Security Assessment

ButterSwap

Jun 9th, 2021

Table of Contents

Disclaimer

About

Summary

This report has been prepared for ButterSwap 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 Manual Review and Static Analysis 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	ButterSwap
Platform	Heco
Language	Solidity
Codebase	• https://github.com/butter-swap/butter-swap-farm • https://github.com/butter-swap/butter-swap-core • https://github.com/butter-swap/butter-swap-periphery
Commits	• butter-swap-farm: 87558e358302ed3dee80b0b152449998d36cbfc9 • butter-swapcore: 02d785381610b5a64f1f623b24a3ec5db330cb88 • butter-swap-periphery: d167939a0c065c968fe085bbaa962dab6c79c785

Audit Summary

Delivery Date	Jun 09, 2021
Audit Methodology	Manual Review, Static Analysis
Key Components	

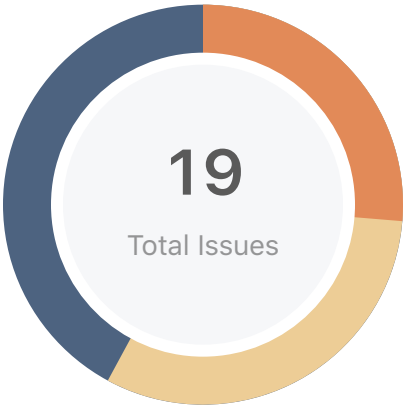
Vulnerability Summary

Total Issues	19
Critical	0
Major	5
Medium	0
Minor	6
Informational	8
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
BTC	butter-farm/contracts/ButterToken.sol	90bcc9e251fa6d358ea200fb005af67b3f741473be63e8b2d2e6971bc734d626
CTC	butter-farm/contracts/CreamToken.sol	815cf09c4479592fc76994b0b502f0605bfa6cdb7fd10d2cbb60f7c0160d2d6d
MCC	butter-farm/contracts/MasterChef.sol	5298ec776b33781f8aaa9a2afbec69130a6ff2a3c51444ddfac5e0b9482ef41a
MCK	butter-farm/contracts/Migrations.sol	4fd6092bdfa8b42f19d535c5ac69c4323b0b894717c699e58d5552eeabd04cd4
SCC	butter-farm/contracts/SousChef.sol	fb5eabf60dce395ace6d6e00eb80cf30415c195f9a74d09be1ed10990f4740bb
BER	butter-swap-core/contracts/ButterERC20.sol	e7db8c9602c7d010a3e0cd84450231b866908415203765e97ec5b27f1f0e95a1
BFC	butter-swap-core/contracts/ButterFactory.sol	74710416d4f81374867a52da81583e60db47172768692facac6e31f5e5a4d1fd
BPC	butter-swap-core/contracts/ButterPair.sol	78eff4beabf78415d2c68e0ff250c997f3c69d08c2858365e53e0db219b57019
BMC	butter-swap-periphery/contracts/ButterMigrator.sol	542dc42e61e028e91f2390c55d4e08e56456a82ca9eed3ffefe112cf9362eb99
BRC	butter-swap-periphery/contracts/ButterRouter.sol	187d552beba8037703e87991b20d12d92ac8dc704fdc158e359a280ceb93f983
BRK	butter-swap-periphery/contracts/ButterRouter01.sol	44d9d9e3359616b0a6316101899f7d383316b0d0b4eb0dc9678228916a7f8ea7

Findings



Critical	0 (0.00%)
Major	5 (26.32%)
Medium	0 (0.00%)
Minor	6 (31.58%)
Informational	8 (42.11%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
BFC-01	Missing Emit Events	Gas Optimization	Informational	Acknowledged
BFC-02	Unnecessary Array as Counter	Gas Optimization	Informational	Acknowledged
BPC-01	Replace Libraries with Inherited Contract in Contract Template	Gas Optimization	Minor	Acknowledged
BPC-02	Variable Declare as `Immutable`	Gas Optimization	Informational	Acknowledged
BPC-03	Divide by Zero	Logical Issue	Minor	Acknowledged
BTC-01	Does Not Move Delegates While Transferring Token	**Centralization / Privilege**	**Major**	**Acknowledged**
CTC-01	Does Not Move Delegates While Transferring Token	**Centralization / Privilege**	**Major**	**Acknowledged**
MCC-01	add() Function Not Restricted	Volatile Code	Major	Resolved
MCC-02	`Checks-effects-interactions` Pattern Not Used	Logical Issue	Major	Resolved
MCC-03	Variable Naming Convention	Coding Style	Informational	Resolved
MCC-04	Recommended Explicit Pool Validity Checks	Logical Issue	Informational	Acknowledged

