BLOCKSEC

# Security Audit
# Report for Astherus Contracts

**Date:** February 10, 2025 **Version:** 1.0

**Contact:** contact@blocksec.com

# Contents

## Report Manifest

| Item | Description |
| --- | --- |
| Client | Astherus |
| Target | Astherus Contracts |

## Version History

| Version | Date | Description |
| --- | --- | --- |
| 1.0 | February 10, 2025 | First release |

## Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-12-04 06:31:21+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 of ass-bnb-earn-contract[1], asBTC[2], ass-cake-earn-contract[3], asUSDFEarn[4], asUSDF[5] of Astherus. Note that, we did **NOT** audit all the modules in the repository. Specifically, the files covered in this audit include:

- ass-bnb-earn-contract/src/*
- ass-cake-earn-contract/src/*
- astherus-earn-contract/contracts/oft/asBTC.sol
- asUSDF.sol
- asUSDFEarn.sol

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.

| Project | Version | Commit Hash |
|---|---|---|
| ass-bnb-earn-contract | Version 1 | 61fc9f5610914cf94dddc3ab83e21ca22d388e92 |
| | Version 2 | 9bddf07c4a2792e8567b2863c48a62e079d07273 |
| astherus-earn-contract | Version 1 | b3bd39d38dac0e1671514c8e600dfee7d0fdf148 |
| ass-cake-earn-contract | Version 1 | 5eb634c4ca7e1597e943c8c46e91683d76ac3ba9 |
| | Version 2 | ebb5af69a25cb1d5363731edf6ad662f021dec85 |

## 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. 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.

[1] https://github.com/astherus-contract/ass-bnb-earn-contract

[2] https://github.com/astherus-contract/astherus-earn-contract/blob/audit/contracts/oft/asBTC.sol

[3] https://github.com/astherus-contract/ass-cake-earn-contract

[4] https://bscscan.com/address/0xdB57a53C428a9faFcbFefFB6dd80d0f427543695

[5] https://bscscan.com/address/0x917AF46B3C3c6e1Bb7286B9F59637Fb7C65851Fb

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 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

### 1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism

- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

### 1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

### 1.3.4 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 [6] and Common Weakness Enumeration [7]. 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**,

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

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

**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 four 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.
- **Fixed** The item has been confirmed and fixed by the client.

# Chapter 2 Findings

In total, we found **four** potential security issues. Besides, we have **seven** recommendations and **four** notes.

- Low Risk: 4
- Recommendation: 7
- Note: 4

| ID | Severity | Description | Category | Status |
|---|---|---|---|---|
| 1 | Low | Lack of invoking function `reDelegateTokens()` after mpcWallet update | DeFi Security | Fixed |
| 2 | Low | Lack of refunding mechanism for excessive cross-chain fees | DeFi Security | Fixed |
| 3 | Low | Potential reward loss due to COMPOUNDER role change | DeFi Security | Confirmed |
| 4 | Low | Precision loss in reward distribution results in permanently locked dust tokens | DeFi Security | Fixed |
| 5 | - | Lack of non-zero check for key parameters | Recommendation | Fixed |
| 6 | - | Lack of range check in function `updateMaxRewardPercent()` | Recommendation | Fixed |
| 7 | - | Inconsistent validation of revenueSharingPools variable | Recommendation | Fixed |
| 8 | - | Add non-zero checks in `add1InchRouterWhitelist()` and `addSwapSrcTokenWhitelist()` functions | Recommendation | Fixed |
| 9 | - | Redundant event definitions in IWithdrawVault contract | Recommendation | Confirmed |
| 10 | - | Grant the BOT role in function `initialize()` of Buyback contract | Recommendation | Fixed |
| 11 | - | Incorrect comment in function `_smartMint()` | Recommendation | Fixed |
| 12 | - | Potential centralization risk | Note | - |
| 13 | - | Timely distributing rewards before updating rewardsSender in function `setRewardsSender()` | Note | - |
| 14 | - | Timely invocation of the function `settleActivity()` | Note | - |
| 15 | - | Potential DoS due to expired lock in VeCake contract | Note | - |

The details are provided in the following sections.

