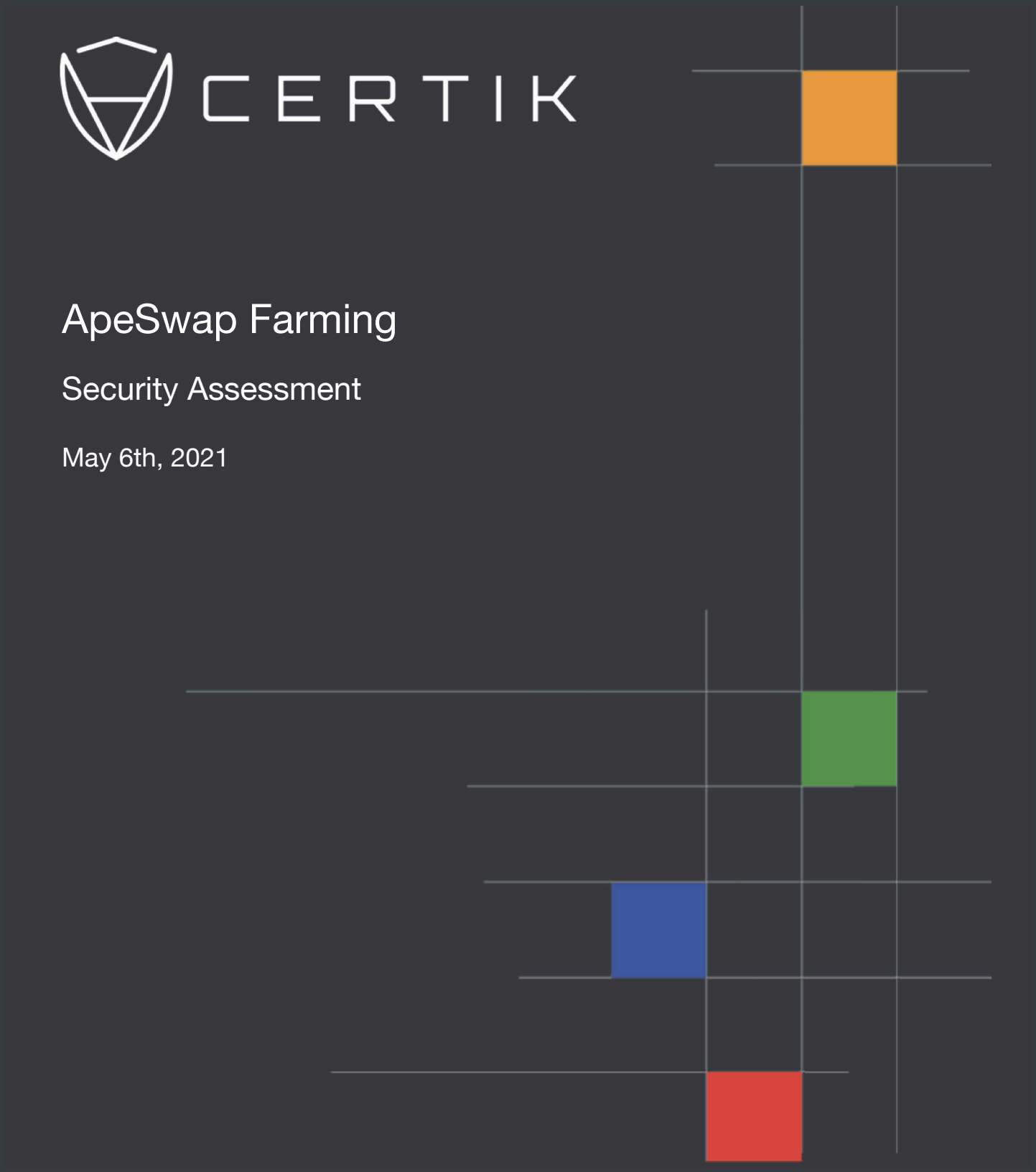


ApeSwap Farming

Security Assessment

May 6th, 2021

For :
ApeSwap

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	ApeSwap
Description	DeFi
Platform	Binance Smart Chain; Solidity
Codebase	GitHub Repository
Commit	93d10f3c680d28eb2bec32db59d69c17f3e16c2c