ID	Title	Category	Severity	Status
MCC-05	Missing Emit Events	Gas Optimization	Informational	Acknowledged
MCC-06	Unknown Implementation of `migrator.migrate()`	Logical Issue	Minor	Acknowledged
MCC-07	Over Minted Token	Logical Issue	Minor	Acknowledged
MCC-08	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Acknowledged
MCC-09	Lack of Input Validation	Volatile Code	Minor	Resolved
SCC-01	`Checks-effects-interactions` Pattern Not Used	Logical Issue	Major	Resolved
SCC-02	Missing Emit Events	Gas Optimization	Informational	Acknowledged
SCC-03	Redundant Variable	Gas Optimization	Informational	Acknowledged

BFC-01 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	Informational	butter-swap-core/contracts/ButterFactory.sol: 42(ButterFactory), 47(ButterFactory)	Acknowledged

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to customers.

- `setMigrator()`
- `emergencyWithdraw()`
- `safeButterTransfer()`
- `dev()`
- `setFeeTo()`
- `setFeeToSetter()`
- `stopReward()`

Recommendation

Consider adding events for sensitive actions, and emit them 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);
    }
```

BFC-02 | Unnecessary Array as Counter

Category	Severity	Location	Status
Gas Optimization	Informational	butter-swap-core/contracts/ButterFactory.sol: 13(ButterFactory)	Acknowledged

Description

The `allPairs` array is used as a counter to maintain the number of created pairs.

Recommendation

We advise the client to replace the `allPairs` with a simple uint type counter to store the number of pairs created.

BPC-01 | Replace Libraries with Inherited Contract in Contract Template

Category	Severity	Location	Status
Gas Optimization	Minor	butter-swap-core/contracts/ButterPair.sol: (ButterPair)	Acknowledged

Description

Libraries `UQ112x112`, `SafeMath` and `Math` will be inherited by the contract `ButterPair.sol` every time a new pair is created, which will cost extra gas on creating new pairs.

Recommendation

We advise the client to include all functions of these libraries in the `ButterPair.sol` directly to save gas on creating new pairs.

Alleviation

The development team replied that, the operation "Create pair" is not frequently used. They don't want to optimize this gas cost by decreasing code readability and cleanliness

BPC-02 | Variable Declare as `Immutable`

Category	Severity	Location	Status
Gas Optimization	Informational	butter-swap-core/contracts/ButterPair.sol: 18(ButterPair)	Acknowledged

Description

Variable that only be assigned in constructor can be declare as `immutable`. Immutable state variables can be assigned during contract creation, but will remain constant throughout the life-time of a deployed contract. The big advantage of immutable is that reading them is significantly cheaper than reading from regular state variables, since immutable variables will not be stored in storage, but their values will be directly inserted into the runtime code.

Recommendation

We recommend using immutable state variable for `factory`

```
address immutable public factory;
```

BPC-03 | Divide by Zero

Category	Severity	Location	Status
Logical Issue	Minor	butter-swap-core/contracts/ButterPair.sol: 143~145(ButterPair)	Acknowledged

Description

The call to `burn()` function will fail if the value of `totalSupply` is 0.

Recommendation

We advise the client to add the following validation in the function `burn()`

```
require(totalSupply != 0, "The value of totalSupply must not be 0");
```

Alleviation

The development team replied that, adding the require check will consume more gas, in real use case, they will make sure burn is called only when totalSupply is not zero.

BTC-01 | Does Not Move Delegates While Transferring Token

Category	Severity	Location	Status
Centralization / Privilege	**Major**	butter-farm/contracts/ButterToken.sol: (ButterToken)	**Acknowledged**

Description

In essence, ButterToken and CreamToken governance lets token holders delegate their voting power to another entity. However, if that token holder then transfers the tokens to someone else, the delegator still maintains their governance power. The second token holder can now delegate tokens once again, multiplying the delegator's power by as much as necessary. The bug is that the token transfer does not call `_moveDelegates()`.

Recommendation

Consider adding call of `_moveDelegates()` in the function `_transfer()`, `_burn()` and other functions that affects the token balance. Also make sure that `_delegates` mapping is correctly initialized, otherwise, delegation will be moved to address 0.

Alleviation

The development team replied that the issue is left over by pancake, and they currently do not need voting. They will deliver modified code for audit when they do need voting.

CTC-01 | Does Not Move Delegates While Transferring Token

Category	Severity	Location	Status
Centralization / Privilege	**Major**	butter-farm/contracts/CreamToken.sol: (CreamToken)	**Acknowledged**

Description

In essence, ButterToken and CreamToken governance lets token holders delegate their voting power to another entity. However, if that token holder then transfers the tokens to someone else, the delegator still maintains their governance power. The second token holder can now delegate tokens once again, multiplying the delegator's power by as much as necessary. The bug is that the token transfer does not call `_moveDelegates()`.

Recommendation

Consider adding call of `_moveDelegates()` in the function `_transfer()`, `_burn()` and other functions that affects the token balance. Also make sure that `_delegates` mapping is correctly initialized, otherwise, delegation will be moved to address 0.

Alleviation

The development team replied that the issue is left over by pancake, and they currently do not need voting. They will deliver modified code for audit when they do need voting.

MCC-01 | add() Function Not Restricted

Category	Severity	Location	Status
Volatile Code	Major	butter-farm/contracts/MasterChef.sol: 124~137(MasterChef)	Resolved

Description

When adding the same LP token more than once. Rewards will be messed up if you do.

The total amount of reward in function `updatePool()` will be incorrectly calculated if the same LP token is added into the pool more than once in function `add()`.

However, the code does not reflect as the comment behaviors as there isn't any valid restriction on preventing this issue.

The current implementation is relying on the trust of the owner to avoid repeatedly adding same LP token to the pool, as the function will only be called by the owner.

Recommendation

Detect whether the given pool for addition is a duplicate of an existing pool. The pool addition is only successful when there is no duplicate. Using a mapping of `addresses` -> `booleans`, which can restricted the same address being added twice.

Alleviation

The team heeded our advice and removed the function in commit d2fd6d290bd5a489ed472be893d54d7382205089.

MCC-02 | `Checks-effects-interactions` Pattern Not Used

Category	Severity	Location	Status
Logical Issue	Major	butter-farm/contracts/MasterChef.sol: 228(MasterChef), 250(MasterChef), 271(MasterChef), 292(MasterChef)	Resolved

Description

During `deposit()`, `withdraw()`, `enterStaking()` and `leaveStaking()` functions call, state variables for balance are changed after transfers are done. This might lead to reentrancy issue. The order of external call/transfer and storage manipulation must follow checks-effects-interactions pattern.

Recommendation

It is recommended to follow checks-effects-interactions pattern for cases like this. It shields public functions from re-entrancy attacks. It's always a good practice to follow this pattern. checks-effects-interactions pattern also applies to ERC20 tokens as they can inform the recipient of a transfer in certain implementations.

Reference: https://docs.soliditylang.org/en/develop/security-considerations.html?highlight=check-effects%23use-the-checks-effects-interactions-pattern

Alleviation

The team heeded our advice and removed the function in commit d2fd6d290bd5a489ed472be893d54d7382205089.

MCC-03 | Variable Naming Convention

Category	Severity	Location	Status
Coding Style	Informational	butter-farm/contracts/MasterChef.sol: 71 (MasterChef)	Resolved

Description

The linked variable `BONUS_MULTIPLIER` 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

The team heeded our advice and removed the function in commit d2fd6d290bd5a489ed472be893d54d7382205089.

MCC-04 | Recommended Explicit Pool Validity Checks

Category	Severity	Location	Status
Logical Issue	Informational	butter-farm/contracts/MasterChef.sol: 140(MasterChef), 171(MasterChef), 188(MasterChef), 209(MasterChef), 228(MasterChef), 250(MasterChef), 312(MasterChef)	Acknowledged

Description

There's no sanity check to validate if a pool is existing.

Recommendation

Consider to adopt following modifier `validatePoolByPid` to functions `set()`, `migrate()`, `deposit()`, `withdraw()`, `emergencyWithdraw()`, `pendingButter()` and `updatePool()`.

```
modifier validatePoolByPid(uint256 _pid) {
    require (_pid < poolInfo . length , "Pool does not exist") ;
    _;
}
```

MCC-05 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	Informational	butter-farm/contracts/MasterChef.sol: 166(MasterChef), 312(MasterChef), 323(MasterChef), 328(MasterChef)	Acknowledged

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to customers.

- `setMigrator()`
- `emergencyWithdraw()`
- `safeButterTransfer()`
- `dev()`
- `setFeeTo()`
- `setFeeToSetter()`
- `stopReward()`

Recommendation

Consider adding events for sensitive actions, and emit them 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);
    }
```

MCC-06 | Unknown Implementation of `migrator.migrate()`

Category	Severity	Location	Status
Logical Issue	Minor	butter-farm/contracts/MasterChef.sol: 177(MasterChef)	Acknowledged

Description

This protocol has external dependencies. `setMigrator()` function can set migrator contract to any contract that implements `IMigratorChef` interface by the owner. As a result, invocation of `migrator.migrate()` in function `migrate()` may bring dangerous effects as it is unknown to the user.

Recommendation

Make sure the third-party implementations and the way these functions are called can meet the requirements.

Alleviation

The development team replied that, this methods(setMigrator) can only be called by owner, and they will make sure the _migrator is safe.

MCC-07 | Over Minted Token

Category	Severity	Location	Status
Logical Issue	Minor	butter-farm/contracts/MasterChef.sol: 221~222(MasterChef)	Acknowledged

Description

The `updatePool()` function over mint the reward in the contract `MasterChef`.

- devaddr address mint the butterReward * 10%
- address(cream) mint the butterReward(100%)

So total butterReward minted is 100% + 10% = 110%.

Recommendation

Fix to mint 100% of the block reward instead of 100% + 10%.

Alleviation

The development team replied that, they didn't invent this. This is intentionally designed for the good of eco-system and investor, other famous DEX such as Pancake also has this mechanism.

MCC-08 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	Minor	butter-farm/contracts/MasterChef.sol: 228(MasterChef), 250(MasterChef)	Acknowledged

Description

The MasterChef contract operates as the main entry for interaction with staking users. The staking users deposit LP tokens into the Butter pool and, in return, get a 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 Butter 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 `MasterChef` contract and, if there is a need to support deflationary tokens, add necessary mitigation mechanisms to keep track of accurate balances.

Alleviation

The development team replied that, HT(Huobi Token) is used as transaction fee, and LP token will not be changed as fee during deposit and withdraw.

MCC-09 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	butter-farm/contracts/MasterChef.sol: 97(MasterChef), 330(MasterChef)	Resolved

Description

Missing validation for the input variables `_devaddr` in function `MasterChef.constructor()` and `MasterChef.dev()`.

Recommendation

Consider adding below checks to ensure these input variables are not equal to `address(0)`:

```
function dev(address _devaddr) public {
    require(msg.sender == devaddr, "dev: wut?");
    require(_devaddr != address(0), "dev:  _devaddr is zero address");
    devaddr = _devaddr;
}
```

Alleviation

The team heeded our advice and removed the function in commit d2fd6d290bd5a489ed472be893d54d7382205089.

SCC-01 | `Checks-effects-interactions` Pattern Not Used

Category	Severity	Location	Status
Logical Issue	Major	butter-farm/contracts/SousChef.sol: 125(SousChef), 148(SousChef)	Resolved

Description

During `deposit()`, `withdraw()`, `enterStaking()` and `leaveStaking()` functions call, state variables for balance are changed after transfers are done. This might lead to reentrancy issue. The order of external call/transfer and storage manipulation must follow checks-effects-interactions pattern.

Recommendation

It is recommended to follow checks-effects-interactions pattern for cases like this. It shields public functions from re-entrancy attacks. It's always a good practice to follow this pattern. checks-effects-interactions pattern also applies to ERC20 tokens as they can inform the recipient of a transfer in certain implementations.

Reference: https://docs.soliditylang.org/en/develop/security-considerations.html?highlight=check-effects%23use-the-checks-effects-interactions-pattern

Alleviation

The team heeded our advice and removed the function in commit d2fd6d290bd5a489ed472be893d54d7382205089.

SCC-02 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	Informational	butter-farm/contracts/SousChef.sol: 79(SousChef)	Acknowledged

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to customers.

- `setMigrator()`
- `emergencyWithdraw()`
- `safeButterTransfer()`
- `dev()`
- `setFeeTo()`
- `setFeeToSetter()`
- `stopReward()`

Recommendation

Consider adding events for sensitive actions, and emit them 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);
    }
```

SCC-03 | Redundant Variable

Category	Severity	Location	Status
Gas Optimization	Informational	butter-farm/contracts/SousChef.sol: 51(SousChef)	Acknowledged

Description

The variable `startBlock` is only initialized and never used.

Recommendation

Consider to remove this variable, and directly initialize `poolInfo.lastRewardBlock` with `_startBlock`.

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.

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.