## 2.1 DeFi Security

### 2.1.1 Lack of invoking function `reDelegateTokens()` after mpcWallet update

**Severity** Low

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** The `YieldProxy` contract is designed to deposit received `slisBNB` into the `SlisBNBProvider` and delegates the minted `clisBNB` to the `mpcWallet` to earn rewards. Note that the `SlisBNBProvider` supports delegation to only one target at a time. Therefore, whenever the `MANAGER` role updates the `mpcWallet` using the `setMPCWallet()` function, it is crucial to immediately invoke the `reDelegateTokens()` function. Failure to do so will cause the delegation to persist with the previous `mpcWallet`, leading to a denial of service (`DoS`) and preventing users from minting `asBNB`.

```
338   function setMPCWallet(address _mpcWallet) external onlyRole(MANAGER) {
339       require(_mpcWallet != address(0) && _mpcWallet != mpcWallet, "Invalid MPC wallet address");
340       address oldMpcWallet = mpcWallet;
341       mpcWallet = _mpcWallet;
342       emit MPCWalletSet(oldMpcWallet, _mpcWallet);
343   }
```

**Listing 2.1:** YieldProxy.sol

```
304   function reDelegateTokens() external onlyRole(MANAGER) whenNotPaused {
305       require(slisBNBProvider != address(0), "slisBNBProvider not set");
306       require(mpcWallet != address(0), "mpcWallet not set");
307       ISlisBNBProvider(slisBNBProvider).delegateAllTo(mpcWallet);
308   }
```

**Listing 2.2:** YieldProxy.sol

```
 80   function provide(uint256 _amount, address _delegateTo)
 81       external
 82       virtual
 83       whenNotPaused
 84       nonReentrant
 85   returns (uint256)
 86   {
 87       require(_amount > 0, "zero deposit amount");
 88       require(_delegateTo != address(0), "delegateTo cannot be zero address");
 89       require(_delegateTo != msg.sender, "delegateTo cannot be self");
 90       require(
 91           delegation[msg.sender].delegateTo == _delegateTo ||
 92           delegation[msg.sender].amount == 0, // first time, clear old delegatee
 93           "delegateTo is differ from the current one"
 94       );
 95
 96
 97       IERC20(token).safeTransferFrom(msg.sender, address(this), _amount);
```

```
98        // do sync before balance modified
99        _syncLp(msg.sender);
100       uint256 userPartLp = _provideCollateral(msg.sender, _delegateTo, _amount);
101
102
103       Delegation storage userDelegation = delegation[msg.sender];
104       userDelegation.delegateTo = _delegateTo;
105       userDelegation.amount += userPartLp;
106
107
108       emit Deposit(msg.sender, _amount, userPartLp);
109       return userPartLp;
110   }
```

**Listing 2.3:** SlisBNBProvider.sol

**Impact** Users are unable to deposit assets into the protocol.

**Suggestion** Merge `setMPCWallet()` and `reDelegateTokens()` into a single function to ensure timely redelegation to the new address after update.

### 2.1.2 Lack of refunding mechanism for excessive cross-chain fees

**Severity** Low

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** The `mintAsBnbToChain()` function allows users to pay a certain amount of native tokens as cross-chain fees to mint a corresponding amount of `asBNB` by staking `slisBNB`. The minted `asBNB` will then be transferred to the target chain for users. However, if the attached cross-chain fees exceed the required amount, the excess fee will not be refunded to the user, which is incorrect.

```
243   function mintAsBnbToChain(
244       uint256 amountIn,
245       SendParam memory sendParam
246   ) external payable override whenNotPaused depositEnabled nonReentrant returns (uint256) {
247       // get cross chain fee and validate the existence of OFTAdapter
248       MessagingFee memory fee = getCrossChainFee(sendParam);
249       require(msg.value >= fee.nativeFee, "Invalid fee");
250       // transfer token from user to this contract
251       token.safeTransferFrom(msg.sender, address(this), amountIn);
252
253
254       return _mintToChain(amountIn, sendParam, fee);
255   }
```

**Listing 2.4:** AsBnbMinter.sol

**Impact** Excess fees paid by the user will not be refunded.

**Suggestion** Add a refund mechanism for excess cross-chain fees or enforce that the attached `msg.value` exactly matches the calculated fee.