Audit Summary

Delivery Date	May 6th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Mar. 2th, 2021 - Mar. 6th, 2021, May 2, 2021 - May 4, 2021, May 6, 2021

Vulnerability Summary

Total Issues	15
Total Major	2
Total Minor	5
Total Informational	8

Executive Summary

This report has been prepared for **ApeSwap** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

To initial setup project correctly, improve overall project quality, preserve the upgradability, the following functions are adopted in the codebase:

- `setAdmin()` to update address of `adminAddress` in smart contract `BnbStaking.sol`.
- `setBlackList()` to label new address in `userInfo` in smart contract `BnbStaking.sol`.
- `removeBlackList()` to disable new address in `userInfo` in smart contract `BnbStaking.sol`.
- `setLimitAmount()` to update the value of `limitAmount` in smart contract `BnbStaking.sol`.
- `setReceiver()` to update value of `receiver` in smart contract `LotteryRewardPool.sol`.
- `setAdmin()` to update value of `adminAddress` in smart contract `LotteryRewardPool.sol`.
- `updateMultiplier()` to update value of `BONUS_MULTIPLIER` in smart contract `MasterApe.sol`.
- `dev()` to update value of `devaddr` in smart contract `MasterApe.sol`.

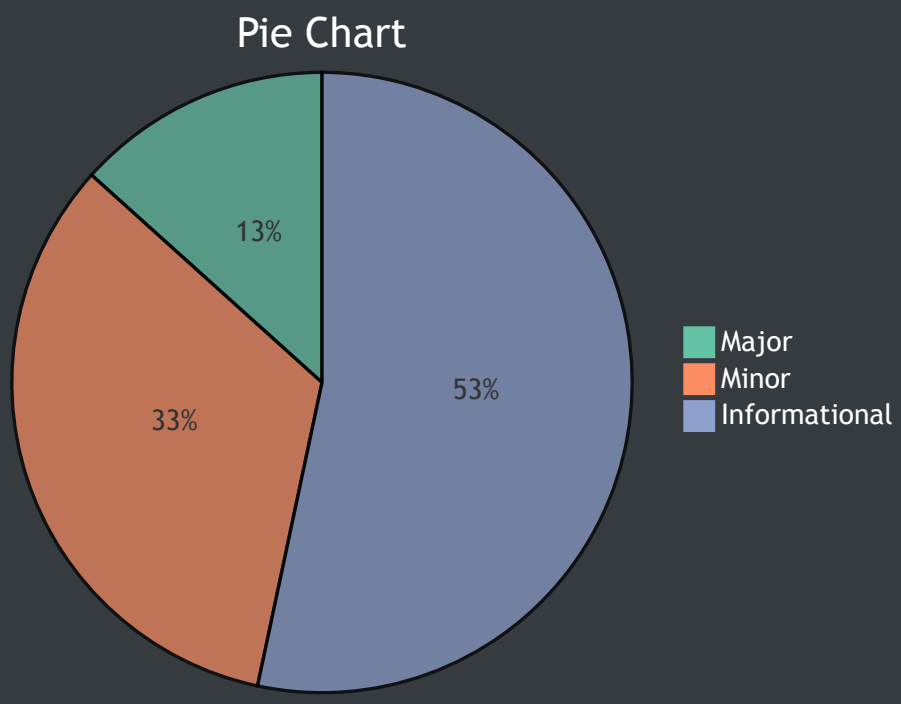
The advantage of the above functions in the codebase is that the client reserves the ability to adjust the project according to the runtime require to best serve the community. It is also worthy of note the potential drawbacks of these functions, which should be clearly stated through client's action/plan on how to prevent abusage of the these functionalities

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. Any plan to invoke abovementioned functions should be also considered to move to the execution queue of Timelock contract.

