BLOCKSEC

Security Audit
Report for Flypad

Date: December 27, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	FlyFoundation
Target	Flypad

Version History

Version	Date	Description
1.0	December 27, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-12-28 12:23:03+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository [1] of Flypad of FlyFoundation.

Flypad is an ERC20 token issuance and yield management system. Its core component is the FlyBNB contract, which wraps asBNB to enable yield accrual and distribution. The Factory contract is responsible for deploying new tokens via CREATE2, initializing Pancake V3 liquidity pools, and integrating with GoPlus for liquidity locking. Protocol fees are split between token creators and the platform at a 40/60 ratio. Yield generated through FlyBNB rebasing is collected by the YieldController and subsequently allocated between the platform and the BuybackAccumulator. The BuybackAccumulator performs token buyback and burn operations on Pancake V3. Together, these contracts implement an automated workflow for token deployment, liquidity provisioning, yield distribution, and buyback execution.

Note this audit only focuses on the smart contracts in the following directories/files:

- contracts/contracts/*

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
Flypad	`Version 1`	`74ed1e4a4369390be12bc3c784d93c4264fce392`
	`Version 2`	`5bcdbfd141c8f697717a22f142460b3a7b15bd49`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

[1] https://github.com/flyfoundation/flypad

Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism

* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **seven** potential security issues. Besides, we have **three** recommendations and **one** note.

- High Risk: 3
- Medium Risk: 1
- Low Risk: 3
- Recommendation: 3
- Note: 1

ID	Severity	Description	Category	Status
1	High	Potential drain of funds due to lack of `lastBalanceUpdateRate` reset on full withdrawal	Security Issue	Fixed
2	High	Lack of yield settlement for users during deposit and withdrawal	Security Issue	Fixed
3	High	Potential fund loss for users with `EIP-7702` enabled	Security Issue	Fixed
4	Medium	Lack of yield accrual on token minted to `platformDeployer`	Security Issue	Fixed
5	Low	Potential DoS due to front-run pool creation	Security Issue	Confirmed
6	Low	Lack of slippage protection in the function `buybackAndBurn()`	Security Issue	Fixed
7	Low	Incorrect token recipient in function `deployCoin()`	Security Issue	Fixed
8	-	Add a zero check in `liquidityConfigs`	Recommendation	Fixed
9	-	Remove unused code	Recommendation	Fixed
10	-	Add contract address validation in function `collectFromContract()`	Recommendation	Fixed
11	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Potential drain of funds due to lack of `lastBalanceUpdateRate` reset on full withdrawal

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `FlyBNB` contract, `lastBalanceUpdateRate` is used as the reference rate for yield calculation and is maintained through the `ERC20 _update()` hook during balance changes. When a user withdraws their entire balance, the contract does not reset `lastBalanceUpdateRate` for the account, leaving a stale rate associated with an address that now holds zero tokens. If

this address later receives `flyBNB` via a transfer from another account, the `_update()` logic fails to reinitialize the rate to the current value, as the recipient's rate update only occurs when the stored rate is zero. Because the stale value was never cleared after the full withdrawal, the account's `lastBalanceUpdateRate` remains anchored to an outdated, lower rate despite holding a new balance.

This behavior allows a malicious actor to deliberately preserve an artificially low rate by withdrawing to zero, wait for the global reward rate to increase, and then receive a large balance via transfers (including flash-loaned funds) without triggering proper rate synchronization. As a result, subsequent yield calculations are performed using a large balance combined with a stale rate, enabling the attacker to extract disproportionate yield rewards.

```
85   function withdraw(uint256 amount) external nonReentrant {
86       require(balanceOf(msg.sender) >= amount, "Insufficient balance");
87
88       uint256 currentRate = getAsBnbRate();
89
90       _isConverting = true;
91       _burn(msg.sender, amount);
92
93       uint256 bnbBalanceBefore = address(this).balance;
94       _convertFromAsBNB(amount, currentRate);
95       uint256 bnbToReturn = address(this).balance - bnbBalanceBefore;
96       _isConverting = false;
97
98       if (balanceOf(msg.sender) > 0) {
99           lastBalanceUpdateRate[msg.sender] = currentRate;
100      }
101
102      (bool success, ) = msg.sender.call{value: bnbToReturn}("");
103      require(success, "BNB transfer failed");
104
105      emit Withdraw(msg.sender, amount, bnbToReturn);
106  }
```

Listing 2.1: contracts/contracts/FlyBNB.sol

Impact This vulnerability could lead to the draining of the `asBNB` reserves within the `FlyBNB` contract, causing a loss of funds for all users.