### 2.1.3 Potential reward loss due to COMPOUNDER role change

**Severity** Low

**Status** Confirmed

**Introduced by** `Version 1`

**Description** In the `ass-cake-earn` protocol, the `RewardDistributionScheduler` contract, acting as the `COMPOUNDER` role for the `Minter` contract, is responsible for redistributing the received `CAKE` tokens according to the pre-set schedules back to the `Minter` contract for further use. However, only contracts with the `COMPOUNDER` role can invoke the function `compoundRewards()`. Therefore, before changing the `COMPOUNDER` role in the `Minter` contract, it is crucial to ensure that all pending reward distributions are executed, as `CAKE` tokens that haven't been executed cannot be compounded after the role change.

```
136   function executeRewardSchedules() external override onlyRole(BOT) nonReentrant whenNotPaused {
137       // flooring the current timestamp to 1 day
138       uint256 currentTimestamp = (block.timestamp / 1 days) * 1 days;
139       // get num. of reward types
140       uint max = (uint)(type(IMinter.RewardsType).max);
141       // rewards type array
142       IMinter.RewardsType[] memory rewardsTypes = new IMinter.RewardsType[](max);
143       // total rewards per type array
144       uint256[] memory totalRewards = new uint256[](max);
145
146
147       // from the day of last distribution
148       // process the rewards day by day
149       while (lastDistributeRewardsTimestamp <= currentTimestamp) {
150         // sum up rewards for each type
151         for (uint i; i < max; ++i) {
152           rewardsTypes[i] = IMinter.RewardsType(i);
153           // if there are rewards to distribute for that type on that day
154           if (epochs[lastDistributeRewardsTimestamp][rewardsTypes[i]] != 0) {
155             // add up rewards for that type and increase allowance
156             totalRewards[i] += epochs[lastDistributeRewardsTimestamp][rewardsTypes[i]];
157             IERC20(token).safeIncreaseAllowance(minter, epochs[lastDistributeRewardsTimestamp][
                      rewardsTypes[i]]);
158             // remove it from the epoch
159             delete epochs[lastDistributeRewardsTimestamp][IMinter.RewardsType(i)];
160           }
161         }
162         // process the next day
163         lastDistributeRewardsTimestamp += 1 days;
164       }
165       // compound all types of rewards at once
166       IMinter(minter).compoundRewards(rewardsTypes, totalRewards);
167   }
```

**Listing 2.5:** RewardDistributionScheduler.sol

```
237   function compoundRewards(
238       IMinter.RewardsType[] memory _rewardsTypes,
```

```
239         uint256[] memory _rewards
240     ) external override onlyRole(COMPOUNDER) whenNotPaused nonReentrant {
241         require(_rewardsTypes.length > 0 && _rewardsTypes.length == _rewards.length, "Invalid
                rewards length");
242
243
244         // separate compoundAmount and fee
245         uint256 compoundAmount = 0;
246         uint256 fee = 0;
247         // compound the rewards
248         for (uint i; i < _rewardsTypes.length; ++i) {
249             IMinter.RewardsType rewardsType = _rewardsTypes[i];
250             uint256 amountInPerType = _rewards[i];
251             // process only if amount is not 0
252             if (amountInPerType != 0) {
253                 uint256 feePerType = 0;
254                 // collect fee per type
255                 if (rewardsType == RewardsType.VeTokenRewards) {
256                   feePerType = (amountInPerType * veTokenRewardsFeeRate) / DENOMINATOR;
257                 } else if (rewardsType == RewardsType.VoteRewards) {
258                   feePerType = (amountInPerType * voteRewardsFeeRate) / DENOMINATOR;
259                 } else if (rewardsType == RewardsType.Donate) {
260                   feePerType = (amountInPerType * donateRewardsFeeRate) / DENOMINATOR;
261                 } else {
262                   revert("Invalid rewardsType");
263                 }
264                 // add up fee
265                 fee += feePerType;
266                 // add up amt. of compound after fee is deducted
267                 compoundAmount += amountInPerType - feePerType;
268                 // accumulate rewards per type
269                 totalRewards[rewardsType] += amountInPerType - feePerType;
270             }
271         }       // transfer token from compounder
272         require(compoundAmount + fee > 0, "Invalid compound amount");
273         IERC20(token).safeTransferFrom(msg.sender, address(this), compoundAmount + fee);
274
275
276         // reset vesting amount
277         _updateVestingAmount(compoundAmount);
278
279
280         // update total fee and actual total tokens
281         totalFee += fee;
282         _totalTokens += compoundAmount;
283
284
285         // compound rewards
286         IERC20(token).safeIncreaseAllowance(universalProxy, compoundAmount);
287         IUniversalProxy(universalProxy).lock(compoundAmount);
288
289
290         emit RewardsCompounded(msg.sender, compoundAmount, fee);
```

```
291   }
```

**Listing 2.6:** Minter.sol

**Impact** The received `CAKE` will remain locked in the `RewardDistributionScheduler` contract.

**Suggestion** Ensure all pending reward distributions are executed before changing the `COMPOUNDER` role to prevent locking `CAKE` tokens.

**Feedback from the project** The team will ensure all rewards will be fully distributed if there is any upcoming change of the `COMPOUNDER` role.

### 2.1.4 Precision loss in reward distribution results in permanently locked dust tokens

**Severity** Low

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the `RewardDistributionScheduler` contract of the `ass-cake-earn` protocol, the privileged role `MANAGER` is authorized to deposit rewards into the contract via the function `addRewardsSchedule()`, which evenly distributes the rewards across each `epoch`. However, due to integer division, this process results in leftover token dust. The contract lacks a mechanism to handle or distribute this dust, and it also cannot be withdrawn. Consequently, these leftover rewards are permanently locked within the contract.

```
 94   function addRewardsSchedule(
 95       IMinter.RewardsType _rewardsType,
 96       uint256 _amount,
 97       uint256 _epochs,
 98       uint256 _startTime
 99   ) external override onlyRole(MANAGER) nonReentrant whenNotPaused {
100       require(_amount > 0, "Invalid amount");
101       require(_epochs > 0, "Invalid epochs");
102       require(_startTime > 0, "Invalid startTime");
103       //valid rewardsType
104       require(
105         IMinter.RewardsType.VeTokenRewards == _rewardsType ||
106           IMinter.RewardsType.VoteRewards == _rewardsType ||
107           IMinter.RewardsType.Donate == _rewardsType,
108         "Invalid rewardsType"
109       );
110
111
112       uint256 startTime = (_startTime / 1 days) * 1 days;
113
114
115       // The rewards have been distributed,
116       // and additional reward plans added later.
117       // if the start time is less than the emitted time. Need to reset the release time
118       if (startTime < lastDistributeRewardsTimestamp) {
119         lastDistributeRewardsTimestamp = startTime;
```

```
120       }
121
122
123       // transfer funds to this contract
124       token.safeTransferFrom(msg.sender, address(this), _amount);
125       // average daily reward amount
126       uint256 amountPerDay = _amount / _epochs;
127       // spread rewards every day
128       for (uint256 i; i < _epochs; i++) {
129         // accumulation of different reward types
130         epochs[startTime + i * 1 days][_rewardsType] += amountPerDay;
131       }
132       // emit event
133       emit RewardsScheduleAdded(msg.sender, _rewardsType, _amount, _epochs, startTime);
134   }
```

**Listing 2.7:** RewardDistributionScheduler.sol

**Impact** Part of the rewards will be permanently locked in the contract and cannot be withdrawn.

**Suggestion** Revise the calculation logic to ensure that the rewards are fully distributed.

## 2.2 Additional Recommendation

### 2.2.1 Lack of non-zero check for key parameters

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the `AsBnbOFT` contract, the function `initialize()` is used to configure important global variables, including roles and essential settings for the contract. However, it does not check if any of the provided addresses are zero when granting roles, which is inappropriate.

```
35   function initialize(
36       address _admin,
37       address _manager,
38       address _pauser,
39       string memory _name,
40       string memory _symbol,
41       address _delegate
42   ) external initializer {
43       __OFT_init(_name, _symbol, _delegate);
44       __Ownable_init(_delegate);
45       __AccessControl_init();
46       __Pausable_init();
47       __ERC20Permit_init(_name);
48       __UUPSUpgradeable_init();
49
50
51       _grantRole(DEFAULT_ADMIN_ROLE, _admin);
52       _grantRole(MANAGER, _manager);
```

```
53        _grantRole(PAUSER, _pauser);
54    }
```

**Listing 2.8:** AsBnbOFT.sol

**Suggestion** Add a check to ensure that the provided addresses are not zero addresses.

### 2.2.2 Lack of range check in function `updateMaxRewardPercent()`

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the `asUSDFEarn` contract, the `maxRewardPercent` parameter controls the maximum percentage of rewards distributed, which should be a value between 0 and 1e18. However, the `updateMaxRewardPercent()` function does not perform a range check. If the parameter is set to 0, rewards will always be 0. If set above 1e18, it could cause a single distribution to exceed the total distributable amount, leading to incorrect behavior.

```
106   function updateMaxRewardPercent(uint newValue) external onlyRole(ADMIN_ROLE) {
107       uint oldValue = maxRewardPercent;
108       require(oldValue != newValue, "already set");
109       maxRewardPercent = newValue;
110       emit UpdateMaxRewardPercent(oldValue, newValue);
111   }
```

**Listing 2.9:** asUSDFEarn.sol

```
185   function dispatchReward(uint amount) external nonReentrant onlyRole(REWARD_ROLE) {
186       //
187       if (getUnvestedAmount() > 0) {
188           return;
189       }
190       require(amount <= USDF.totalSupply() * maxRewardPercent / EXCHANGE_PRICE_DECIMALS, "too
              much");
191       lastDispatchTime = block.timestamp;
192       USDF.safeTransferFrom(msg.sender, address(this), amount);
193       lastReward = amount;
194       emit RewardDispatched(amount, block.timestamp, VESTING_PERIOD);
195   }
```

**Listing 2.10:** asUSDFEarn.sol

**Suggestion** Implement a range check to ensure that `maxRewardPercent` is within a valid range.

### 2.2.3 Inconsistent validation of revenueSharingPools variable

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the `UniversalProxy` contract of the `ass-cake-earn` protocol, the `initialize()` function requires the `revenueSharingPools` array to be non-zero. However, in the `setRevenuePoolIds()` function, it allows the `revenueSharingPools` array to be empty, which is inconsistent with the validation logic.

```
 91   function initialize(
 92       address _admin,
 93       address _pauser,
 94       address _minter,
 95       address _bot,
 96       address _manager,
 97       address _token,
 98       address _veToken,
 99       address _gaugeVoting,
100       address _ifo,
101       address _rewardDistributionScheduler,
102       address[] memory _revenueSharingPools,
103       address _revenueSharingPoolGateway,
104       address _cakePlatform
105   ) external initializer {
106       require(_admin != address(0), "Invalid admin address");
107       require(_token != address(0), "Invalid token address");
108       require(_veToken != address(0), "Invalid veToken address");
109       require(_gaugeVoting != address(0), "Invalid gaugeVoting address");
110       require(_ifo != address(0), "Invalid ifo address");
111       require(_revenueSharingPoolGateway != address(0), "Invalid revenueSharingPoolGateway
              address");
112       require(_cakePlatform != address(0), "Invalid cakePlatform address");
113       require(_rewardDistributionScheduler != address(0), "Invalid rewardDistributionScheduler
              address");
114       require(_revenueSharingPools.length > 0, "revenueSharingPools must have at least one
              address");
115
116
117       __AccessControl_init();
118       __Pausable_init();
119       __ReentrancyGuard_init();
120       __UUPSUpgradeable_init();
121       _grantRole(DEFAULT_ADMIN_ROLE, _admin);
122       _grantRole(PAUSER, _pauser);
123       _grantRole(MINTER, _minter);
124       _grantRole(BOT, _bot);
125       _grantRole(MANAGER, _manager);
126
127
128       lockCreated = false;
129       token = IERC20(_token);
130       veToken = IVeCake(_veToken);
131       gaugeVoting = IGaugeVoting(_gaugeVoting);
132       ifo = IIFOV8(_ifo);
133       revenueSharingPools = _revenueSharingPools;
134       revenueSharingPoolGateway = IRevenueSharingPoolGateway(_revenueSharingPoolGateway);
135       rewardsDistributionScheduler = IRewardDistributionScheduler(_rewardDistributionScheduler);
136       cakePlatform = ICakePlatform(_cakePlatform);
137   }
```