File in Scope

ID	Contract	SHA-256 Checksum
BRA	BNBRewardApe.sol	36daf735d4ab892025b26d7e66b3b7fcf7637a2fd49084a39007327434e986b9
BTK	BananaToken.sol	f27f343072cc4441d4526f28a5e625d0a39d2b9479e461916df94e6fbdb51bd7
BSK	BnbStaking.sol	1a769db970b93080c78f41ac9ff57144261a3a21d65e72d87915458f65c7db62
LRP	LotteryRewardPool.sol	e78cda66e3e4ce3e15ade0ee1188c5ab41570c484b01188e61edf45c653d8b0c
MAP	MasterApe.sol	f253343ed07bd06b1bc4c75b493970a0ee60e4e9f9a93d4e7489cda4de08fb0f
SAP	SupportApe.sol	6567d3d55763e599829a87667753800cb697473e58899a2f9e7f18a4a7b4893d
TLK	Timelock.sol	13c7c7d0005cc311e4613b693fee23765f78d75626ff0143928177cf93772293
MBEP	MockBEP20.sol	1b0605d512d87cde111e9559ec1c935b11e788f053bcfbeb69392337f2efdb03
MTC	Multicall.sol	56ab5b5757b61699d1f6c51e42ff682876899abfd2ab4cedf97f36e4d0e9e649
WBNB	WBNB.sol	92020f2413c24891fa83b1e47ce5ff0471eb16618322bb2cb2af8ec5ede8d660
IERC	IERC20.sol	49b7bee5c2f36e6ed17b17c9cb88712b1298a0641b92c4131ec75a2b5fa513ea
IMA	IMasterApe.sol	2a65874ab06d26736e91cdcdf20226947b611796a004c9cf0ab695768c3d3d0f

Findings



ID	Title	Type	Severity	Resolved
BSK-01	Missing Emit Events	Optimization	Informational	
BSK-02	Check Effect Interaction Pattern Violated	Logical Issue	Minor	
BSK-03	Division Before Multiplication	Logical Issue	Informational	
BSK-04	Proper Usage of public and external type	Optimization	Informational	
LRP-01	Missing Emit Events	Optimization	Informational	
MAP-01	Variable Naming Convention	Coding Style	Informational	
MAP-02	Centralized Control of Bonus Multiplier	Logical Issue	Informational	
MAP-03	Missing Emit Events	Optimization	Informational	
MAP-04	Incompatibility With Deflationary Tokens	Logical Issue	Minor	

				
MAP-05	Over Minted Token	Logical Issue	Minor	
MAP-06	Previous PancakeSwap Hack	Logical Issue	Major	
SAP-01	`addressList` Inaccuracy	Logical Issue	Minor	
SAP-02	Incorrect Reset Mechanism	Logical Issue	Minor	
WBNB-01	Confusing Transfer() Event Usage	Optimization	Informational	
RTK-01	Delegation Not Moved Along With `transfer()`	Logical Issue	Major	

BSK-01: Missing Emit Events

Type	Severity	Location
Optimization	Informational	BnbStaking.sol: L111, L115, L119, L124

Description:

Function that affect the status of sensitive variables should be able to emit events as notifications to customers:

- `setAdmin()`
- `setBlackList()`
- `removeBlackList()`
- `setLimitAmount()`

Recommendation:

Consider adding events for sensitive actions, and emit it in the function like below:

```
event SetAdmin(address indexed user, address indexed _adminAddress);

function setAdmin(address _adminAddress) public onlyOwner {
    adminAddress = _adminAddress;
    emit SetAdmin(msg.sender, _adminAddress);
}
```

Alleviation:

`[ApeSwap]` :This contract has been deprecated and currently no value is being stored in any contracts that use this code, deployed by ApeSwap. With that being said, these are most definitely items we will keep on our checklist

BSK-02: Check Effect Interaction Pattern Violated

Type	Severity	Location
Logical Issue	Minor	BnbStaking.sol: L197, L237

Description:

The order of external call/transfer and storage manipulation must follow check effect interaction pattern.

Recommendation:

We advise client to check if storage manipulation is before the external call/transfer operation.LINK

Alleviation:

`[ApeSwap]` :This contract has been deprecated and currently no value is being stored in any contracts that use this code, deployed by ApeSwap. With that being said, these are most definitely items we will keep on our checklist

BSK-03: Division Before Multiplication

Type	Severity	Location
Logical Issue	Informational	BnbStaking.sol: L140, L154, L180

Description:

Mathematical operations in the aforementioned function perform divisions before multiplications. Performing multiplication before division can sometimes avoid loss of precision.

Recommendation:

We recommend applying multiplications before divisions if integer overflow would not happen in functions.

Alleviation:

`[ApeSwap]` :This contract has been deprecated and currently no value is being stored in any contracts that use this code, deployed by ApeSwap. With that being said, these are most definitely items we will keep on our checklist

BSK-04: Proper Usage of `public` and `external` type

Type	Severity	Location
Optimization	Informational	BnbStaking.sol: L213

Description:

Public functions that are never called by the contract could be declared external . When the inputs are arrays external functions are more efficient than “public” functions.public functions that are never called by the contract could be declared external . When the inputs are arrays external functions are more efficient than “public” functions.

Recommendation:

Consider using the external attribute for functions never called from the contract.

Alleviation:

`[ApeSwap]` :This contract has been deprecated and currently no value is being stored in any contracts that use this code, deployed by ApeSwap. With that being said, these are most definitely items we will keep on our checklist

LRP-01: Missing Emit Events

Type	Severity	Location
Optimization	Informational	LotteryRewardPool.sol: L62, L76

Description:

Function that affect the status of sensitive variables should be able to emit events as notifications to customers

- `setReceiver()`
- `setAdmin()`

Recommendation:

Consider adding events for sensitive actions, and emit it in the function like below.

```
event SetAdmin(address indexed user, address indexed _adminAddress);

function setAdmin(address _admin) external onlyOwner {
    adminAddress = _admin;
    emit SetAdmin(msg.sender, _admin);
}
```

Alleviation:

N/A

MAP-01: Variable Naming Convention

Type	Severity	Location
Coding Style	Informational	MasterApe.sol: L69

Description:

The linked variables do not conform to the standard naming convention of Solidity whereby functions and variable names utilize the format, unless variables are declared as constant in which case they utilize the format.

Recommendation:

We advise that the naming conventions utilized by the linked statements are adjusted to reflect the correct type of declaration according to the Solidity style guide.

Alleviation:

N/A

MAP-02: Centralized Control of Bonus Multiplier

Type	Severity	Location
Logical Issue	Informational	MasterApe.sol: L117

Description:

The function can alter the `BONUS_MULTIPLIER` variable and consequently the output of which is directly utilized for the minting of new cake tokens.

Recommendation:

This is intended functionality of the protocol, however users should be aware of this functionality.

Alleviation:

`[ApeSwap]` :We have put the MasterApe contract behind a Timelock to give users an indication of a change. A possible solution is disabling control of that function by creating a partial interfaced proxy contract that becomes the owner of the MasterApe.

MAP-03: Missing Emit Events

Type	Severity	Location
Optimization	Informational	MasterApe.sol: L331, L117

Description:

Function that affect the status of sensitive variables should be able to emit events as notifications to customers:

- `dev()`
- `updateMultiplier()`

Recommendation:

Consider adding events for sensitive actions, and emit it in the function.

```
event SetDev(address indexed user, address indexed _devaddr);

function dev(address _devaddr) public {
    require(msg.sender == devaddr, "dev: wut?");
    devaddr = _devaddr;
    emit SetDev(msg.sender, _devaddr);
}
```

Alleviation:

N/A

MAP-04: Incompatibility With Deflationary Tokens

Type	Severity	Location
Logical Issue	Minor	MasterApe.sol: L225, L247

Description:

In Ape Swap, the MasterChef contract operates as the main entry for interaction with staking users. The staking users deposit LP tokens into the Ape Swap pool and in return get proportionate share of the pool's rewards. Later on, the staking users can withdraw their own assets from the pool. In this procedure, `deposit()` and `withdraw()` are involved in transferring users' assets into (or out of) the Ape Swap protocol. When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged (and burned) transaction fee. As a result, this may not meet the assumption behind these low-level asset-transferring routines and will bring unexpected balance inconsistencies.

Recommendation:

Regulate the set of LP tokens supported in Ape Swap and, if there is a need to support deflationary tokens, add necessary mitigation mechanisms to keep track of accurate balances.

Alleviation:

`[ApeSwap]` :This is an important issue which we will mitigate by not adding deflationary/reflect tokens to the MasterApe contract. If we do add a pool that is affected by this issue we will deprecate it as soon as possible by setting the reward allocation to zero.

MAP-05: Over Minted Token

Type	Severity	Location
Logical Issue	Minor	MasterApe.sol: L218, L219

Description:

`updatePool()` function minted 100% + 10% (dev fee is included as 10% of the 100%) of total rewards.

Recommendation:

Fix to mint 100% of the block reward instead of 100% + 10% (dev fee is included as 10% of the 100%) like in other MasterChef clones.

Alleviation:

N/A

MAP-06: Previous PancakeSwap Hack

Type	Severity	Location
Logical Issue	Major	MasterApe.sol: L308

Description:

An exploit in the interaction between the `MasterChef` contract and the `SyrupBar` contract was abused by bad actors. Previously when `CakeToken` was staked, an equal amount of `SyrupBar` tokens would be minted. Once the `CakeToken` was unstaked and withdrawn, the `SyrupBar` tokens would be burned. The specific exploit here was that if a user used the `emergencyWithdraw()` function in the `MasterChef` contract to withdraw their staked `CakeToken`, the corresponding `SyrupBar` tokens would not be burnt as intended. This allowed bad actors to repeatedly mint more `SyrupBar` tokens with their `CakeToken` tokens.

Recommendation:

As the `MasterApe`, `BananaToken` and `BananaSplitBar` have the similar functionality with PancakeSwap's `MasterChef`, `CakeToken` and `SyrupBar` respectively, to protect the ApeSwap from such attack vector, we advise the client to make changes as following in `emergencyWithdraw()` function.

```
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    if(_pid == 0) {
        syrup.burn(msg.sender, user.amount);
    }
    pool.lpToken.safeTransfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    user.amount = 0;
    user.rewardDebt = 0;
}
```

Alleviation:

`[ApeSwap]` :This paragraph describes a scenario where the MasterApe contract was exploited, but the header infers this is meant to describe the PancakeSwap Hack. We currently have no use for the BananaSplitToken and have no plans of adding any utility to it keeping it valueless.

SAP-01: `addressList` Inaccuracy

Type	Severity	Location
Logical Issue	Minor	SupportApe.sol L141, L168

Description:

The first linked if block pushes a new userInfo address to the addressList array in the case the userInfo mapping lookup yields 0 on the amount member This case is possible even after the user has already been added to the array, either by invoking emergencyWithdraw or withdrawing the full amount held by the user.

Recommendation:

We advise that the push mechanism is revised to ensure that the user does not already exist in the array.

Alleviation:

N/A

SAP-02: Check Effect Interaction Pattern Violated

Type	Severity	Location
Logical Issue	Minor	SupportApe.sol L148

Description:

The order of external call/transfer and storage manipulation must follow check effect interaction pattern.

Recommendation:

We advise client to check if storage manipulation is before the external call/transfer operation.LINK

Alleviation:

N/A

WBNB-01: Confusing Transfer() Event Usage

Type	Severity	Location
Optimization	Informational	WBNB.sol L58

Description:

Use Transfer() to emit an event associated with a transfer operation

Recommendation:

We advise client to use `emit` keyword to emit event instead of using `Transfer` event directly.

Alleviation:

N/A

BTK-01: Delegation Not Moved Along With `transfer()`

Type	Severity	Location
Logical Issue	Major	BananaToken.sol L1, BananaSplitBar.sol L1

Description:

The voting power of delegation is not moved from token sender to token recipient along with the `transfer()`. Current `transfer()` is from `BEP20` protocol and don't invoke `_moveDelegates()`.

Recommendation:

We advise client to consider adopting a specific implementation of the `transfer()` function that has a `_moveDelegates()` logic called upon transferring.

Reference:
https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAM.sol#L108

Alleviation:

N/A

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.