Suggestion When a user's balance becomes zero after a withdrawal, their `lastBalanceUpdateRate` should be reset to zero.

2.1.2 Lack of yield settlement for users during deposit and withdrawal

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `FlyBNB` contract, both functions `deposit()` and `withdraw()` temporarily set the `_isConverting` flag to `true` while modifying user balances. When this flag is active, the

ERC20 `_update()` hook intentionally skips yield settlement logic, including calls to the internal function `_accumulateYield()`, which is responsible for settling a user's accrued rewards up to the current rate.

As a result, if a user with an existing balance and unclaimed yield invokes function `deposit()` or `withdraw()`, their pending yield is not settled before their balance is increased or decreased. Instead, the balance change occurs while yield accumulation is effectively paused, and the user's `lastBalanceUpdateRate` is overwritten afterward. This breaks the expected accounting invariant that accrued yield must always be settled before any balance mutation.

Consequently, previously earned rewards are silently discarded, leading to a loss of yield for the user.

```
62  function deposit() public payable nonReentrant {
63      require(msg.value > 0, "Zero deposit");
64
65      uint256 bnbAmount = msg.value;
66      uint256 currentRate = getAsBnbRate();
67
68      _isConverting = true;
69      uint256 asBnbBefore = IERC20(ASBNB).balanceOf(address(this));
70      IAsBnbMinter(ASBNB_MINTER).mintAsBnb{value: bnbAmount}();
71      uint256 asBnbReceived = IERC20(ASBNB).balanceOf(address(this)) - asBnbBefore;
72
73      // Verify we received enough asBNB to back the flyBNB at current rate
74      // asBNB received should be worth at least the BNB deposited (0.1% tolerance for rounding)
75      uint256 asBnbValue = (asBnbReceived * currentRate) / 1 ether;
76      require(asBnbValue >= (bnbAmount * 999) / 1000, "Insufficient asBNB received");
77
78      _mint(msg.sender, bnbAmount);
79      _isConverting = false;
80
81      lastBalanceUpdateRate[msg.sender] = currentRate;
82      emit Deposit(msg.sender, bnbAmount, bnbAmount);
83  }
```

Listing 2.2: contracts/contracts/FlyBNB.sol

```
85  function withdraw(uint256 amount) external nonReentrant {
86      require(balanceOf(msg.sender) >= amount, "Insufficient balance");
87
88      uint256 currentRate = getAsBnbRate();
89
90      _isConverting = true;
91      _burn(msg.sender, amount);
92
93      uint256 bnbBalanceBefore = address(this).balance;
94      _convertFromAsBNB(amount, currentRate);
95      uint256 bnbToReturn = address(this).balance - bnbBalanceBefore;
96      _isConverting = false;
97
98      if (balanceOf(msg.sender) > 0) {
99          lastBalanceUpdateRate[msg.sender] = currentRate;
100     }
```

```
101
102         (bool success, ) = msg.sender.call{value: bnbToReturn}("");
103         require(success, "BNB transfer failed");
104
105         emit Withdraw(msg.sender, amount, bnbToReturn);
106     }
```

Listing 2.3: contracts/contracts/FlyBNB.sol

Impact Users may permanently lose legitimately accrued yield when interacting with function `deposit()` or `withdraw()`, resulting in incorrect reward accounting and financial loss.

Suggestion Ensure that a user's accrued yield is fully settled before any balance changes during functions `deposit()` and `withdraw()`.

2.1.3 Potential fund loss for users with `EIP-7702` enabled

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `FlyBNB` contract, the `claimFor()` function is intended to settle yield for contract accounts and relies on an `account.code.length > 0` check to differentiate smart contracts from `EOA`s. This assumption becomes invalid under `EIP-7702`, where an `EOA` may have executable code associated with its address while still being controlled by a user.

As a result, an `EIP-7702` enabled user address will satisfy the `_isContract(account)` condition and be classified as a contract account. If the function `claimFor()` is invoked for such an address, the contract-side yield settlement logic is applied instead of the user yield path, causing the user's accrued yield to be redirected to the `platformDeployer`.

```
297     /// @notice Settle yield for a contract that hasn't transferred - yield goes to platform (+
            controller for pools)
298     function claimFor(address account) external nonReentrant {
299         require(_isContract(account), "Use claim() for users");
300         uint256 currentRate = getAsBnbRate();
301         _settleContractYield(account, currentRate);
302         lastBalanceUpdateRate[account] = currentRate;
303     }
```

Listing 2.4: contracts/contracts/FlyBNB.sol

Impact Users who have enabled `EIP-7702` are at risk of losing all their accrued yield.

Suggestion Revise the logic accordingly.

2.1.4 Lack of yield accrual on token minted to `platformDeployer`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `FlyBNB`, the function `_settleContractYield()` is responsible for settling the accrued yield. When the account corresponds to a pool created by the contract `Factory`, half of the settled yield is minted to `platformDeployer`; otherwise, the entire yield is minted to `platformDeployer`.

However, when yield is minted directly to `platformDeployer`, the function `_settleContractYield()` does not first settle the yield accrued on `platformDeployer`'s existing `flyBNB` balance. As a result, the newly minted `flyBNB` inherits the previously recorded rate, causing excess yield to be accounted for by `platformDeployer`.

```
240   function _settleContractYield(address account, uint256 currentRate) internal {
241       uint256 currentBalance = balanceOf(account);
242       if (currentBalance == 0) return;
243
244       uint256 lastRate = lastBalanceUpdateRate[account];
245       if (lastRate == 0) return;
246
247       // Only settle if rate strictly increased (prevents 0 or negative yields / overflow)
248       if (currentRate <= lastRate) return;
249
250       uint256 yieldAmount = (currentBalance * currentRate) / lastRate - currentBalance;
251       if (yieldAmount == 0) return;
252
253       _isRebasing = true;
254
255       address creator = _getPoolCreator(account);
256       if (creator != address(0) && yieldController != address(0)) {
257           // Pool: 50% to platform, 50% to yieldController for buyback
258           uint256 platformShare = yieldAmount / 2;
259           uint256 controllerShare = yieldAmount - platformShare;
260
261           _mint(platformDeployer, platformShare);
262           _mint(address(this), controllerShare);
263
264           address token = _getPoolToken(account);
265           IYieldController(yieldController).depositYield(controllerShare, token);
266
267           emit PoolYieldDistributed(account, creator, yieldAmount);
268       } else {
269           // Non-pool contract: 100% to platform
270           _mint(platformDeployer, yieldAmount);
271           emit ContractRebaseRedirected(account, platformDeployer, yieldAmount);
272       }
273
274       _isRebasing = false;
275   }
```

Listing 2.5: contracts/contracts/FlyBNB.sol

Impact Direct minting of `flyBNB` to `platformDeployer` without settling existing yield allows it to receive excess yield.

Suggestion Revise the corresponding logic.

2.1.5 Potential DoS due to front-run pool creation

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `Factory`, the function `deployCoin()` allows users to deploy a new token and mint its entire supply as liquidity into a `Pancake V3 pool` within a predefined liquidity range, with the resulting liquidity position NFT being locked.

However, an attacker can front-run the transaction in the mempool by preemptively creating the corresponding `Pancake V3 pool` and initializing it with an arbitrary price. If the current pool price falls within the intended liquidity range, the subsequent attempt to mint single-sided liquidity in the function `deployCoin()` will fail, causing the transaction to revert and resulting in a DoS to the token deployment process.

```
85   function deployCoin(string memory _name, string memory _symbol, string memory _metadata,
         bytes32 salt, address receiver, uint256 split, uint256 configId) public payable returns (
         uint256 tokensReceived) {
86       require(deployCoinEnabled, "Token deployment is currently disabled");
87       require(configId < liquidityConfigCount, "Invalid liquidity config ID");
88
89       bytes32 validSalt = findValidSalt(_name, _symbol, msg.sender, salt);
90
91       Token t = new Token{salt: validSalt}(
92           _name,
93           _symbol,
94           msg.sender,
95           address(this),
96           flyBNB
97       );
98       emit ERC20TokenCreated(address(t));
99
100      address coin_address = address(t);
101      require(coin_address < flyBNB, "Token address must be < flyBNB");
102
103      provideLiquidity(coin_address, flyBNB, configId, msg.sender);
104
105      tokensReceived = 0;
106
107      if (msg.value > 0) {
108          LiquidityConfig memory config = liquidityConfigs[configId];
109
110          uint256 flyBNBBefore = IERC20(flyBNB).balanceOf(address(this));
111          IFlyBNB(flyBNB).deposit{value: msg.value}();
112          uint256 flyBNBReceived = IERC20(flyBNB).balanceOf(address(this)) - flyBNBBefore;
113
114          uint256 expectedTokens = _calculateExpectedTokens(flyBNBReceived, config.sqrtPriceX96);
115          uint256 minTokens = (expectedTokens * (10000 - slippageBps)) / 10000;
116
117          IERC20(flyBNB).approve(SWAP_ROUTER, flyBNBReceived);
118
```

```
119             ISwapRouter02(SWAP_ROUTER).exactInputSingle(
120                 ISwapRouter02.ExactInputSingleParams({
121                     tokenIn: flyBNB,
122                     tokenOut: coin_address,
123                     fee: FEE_TIER,
124                     recipient: address(this),
125                     amountIn: flyBNBReceived,
126                     amountOutMinimum: minTokens,
127                     sqrtPriceLimitX96: 0
128                 })
129             );
130 
131             IERC20 token = IERC20(coin_address);
132             tokensReceived = token.balanceOf(address(this));
133             token.transfer(msg.sender, tokensReceived);
134         }
135 
136         TokenInfo memory newTokenInfo = TokenInfo({
137             tokenAddress: coin_address,
138             name: _name,
139             symbol: _symbol,
140             deployer: msg.sender,
141             time: block.timestamp,
142             metadata: _metadata,
143             marketCapInETH: 0,
144             receiver: receiver,
145             split: split,
146             nftId: tokenToNFTId[coin_address],
147             tokenBalance: 0,
148             totalFeesGenerated: 0
149         });
150 
151         deployedTokens[tokenCount] = newTokenInfo;
152         tokenInfoByAddress[coin_address] = newTokenInfo;
153 
154         receiverTokens[receiver].push(coin_address);
155         creatorTokens[msg.sender].push(coin_address);
156 
157         tokenCount++;
158 
159         return tokensReceived;
160     }
```

Listing 2.6: contracts/contracts/Factory.sol

Impact Front-run pool creation can lead to a DoS, preventing successful token deployment via the function `deployCoin()`.

Suggestion Revise the corresponding logic.

Feedback from the project Even if a DoS attack happens, the deployer can redeploy without much impact.

2.1.6 Lack of slippage protection in the function `buybackAndBurn()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `BuybackAccumulator`, the function `buybackAndBurn()` uses the accumulated `flyBNB` yield from a specific token pool to buy back and burn the corresponding token. However, the swap logic lacks slippage protection, and anyone can call this function, making it vulnerable to sandwich attacks, including via flash loans.

```
66  function buybackAndBurn(address token) external nonReentrant returns (uint256 tokensBurned) {
67      uint256 amountToSpend = accumulated[token];
68      require(amountToSpend > 0, "No accumulated yield");
69
70      accumulated[token] = 0;
71
72      uint256 tokenBalanceBefore = IERC20(token).balanceOf(address(this));
73
74      // Swap flyBNB for token
75      ISwapRouter02(swapRouter).exactInputSingle(
76          ISwapRouter02.ExactInputSingleParams({
77              tokenIn: flyBNB,
78              tokenOut: token,
79              fee: FEE_TIER,
80              recipient: address(this),
81              amountIn: amountToSpend,
82              amountOutMinimum: 0,
83              sqrtPriceLimitX96: 0
84          })
85      );
86
87      tokensBurned = IERC20(token).balanceOf(address(this)) - tokenBalanceBefore;
88
89      // Send to dead address
90      if (tokensBurned > 0) {
91          IERC20(token).transfer(DEAD_ADDRESS, tokensBurned);
92          totalBurned[token] += tokensBurned;
93          totalBuybackVolume[token] += amountToSpend;
94      }
95
96      emit BuybackExecuted(token, msg.sender, amountToSpend, tokensBurned);
97
98      return tokensBurned;
99  }
```

Listing 2.7: contracts/contracts/BuybackAccumulator.sol

Impact The lack of slippage protection allows attackers to perform sandwich attacks, causing financial loss.

Suggestion Add slippage protections.

2.1.7 Incorrect token recipient in function `deployCoin()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `Factory` contract, the `deployCoin()` function accepts a `receiver` parameter that is stored as part of the deployed token's metadata and is intended to represent the designated recipient of the initially purchased tokens. However, when native tokens are provided and a swap is executed, the purchased tokens are transferred to `msg.sender` instead of the specified `receiver`.

This creates an inconsistency between the function interface, the recorded token metadata, and the actual token distribution behavior, which may not align with the caller's expectations.

```
85   function deployCoin(string memory _name, string memory _symbol, string memory _metadata,
         bytes32 salt, address receiver, uint256 split, uint256 configId) public payable returns (
         uint256 tokensReceived) {
86       require(deployCoinEnabled, "Token deployment is currently disabled");
87       require(configId < liquidityConfigCount, "Invalid liquidity config ID");
88
89       bytes32 validSalt = findValidSalt(_name, _symbol, msg.sender, salt);
90
91       Token t = new Token{salt: validSalt}(
92           _name,
93           _symbol,
94           msg.sender,
95           address(this),
96           flyBNB
97       );
98       emit ERC20TokenCreated(address(t));
99
100      address coin_address = address(t);
101      require(coin_address < flyBNB, "Token address must be < flyBNB");
102
103      provideLiquidity(coin_address, flyBNB, configId, msg.sender);
104
105      tokensReceived = 0;
106
107      if (msg.value > 0) {
108          LiquidityConfig memory config = liquidityConfigs[configId];
109
110          uint256 flyBNBBefore = IERC20(flyBNB).balanceOf(address(this));
111          IFlyBNB(flyBNB).deposit{value: msg.value}();
112          uint256 flyBNBReceived = IERC20(flyBNB).balanceOf(address(this)) - flyBNBBefore;
113
114          uint256 expectedTokens = _calculateExpectedTokens(flyBNBReceived, config.sqrtPriceX96);
115          uint256 minTokens = (expectedTokens * (10000 - slippageBps)) / 10000;
116
117          IERC20(flyBNB).approve(SWAP_ROUTER, flyBNBReceived);
118
```

```
119             ISwapRouter02(SWAP_ROUTER).exactInputSingle(
120                 ISwapRouter02.ExactInputSingleParams({
121                     tokenIn: flyBNB,
122                     tokenOut: coin_address,
123                     fee: FEE_TIER,
124                     recipient: address(this),
125                     amountIn: flyBNBReceived,
126                     amountOutMinimum: minTokens,
127                     sqrtPriceLimitX96: 0
128                 })
129             );
130
131             IERC20 token = IERC20(coin_address);
132             tokensReceived = token.balanceOf(address(this));
133             token.transfer(msg.sender, tokensReceived);
134         }
135
136         TokenInfo memory newTokenInfo = TokenInfo({
137             tokenAddress: coin_address,
138             name: _name,
139             symbol: _symbol,
140             deployer: msg.sender,
141             time: block.timestamp,
142             metadata: _metadata,
143             marketCapInETH: 0,
144             receiver: receiver,
145             split: split,
146             nftId: tokenToNFTId[coin_address],
147             tokenBalance: 0,
148             totalFeesGenerated: 0
149         });
150
151         deployedTokens[tokenCount] = newTokenInfo;
152         tokenInfoByAddress[coin_address] = newTokenInfo;
153
154         receiverTokens[receiver].push(coin_address);
155         creatorTokens[msg.sender].push(coin_address);
156
157         tokenCount++;
158
159         return tokensReceived;
160     }
```

Listing 2.8: contracts/contracts/Factory.sol

Impact Tokens acquired during deployment may be delivered to an unintended address, potentially breaking downstream assumptions, integrations, or business logic that rely on the `receiver` parameter.

Suggestion Update the token transfer logic to send the purchased tokens to `receiver` instead of `msg.sender`, ensuring consistency with the function parameters and recorded token information.

2.2 Recommendation

2.2.1 Add a zero check in `liquidityConfigs`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `provideLiquidity()` function, a liquidity configuration is retrieved using a `configId` that is only validated against `liquidityConfigCount`. However, when a liquidity configuration is deleted, the corresponding entry is not removed or compacted, and `liquidityConfigCount` is not decremented. As a result, a `configId` referring to a deleted configuration can still pass the bounds check.

In such cases, the function proceeds to read a zero-initialized configuration struct. Using these empty values in subsequent logic can lead to unexpected reverts during execution.

```
87        require(configId < liquidityConfigCount, "Invalid liquidity config ID");
```

Listing 2.9: contracts/contracts/Factory.sol

```
372   function provideLiquidity(address tokenA, address tokenB, uint256 configId, address creator)
          internal {
373       LiquidityConfig memory config = liquidityConfigs[configId];
374
375       require(tokenA < tokenB, "Token must be < WBNB");
376
377       address token0 = tokenA;
378       address token1 = tokenB;//flyBNB
379
380       IERC20(token0).approve(POSITION_MANAGER, type(uint256).max);
381       IERC20(token1).approve(POSITION_MANAGER, type(uint256).max);
382
383       INonfungiblePositionManager manager = INonfungiblePositionManager(POSITION_MANAGER);
384
385       uint160 sqrtPriceX96 = config.sqrtPriceX96;
386
387       int24 tickLower = -config.tickLower;
388       int24 tickUpper = config.tickUpper;
389
390       uint256 amount0Desired = config.amount0Desired;
391       uint256 amount1Desired = config.amount1Desired;
392
393       address pool = manager.createAndInitializePoolIfNecessary(token0, token1, FEE_TIER,
              sqrtPriceX96);
394
395       poolToCreator[pool] = creator;
396
397       (uint256 tokenId, , , ) = manager.mint(
398           INonfungiblePositionManager.MintParams({
399               token0: token0,
400               token1: token1,
401               fee: FEE_TIER,
402               tickLower: tickLower,
```

```
403             tickUpper: tickUpper,
404             amount0Desired: amount0Desired,
405             amount1Desired: amount1Desired,
406             amount0Min: 0,
407             amount1Min: 0,
408             recipient: address(this),
409             deadline: block.timestamp
410         })
411     );
412     _storeNFTId(tokenA, tokenId);
413
414     // Lock LP NFT in GoPlus locker (permanently locked, Factory as collector)
415     manager.approve(LP_LOCKER, tokenId);
416     uint256 lockId = IUniV3LPLocker(LP_LOCKER).lock(
417         manager,
418         tokenId,
419         address(this),        // owner (can unlock after endTime - but we set 100 years)
420         address(this),        // collector (can collect fees)
421         block.timestamp + LOCK_DURATION,
422         "DEFAULT"             // Fee structure: 0.4% LP fee, 1.6% collect fee, 0 BNB lock fee
423     );
424     nftIdToLockId[tokenId] = lockId;
425 }
```

Listing 2.10: contracts/contracts/Factory.sol

Suggestion Add a check to ensure the config is not empty before using it.

2.2.2 Remove unused code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `FlyBNB` contract, the constant `POOL_FEE` is defined but never referenced in the codebase. Additionally, the `virtualAmount` field in the `LiquidityConfig` struct is declared but not used in any logic.

These unused definitions increase code complexity and may cause confusion about intended behavior or incomplete features.

```
28  uint24 public constant POOL_FEE = 10000;
```

Listing 2.11: contracts/contracts/FlyBNB.sol

```
65      liquidityConfigs[0] = LiquidityConfig({
66          sqrtPriceX96: 5587685005711249773720921, // At tick -191200
67          tickLower: 191150,
68          tickUpper: 887200,
69          amount0Desired: 100000000000000000000000000,
70          amount1Desired: 0,
71          virtualAmount: 1.5 ether
72      });
```

Listing 2.12: contracts/contracts/Factory.sol

Suggestion Remove unused code.

2.2.3 Add contract address validation in function `collectFromContract()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `collectFromContract()` function is intended to collect `flyBNB` tokens from contract accounts. However, the function currently does not verify whether the provided `contractAddress` is actually a contract. As a result, an externally owned account (`EOA`) address can be passed in, as long as it has granted allowance to this contract.

```
109   function collectFromContract(address contractAddress) external onlyPlatformDeployer
          nonReentrant {
110       require(contractAddress != address(0), "Invalid address");
111       require(contractAddress != address(this), "Cannot collect from self");
112
113       uint256 amount = IERC20(flyBNB).balanceOf(contractAddress);
114       require(amount > 0, "No flyBNB to collect");
115
116       uint256 balanceBefore = IERC20(flyBNB).balanceOf(address(this));
117       IERC20(flyBNB).transferFrom(contractAddress, address(this), amount);
118       uint256 received = IERC20(flyBNB).balanceOf(address(this)) - balanceBefore;
119
120       if (received > 0) {
121           IERC20(flyBNB).transfer(platformDeployer, received);
122           totalPlatformYield += received;
123           totalYieldDistributed += received;
124       }
125
126       emit FlyBNBCollectedFromContract(contractAddress, received);
127   }
```

Listing 2.13: contracts/contracts/YieldController.sol

Suggestion Add the check if the address is a contract.

2.3 Note

2.3.1 Potential centralization risks

Introduced by `Version 1`

Description In this protocol, privileged roles such as the `platformController` can conduct sensitive operations (e.g., using function `emergencyWithdraw()` to extract assets reserved for meme token buybacks). If the private key of this account is compromised or maliciously exploited, it could pose a significant risk to the protocol and potentially lead to user asset loss.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS