CERTIK

Preliminary Comments

SMARS

May 19th, 2021

Table of Contents

Summary

Overview

- Project Summary
- Audit Summary
- Vulnerability Summary
- Audit Scope

Findings

- SMC-01 : Typos in the contract
- SMC-02 : Incorrect error message
- SMC-03 : Contract gains non-withdrawable BNB via the `swapAndLiquify` function
- SMC-04 : Return value not handled
- SMC-05 : Centralized risk in `addLiquidity`
- SMC-06 : Redundant code
- SMC-07 : Variable could be declared as `constant`
- SMC-08 : 3rd party dependencies
- SMC-09 : Missing event emitting
- SMC-10 : Privileged ownership
- SMC-11 : The purpose of function `deliver`
- SMC-12 : Possible to gain ownership after renouncing the contract ownership

Appendix

Disclaimer

About

Summary

This report has been prepared for SMARS smart contracts, 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 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.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	SMARS
Platform	BSC
Language	Solidity
Codebase	https://bscscan.com/address/0xC0366a104b429f0806BfA98d0008DAA9555b2BEd#code
Commits	Deployed contract address: 0xC0366a104b429f0806BfA98d0008DAA9555b2BEd

Audit Summary

Delivery Date	May 19, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

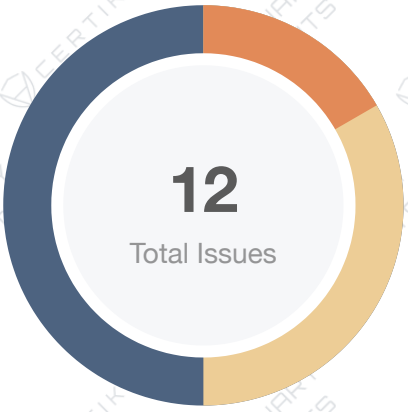
Vulnerability Summary

Total Issues	12
Critical	0
Major	2
Medium	0
Minor	4
Informational	6
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
SMC	SafeMars.sol	f90164092172ae6aea6d665923a4e897933c8258739e0ac604be73e5eb9afd1e

Findings



Critical	0 (0.00%)
Major	2 (16.67%)
Medium	0 (0.00%)
Minor	4 (33.33%)
Informational	6 (50.00%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
SMC-01	Typos in the contract	Coding Style	Informational	Pending
SMC-02	Incorrect error message	Logical Issue	Minor	Pending
SMC-03	Contract gains non-withdrawable BNB via the `swapAndLiquify` function	Logical Issue	Major	Pending
SMC-04	Return value not handled	Volatile Code	Informational	Pending
SMC-05	Centralized risk in `addLiquidity`	**Centralization / Privilege**	**Major**	**Pending**
SMC-06	Redundant code	Logical Issue	Informational	Pending
SMC-07	Variable could be declared as `constant`	Gas Optimization	Informational	Pending
SMC-08	3rd party dependencies	Control Flow	Minor	Pending
SMC-09	Missing event emitting	Coding Style	Informational	Pending
SMC-10	Privileged ownership	**Centralization / Privilege**	**Minor**	**Pending**
SMC-11	The purpose of function `deliver`	Control Flow	Informational	Pending
SMC-12	Possible to gain ownership after renouncing the contract ownership	**Logical Issue, Centralization / Privilege**	**Minor**	**Pending**

SMC-01 | Typos in the contract

Category	Severity	Location	Status
Coding Style	Informational	SafeMars.sol: 937, 1177	Pending

Description

There are several typos in the code and comments.

1. In the following code snippet, `tokensIntoLiqudity` should be `tokensIntoLiquidity`.

```
event SwapAndLiquify(
        uint256 tokensSwapped,
        uint256 ethReceived,
        uint256 tokensIntoLiqudity
    );
```

2. `recieve` should be `receive` and `swaping` should be `swapping` in the line of comment `//to recieve ETH from uniswapV2Router when swaping`.

Recommendation

We recommend correcting all typos in the contract.

SMC-02 | Incorrect error message

Category	Severity	Location	Status
Logical Issue	Minor	SafeMars.sol: 1118	Pending

Description

The error message in `require(_isExcluded[account], "Account is already excluded")` does not describe the error correctly.

Recommendation

The message "Account is already excluded" can be changed to "Account is not excluded" .

SMC-03 | Contract gains non-withdrawable BNB via the `swapAndLiquify` function

Category	Severity	Location	Status
Logical Issue	Major	SafeMars.sol: 1367	Pending

Description

The `swapAndLiquify` function converts half of the `contractTokenBalance` SMARS tokens to BNB. The other half of SMARS tokens and part of the converted BNB are deposited into the SMARS-BNB pool on pancakeswap as liquidity. For every `swapAndLiquify` function call, a small amount of BNB leftover in the contract. This is because the price of SMARS drops after swapping the first half of SMARS tokens into BNBs, and the other half of SMARS tokens require less than the converted BNB to be paired with it when adding liquidity. The contract doesn't appear to provide a way to withdraw those BNB, and they will be locked in the contract forever.

Recommendation

It's not ideal that more and more BNB are locked into the contract over time. The simplest solution is to add a `withdraw` function in the contract to withdraw BNB. Other approaches that benefit the SMARS token holders can be:

- Distribute BNB to SMARS token holders proportional to the amount of token they hold.
- Use leftover BNB to buy back SMARS tokens from the market to increase the price of SMARS.

SMC-04 | Return value not handled

Category	Severity	Location	Status
Volatile Code	Informational	SafeMars.sol: 1413~1420	Pending

Description

The return values of function `addLiquidityETH` are not properly handled.

```
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            owner(),
            block.timestamp
        );
```

Recommendation

We recommend using variables to receive the return value of the functions mentioned above and handle both success and failure cases if needed by the business logic.

SMC-05 | Centralized risk in `addLiquidity`

Category	Severity	Location	Status
Centralization / Privilege	Major	SafeMars.sol: 1413~1420	Pending

Description

```
// add the liquidity
uniswapV2Router.addLiquidityETH{value: ethAmount}(
    address(this),
    tokenAmount,
    0, // slippage is unavoidable
    0, // slippage is unavoidable
    owner(),
    block.timestamp
);
```

The `addLiquidity` function calls the `uniswapV2Router.addLiquidityETH` function with the `to` address specified as `owner()` for acquiring the generated LP tokens from the `SMARS-BNB` pool. As a result, over time the `_owner` address will accumulate a significant portion of LP tokens.If the `_owner` is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole.

Recommendation

We advise the `to` address of the `uniswapV2Router.addLiquidityETH` function call to be replaced by the contract itself, i.e. `address(this)`, and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the `_owner` account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

SMC-06 | Redundant code

Category	Severity	Location	Status
Logical Issue	Informational	SafeMars.sol: 1437	Pending

Description

The condition `!_isExcluded[sender] && !_isExcluded[recipient]` can be included in `else` .

Recommendation

The following code can be removed:

```
... else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
    _transferStandard(sender, recipient, amount);
} ...
```

SMC-07 | Variable could be declared as `constant`

Category	Severity	Location	Status
Gas Optimization	Informational	SafeMars.sol	Pending

Description

Variables `_tTotal`, `numTokensSellToAddToLiquidity`, `_name`, `_symbol` and `_decimals` could be declared as `constant` since these state variables are never to be changed.

Recommendation

We recommend declaring those variables as `constant`.

SMC-08 | 3rd party dependencies

Category	Severity	Location	Status
Control Flow	Minor	SafeMars.sol	Pending

Description

The contract is serving as the underlying entity to interact with third party PancakeSwap protocols. The scope of the audit would treat those 3rd party entities as black boxes and assume its functional correctness. However in the real world, 3rd parties may be compromised that led to assets lost or stolen.

Recommendation

We understand that the business logic of the SafeMars protocol requires the interaction PancakeSwap protocol for adding liquidity to SMARS-BNB pool and swap tokens. We encourage the team to constantly monitor the statuses of those 3rd parties to mitigate the side effects when unexpected activities are observed.

SMC-09 | Missing event emitting

Category	Severity	Location	Status
Coding Style	Informational	SafeMars.sol	Pending

Description

In contract `SafeMars`, there are a bunch of functions can change state variables. However, these function do not emit event to pass the changes out of chain.

Recommendation

Recommend emitting events, for all the essential state variables that are possible to be changed during runtime.

SMC-10 | Privileged ownership

Category	Severity	Location	Status
Centralization / Privilege	Minor	SafeMars.sol	Pending

Description

The owner of contract `SafeMars` has the permission to:

1. change the address that can receive LP tokens,
2. lock the contract,
3. exclude/include addresses from rewards/fees,
4. set `taxFee`, `liquidityFee` and `_maxTxAmount`,
5. enable `swapAndLiquifyEnabled`

without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations.

SMC-11 | The purpose of function `deliver`

Category	Severity	Location	Status
Control Flow	Informational	SafeMars.sol	Pending

Description

The function `deliver` can be called by anyone. It accepts an uint256 number parameter `tAmount`. The function reduces the SMARS token balance of the caller by `rAmount`, which is `tAmount` reduces the transaction fee. Then, the function adds `tAmount` to variable `_tFeeTotal`, which represents the contract's total transaction fee. We wish the team could explain more on the purpose of having such functionality.

SMC-12 | Possible to gain ownership after renouncing the contract ownership

Category	Severity	Location	Status
Logical Issue, Centralization / Privilege	Minor	SafeMars.sol	Pending

Description

An owner is possible to gain ownership of the contract even if he calls function `renounceOwnership` to renounce the ownership. This can be achieved by performing the following operations:

1. Call `lock` to lock the contract. The variable `_previousOwner` is set to the current owner.
2. Call `unlock` to unlock the contract.
3. Call `renounceOwnership` to leave the contract without an owner.
4. Call `unlock` to regain ownership.

Recommendation

We advise updating/removing `lock` and `unlock` functions in the contract; or removing the `renounceOwnership` if such a privilege retains at the protocol level. If timelock functionality could be introduced, we recommend using the implementation of Compound finance as reference. Reference: https://github.com/compound-finance/compound-protocol/blob/master/contracts/Timelock.sol

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings 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.

Logical Issue

Logical Issue findings 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.

Coding Style

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

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.