structures, allowing users to extract their accumulated rewards while maintaining platform sustainability. Additionally, the contract includes emergency fund extraction functions for administrators to manage contract balances, though these require careful implementation

to prevent potential security vulnerabilities. The entire system is designed to operate autonomously on the blockchain, with all transactions and state changes recorded immutably, ensuring transparency and trust in the platform's operations.

3.2 Audit Information

Project Name	CaryPact
Platform	BNBChain
Audit Scope	NodePool.sol#03c9ca9e69c73b1318896a70ac25c345
Website	https://app.carypact.com

3.3 External Visibility Analysis

Function	Visibility	State Change	Modifier	Payable	Description
setIsWhitelistedOn	external	true	---	---	---
setManager	public	true	---	---	---
removeManager	public	true	---	---	---
setWhiteList	external	true	---	---	---
removeWhiteList	external	true	---	---	---
registerWithInviteCode	public	true	---	---	---
addMemberInviter	public	true	---	---	---
setReceiveTokenAccount	external	true	---	---	---
setWithdrawFee	external	true	---	---	---
initFirstInviteCode	public	true	---	---	---
setUserInviteCode	public	true	---	---	---
buyVip	public	true	---	---	---
drawProfit	public	true	---	---	---

setVipPrice	public	true	---	---	---
setVipDirect RewardRate	public	true	---	---	---

3.4 Audit Process

Audit time: November 29, 2025 - November 30 2025

Audit methods: Static Analysis, Dynamic Testing, Typical Case Testing and Manual Review.

Audit team: Binenet Security Team.

BINENET

4 Audit Categories

Categories	Subitems	Status	Description
Business Security	Transfer token function	Pass	---
	Mint token and burn token vulnerability	Pass	---
	Contract logic function	Pass	---
	Mining pool deposit and withdrawal function	Pass	---
	Reasonableness of agreement amendment	Pass	---
	Functional design	Pass	---
	Dos caused by time	Pass	---
	Insecure orcles and their design	Pass	---
	Deployer private key leak hazard	Pass	---
General Vulnerability	Compiler version security	Pass	---
	Redundant code	Pass	---
	Use of safemath library	Pass	---
	Not recommended encoding	Pass	---
	Use require/assert mistakely	Pass	---
	Fallback function safety	Pass	---
	tx.origin authentication	Pass	---
	Owner permission control	Pass	---
	Gas consumption detection	Pass	---
	Call injection attack	Pass	---
	Low-level function safety	Pass	---
	Additional token vulnerabilities	Pass	---
	Access control	Pass	---
	Numeric overflow detection	Pass	---

BINENET

	Arithmetic precision error	Pass	---
	Misuse of random number detection	Pass	---
	Unsafe external call	Pass	---
	Variable override	Pass	---
	Uninitialized storage pointer	Pass	---
	Return value call validation	Pass	---
	Transaction order dependent detection	Pass	---
	Timestamp dependent attack	Pass	---
	Denial of service attack detection	Pass	---
	Fake recharge vulnerability detection	Pass	---
	Reentrancy Attack Detection	Pass	---
	Replay attack detection	Pass	---
	Reordering attack detection	Pass	---

BINENET

5 Security Finding Details

5.1 Missing Zero Address Check: setReceiveTokenAccount function does not validate address validity

Severity Level : Info

Lines : NodePool.sol:L228-L230

Description:

The setReceiveTokenAccount function allows the owner to set the token receiving address, but does not check if _receiveTokenAccount is address(0). Although the buyVip function has a require(receiveTokenAccount != address(0), \"not open\") check, if the owner mistakenly sets receiveTokenAccount to the zero address, the entire buyVip functionality will be unavailable until the owner calls this function again to set the correct address. This may cause service interruption. Additionally, if the zero address is set as the receiving address, subsequent transferFrom calls will fail, but the error message may not be clear enough.

```
228     function setReceiveTokenAccount(address _receiveTokenAccount) external onlyOwner {
229         receiveTokenAccount = _receiveTokenAccount;
230     }
```

Recommendations: Add require(_receiveTokenAccount != address(0), "Invalid address") check in the setReceiveTokenAccount function.

Status : Audited.

Fix Result: ---

5.2 Logic Error: buyVip function allows vip=4 but undefined corresponding price and reward rate

Severity Level : Info

Lines : NodePool.sol:L289-L312

Description:

The buyVip function allows the vip parameter to be 4, but the constructor only initializes vipPrice[1-3] and vipDirectRewardRate[1-3], causing the price to be 0 or undefined when vip=4

```
289     function buyVip(uint vip, uint256 amount) public onlyWhiteList{
290         require(vip >= 1 && vip <= 4, "invalid vip");
291         address parent = inviter[msg.sender];
292         require(parent != address(0), "must have inviter");
293         require(receiveTokenAccount != address(0), "not open");
294 
295         address sender = msg.sender;
296         uint256 needVipAmount = vipPrice[vip].mul(amount);
297         usdtToken_.transferFrom(sender, address(receiveTokenAccount), needVipAmount);
298 
299         userInfo_[sender].vipLevel = vip;
```

```
66          vipPrice[1] = usdtDecimal_.mul(1000);
67          vipPrice[2] = usdtDecimal_.mul(5000);
68          vipPrice[3] = usdtDecimal_.mul(10000);
69 
```

Recommendations:

1. Modify the validation logic of the buyVip function to only allow vip to be 1-3.
2. Or initialize vipPrice[4] and vipDirectRewardRate[4] in the constructor.
3. Add checks in the buyVip function to ensure that vipPrice[vip] and vipDirectRewardRate[vip] are not 0.
4. Add range checks in the setVipPrice and setVipDirectRewardRate functions to ensure vip is within the valid range.

Status : Audited.

Fix Result: ---

5.3 Profit Withdrawal Function in drawProfit

Severity Level : Pass

Lines : NodePool.sol:L314-L335

Description:

The drawProfit function allows whitelisted users who have purchased VIP levels to withdraw their accumulated profits from the profitPool. The function checks if the user

has a VIP level, retrieves their profit amount, calculates withdrawal fees if applicable, and transfers the final profit amount to the user. However, this function has a critical reentrancy vulnerability: it updates user.drawedProfit and performs an external token transfer (usdtToken_.transfer) after setting profitPool[sender] to 0, but the order of operations does not strictly follow the checks-effects-interactions pattern. If usdtToken_ is a malicious contract, it could reenter drawProfit during the transfer callback, potentially causing fund loss if profitPool is increased through other means (like buyVip) during reentrancy. The function should use ReentrancyGuard or ensure all state updates complete before external calls.

Recommendations: ---

Status : *Audited.*

Fix Result: ---

5.4 Invite Code Registration Function in registerWithInviteCode

Severity Level : Pass

Lines : NodePool.sol:L192-L211

Description:

The registerWithInviteCode function allows whitelisted users to register with an existing invite code, establishing an inviter-invitee relationship. The function validates that the caller is not a contract, hasn't registered before, the invite code is valid and not self-referential, then sets the inviter relationship, increments the inviter's invite count, and generates a new invite code for the new user. However, the function relies on _generateInviteCode which uses predictable random sources (block.timestamp, block.difficulty), making invite codes potentially predictable. Additionally, the function doesn't prevent re-registration attempts after the initial registration, though the require check should prevent this. The invite code lookup uses a string-to-address mapping, which is gas-intensive but necessary for the invite code system.

Recommendations: ---

Status : *Audited.*

Fix Result: ---

5.5 Admin Invite Code Management Function in setUserInviteCode

Severity Level : Pass

Lines : NodePool.sol:L241-L259, L261-L273

Description:

The setUserInviteCode function has two overloaded versions allowing the owner to manually set invite codes or inviter relationships for users. The first version (lines 241-259) sets a custom invite code string for a user, with checks to ensure the user has no existing code, hasn't invited others, and the new code isn't already in use. The second version (lines 261-273) directly sets the inviter relationship between two addresses and increments the inviter's count. While these functions provide administrative flexibility, they bypass the normal registration flow and could be used to manipulate the invitation system. The functions properly validate addresses and prevent self-invitation, but there's no timelock or multi-sig protection, meaning a compromised owner could arbitrarily modify invitation relationships. Additionally, the first overload correctly validates the user address, but the validation is missing in some edge cases.

Recommendations: ---

Status : *Audited.*

Fix Result: ---

5.6 Backward Compatible Inviter Registration Function in addMemberInviter

Severity Level : Pass

Lines : NodePool.sol:L214-L225

Description:

The addMemberInviter function provides backward compatibility for existing frontend code, allowing whitelisted users to register by providing an inviter's address directly instead of an invite code string. The function validates that the caller is not a contract, hasn't registered before, and the inviter address is valid and not self-referential. It then retrieves the inviter's invite code and calls registerWithInviteCode internally. However, this function has a potential issue: it performs the same contract check and registration validation as registerWithInviteCode, but the contract check (isContract) can be bypassed if called during contract construction (when extcodesize returns 0). Additionally, the function relies on the inviter having a valid invite code, which may not always be the case if the inviter was set up through admin functions. The function essentially acts as a wrapper around registerWithInviteCode, providing a more user-friendly interface for address-based registration, but it doesn't add any additional security checks beyond what registerWithInviteCode already performs.

Recommendations: ---

Status : *Audited.*

Fix Result: ---

BINENET

6 Explanation of Vulnerability Rating

Vulnerability Rating	Rating Description
High Risk Vulnerability	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: overflow、reentrancy 、false recharge， which can cause the value of tokens to be zeroed, or causing false exchanges to lose tokens, or causing losing ETH or tokens, etc; Vulnerabilities that can cause loss of ownership of token contracts, such as: access control flaws of key functions, call injection leading to access control bypass of key functions, etc; Vulnerabilities that can cause token contracts to fail to work properly, such as: denial of service vulnerabilities caused by sending ETH to malicious addresses, and denial of service vulnerabilities caused by gas exhaustion;
Medium Risk Vulnerability	High-risk vulnerabilities that require specific addresses to be triggered, such as overflow that can only be triggered by token contract owners; access control flaws of non-critical functions, logic design flaws that cannot cause direct financial losses, etc;
Low Risk Vulnerability	Vulnerabilities that are difficult to be triggered, vulnerabilities that cause limited harm after triggering, such as overflow vulnerabilities that require a large amount of ETH or tokens to be triggered, vulnerabilities that the attacker cannot directly profit after triggering overflow, and transaction sequence-dependent risks

	triggered by specifying high gas wait;

BINENET

7 Statement

Binenet only issues this report based on the facts that have occurred or existed before the issue of this report, and assumes corresponding responsibilities for it. For the facts that occurred or existed after the issuance, we cannot judge the security status of the smart contract , and we will not be responsible for it.

This report does not include external contract calls , new types of attacks that may appear in the future, and contract upgrades or tampered codes (with the development of the project side, smart contracts may add new pools, new functional modules, new external contract calls, etc.), does not include front-end security and server security.

The documents and materials provided to us by the information provider as of the date of this report.

Binenet assumes that there is no missing, tampered, deleted or concealed information provided. If the information provided is missing, tampered, deleted, concealed or reflected inconsistent with the actual situation, Binenet shall not be liable for any losses and adverse effects resulting therefrom.

BINENET

8 About Binenet

Founded in June 2021, Binenet is a dedicated and pure blockchain security company, focusing on accurate, efficient and intelligent blockchain threat detection and response. Committed to providing users with professional products and dedicated services in the field of blockchain security. Business functions cover penetration testing, code auditing, emergency response, on-chain data monitoring, AML anti-money laundering, etc., covering all aspects of blockchain ecosystem security.

BINENET



Official Website
https://binenet.com

Telegram
https://t.me/binenetxyz

Twitter
https://twitter.com/binenetxyz

E-mail
team@binenet.com