**Listing 2.11:** UniversalProxy.sol

```
347   function setRevenuePoolIds(address[] memory poolIds) external onlyRole(MANAGER) {
348       revenueSharingPools = poolIds;
349       emit RevenuePoolIdsSet(poolIds);
350   }
```

**Listing 2.12:** UniversalProxy.sol

**Suggestion** Revise the logic to ensure the validation is consistent.

### 2.2.4 Add non-zero checks in `add1InchRouterWhitelist()` and `addSwapSrcTokenWhitelist()` functions

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the `Buyback` contract, the `add1InchRouterWhitelist()` function lacks a check for the `oneInchRouter` parameter, and the `addSwapSrcTokenWhitelist()` function does not check the `srcToken` parameter. It is recommended to add a non-zero address check to ensure that these parameters are valid addresses.

```
 91   function initialize(
 92       address _admin,
 93       address _pauser,
 94       address _minter,
 95       address _bot,
 96       address _manager,
 97       address _token,
 98       address _veToken,
 99       address _gaugeVoting,
100       address _ifo,
101       address _rewardDistributionScheduler,
102       address[] memory _revenueSharingPools,
103       address _revenueSharingPoolGateway,
104       address _cakePlatform
105   ) external initializer {
106       require(_admin != address(0), "Invalid admin address");
107       require(_token != address(0), "Invalid token address");
108       require(_veToken != address(0), "Invalid veToken address");
109       require(_gaugeVoting != address(0), "Invalid gaugeVoting address");
110       require(_ifo != address(0), "Invalid ifo address");
111       require(_revenueSharingPoolGateway != address(0), "Invalid revenueSharingPoolGateway
              address");
112       require(_cakePlatform != address(0), "Invalid cakePlatform address");
113       require(_rewardDistributionScheduler != address(0), "Invalid rewardDistributionScheduler
              address");
114       require(_revenueSharingPools.length > 0, "revenueSharingPools must have at least one
              address");
115
116
117       __AccessControl_init();
118       __Pausable_init();
119       __ReentrancyGuard_init();
```

```
120         __UUPSUpgradeable_init();
121         _grantRole(DEFAULT_ADMIN_ROLE, _admin);
122         _grantRole(PAUSER, _pauser);
123         _grantRole(MINTER, _minter);
124         _grantRole(BOT, _bot);
125         _grantRole(MANAGER, _manager);
126
127
128         lockCreated = false;
129         token = IERC20(_token);
130         veToken = IVeCake(_veToken);
131         gaugeVoting = IGaugeVoting(_gaugeVoting);
132         ifo = IIFOV8(_ifo);
133         revenueSharingPools = _revenueSharingPools;
134         revenueSharingPoolGateway = IRevenueSharingPoolGateway(_revenueSharingPoolGateway);
135         rewardsDistributionScheduler = IRewardDistributionScheduler(_rewardDistributionScheduler);
136         cakePlatform = ICakePlatform(_cakePlatform);
137     }
```

**Listing 2.13:** UniversalProxy.sol

```
243     function addSwapSrcTokenWhitelist(address srcToken) external onlyRole(MANAGER) {
244         require(!swapSrcTokenWhitelist[srcToken], "srcToken already whitelisted");
245         // add srcToken to swapSrcTokenWhitelist
246         swapSrcTokenWhitelist[srcToken] = true;
247         emit SwapSrcTokenChanged(srcToken, true);
248     }
```

**Listing 2.14:** Buyback.sol

**Suggestion** Add a non-zero addresses check for the `oneInchRouter` and `srcToken` parameters to prevent invalid zero addresses from being added to the whitelist.

### 2.2.5 Redundant event definitions in IWithdrawVault contract

**Status** Confirmed

**Introduced by** `Version 1`

**Description** In the `IWithdrawVault` contract, several events are defined but never emitted, making them redundant.

```
 8   interface IWithdrawVault {
 9
10
11      event ReceiveETH(address indexed from, address indexed to, uint256 amount);
12      event TransferNative(address receipt, uint256 amount);
13      event Transfer(address receipt, IERC20 token, uint256 amount);
14
15
16      function transfer(address receipt, IERC20 token, uint256 amount) external;
17      function transferNative(address receipt, uint256 amount) external;
18
19  }
```

**Listing 2.15:** IWithdrawVault.sol

**Suggestion** Remove the redundant event definitions from the contract, as they are not being emitted.

**Feedback from the project** Currently, we not support `NativeToken` so not use it, just for future functionality.

### 2.2.6 Grant the BOT role in function `initialize()` of Buyback contract

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the `Buyback` contract, the `buyback()` function uses the `onlyRole(BOT)` modifier to restrict access to the `BOT` role. However, the role is not granted during the contract initialization. This means that the `ADMIN` must manually invoke the `grantRole()` function after the contract is initialized. To simplify the process and enhance contract functionality, it is recommended to grant the `BOT` role directly in the `initialize()` function.

```
79   function initialize(
80       address _admin,
81       address _manager,
82       address _pauser,
83       address _swapDstToken,
84       address _receiver,
85       address _oneInchRouter,
86       address _swapNativeAddress
87   ) external override initializer {
88       require(_admin != address(0), "Invalid admin address");
89       require(_manager != address(0), "Invalid _manager address");
90       require(_pauser != address(0), "Invalid _pauser address");
91       require(_swapDstToken != address(0), "Invalid swapDstToken address");
92       require(_receiver != address(0), "Invalid receiver address");
93       require(_oneInchRouter != address(0), "Invalid oneInchRouter address");
94       require(_swapNativeAddress != address(0), "Invalid swapNativeAddress address");
95
96
97       __AccessControl_init();
98       __Pausable_init();
99       __ReentrancyGuard_init();
100      __UUPSUpgradeable_init();
101
102
103      _grantRole(DEFAULT_ADMIN_ROLE, _admin);
104      _grantRole(MANAGER, _manager);
105      _grantRole(PAUSER, _pauser);
106
107
108      swapDstToken = _swapDstToken;
109      receiver = _receiver;
110      swapNativeAddress = _swapNativeAddress;
```

```
111         oneInchRouterWhitelist[_oneInchRouter] = true;
112     }
```

**Listing 2.16:** Buyback.sol

**Suggestion** Grant the `BOT` role directly in the `initialize()` function to avoid manual assignment after contract deployment.

### 2.2.7 Incorrect comment in function `_smartMint()`

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** The `_smartMint()` function in the `Minter` contract contains a misleading comment. The comment "`swap to asCAKE by pancakeSwap`" inaccurately describes the implementation, as the section of the code is responsible for minting tokens, not for performing a token swap.

```
475     function _smartMint(uint256 _amountIn, uint256 _mintRatio, uint256 _minOut) private returns (
            uint256) {
476         // valid amount
477         require(_amountIn > 0, "Invalid amount");
478         // valid mintRatio
479         require(_mintRatio <= DENOMINATOR, "Incorrect Ratio");
480         // transfer funds to this contract
481         token.safeTransferFrom(msg.sender, address(this), _amountIn);
482         //convert CAKE amount by this contract
483         uint256 mintAmount = (_amountIn * _mintRatio) / DENOMINATOR;
484         // swap CAKE amount by pancakeSwap
485         uint256 buybackAmount = _amountIn - mintAmount;
486         // asCAKE amount
487         uint256 amountRec = 0;
488
489
490         if (mintAmount > 0) {
491           // swap to asCAKE by pancakeSwap
492           amountRec += _mint(mintAmount);
493           // increase total CAKE
494           _totalTokens += mintAmount;
495         }
496
497
498         if (buybackAmount > 0) {
499           // make sure minOut is not 0
500           require(_minOut > amountRec, "MinOut not match");
501           //swap CAKE by pancakeSwap
502           amountRec += _buyback(buybackAmount, _minOut - amountRec);
503         }
504         // the received amount is greater than equal to the estimated amount
505         require(amountRec >= _minOut, "MinOut not match");
506
507
```

```
508         // transfer asCAKE to sender
509         IERC20(assToken).safeTransfer(msg.sender, amountRec);
510
511
512         // emit event
513         emit SmartMinted(msg.sender, _amountIn, amountRec);
514
515
516         return amountRec;
517     }
```

**Listing 2.17:** Minter.sol

**Suggestion** Update the comment to accurately reflect the functionality.

## 2.3 Notes

### 2.3.1 Potential centralization risk

**Introduced by** Version 1

**Description** In the current implementation, several privileged roles are set to govern and regulate the system-wide operation (e.g., parameter setting, pause/unpause and grant roles). Additionally, the `owner` also has the ability to upgrade contracts. If the private keys of them are lost or maliciously exploited, it could potentially lead to losses for users.

**Feedback from the project** We will use a time-lock contract with minDelay 6 hours to perform any contract upgrade. Also, all proposers and executors are multi-sig wallet.

### 2.3.2 Timely distributing rewards before updating rewardsSender in function `setRewardsSender()`

**Introduced by** Version 1

**Description** In the `YieldProxy` contract, when native tokens are received, the contract checks if the `msg.sender` matches the designated `rewardsSender` to determine whether the tokens are user rewards. If they are identified as user rewards, the amount is added to the global variable `rewardedAmount`, which is later converted into `slisbnb` for distribution to users. However, the `rewardsSender` can be changed through the privileged function `setRewardsSender()`. If rewards are not distributed promptly before `rewardsSender` is updated, it could result in issues with reward distribution, potentially causing user losses.

### 2.3.3 Timely invocation of the function `settleActivity()`

**Introduced by** Version 1

**Description** The protocol aggregates users' `slisBNB`, deposits them into `Lista` to receive `clisBNB`, and mints `asBNB` for the users. When rewards reach a certain threshold, they are processed via the `settleActivity()` function, adjusting the exchange rate between `asBNB` and `slisBNB`. As a result, when users burn `asBNB` to redeem `slisBNB`, they will receive more than

their initial deposit. However, the function `settleActivity()` is actively invoked by the `BOT` role once rewards are distributed. It is crucial that the `BOT` triggers the function promptly. Otherwise, users may miss out on their rewards.

**Feedback from the project** The `Astherus` team will ensure the high availability of the off-chain service.

### 2.3.4 Potential DoS due to expired lock in VeCake contract

**Introduced by** `Version 1`

**Description** In the `ass-cake-earn`'s `UniversalProxy` contract, the `BOT` role must timely invoke the privileged function `extendLock()` to extend the lock duration in the `VeCake` contract, ensuring that it does not expire. If the lock expires, the global variable `lockCreated` in the contract cannot revert from `true` to `false`, preventing the contract from creating a new lock in the `VeCake` contract. In such a case, any attempt by the contract to lock new `CAKE` into `VeCake` by invoking the function `increaseLockAmount()` will result in a DoS.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS