Security Assessment

Biswap

Jun 9th, 2021

Table of Contents

Summary

This report has been prepared for Biswap 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	Biswap
Description	MasterChef+BiSwap
Platform	BSC
Language	Solidity
Codebase	https://github.com/biswap-org
Commit	4f26647e80c684062bbb936b9abc3755ea293558

Audit Summary

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

Vulnerability Summary

Total Issues	10
Critical	0
Major	0
Medium	3
Minor	4
Informational	3
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
BER	core/BiswapERC20.sol	a7d1ee2b0c1b2fe3539d8880a703fcdf8b07427d176e793f532ae5e400192834
BFB	core/BiswapFactory.sol	904f2b14bbe2c2768862f044c470f313331a418b8d893293a5a63a7ec4eae1c5
BPB	core/BiswapPair.sol	d9fd8338543193269aeb0ab1676a88f9869ae83ec2573ad53f2f11992f6ff94d
BMB	periphery/BiswapMigrator.sol	f707eb04289c5e81e198a8ff703bfda334deab076badf24728bdc84ac6de57be
BRB	periphery/BiswapRouter01.sol	4d37915ef892529b821b596b4e540493c16e010da840c54e511805a1ecf1aa59
BRS	periphery/BiswapRouter02.sol	37435b58d759ca6af54cf40afd12eb0737761f87e08e7f503424b2095fdf6812
BSW	staking/BSWToken.sol	77eaeb5998d3216155d4aad3c49df171851406d1195f90ea467efcb99eb952c2
MCB	staking/MasterChef.sol	7286fd727fd496c1abba25a8ab5dbf0378489bc0158320ee0752dc9c9c5a5139

Findings

Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**3** (30.00%)
Minor	**4** (40.00%)
Informational	**3** (30.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
BFB-01	Unmatch function parameter	Logical Issue	Informational	Acknowledged
BRS-01	Third-party dependencies	Logical Issue	Minor	Acknowledged
BSW-01	Role of onlyMinter() is not set correctly	Logical Issue	Minor	Acknowledged
BSW-02	Privileged ownerships on BSWToken	**Centralization / Privilege**	**Minor**	**Acknowledged**
BSW-03	Delegation Not Moved Along With `transfer()`	Logical Issue	Medium	Acknowledged
MCB-01	`add()` function not restricted	Volatile Code	Medium	Acknowledged
MCB-02	Missing emit events	Gas Optimization	Informational	Acknowledged
MCB-03	Recommended explicit pool validity checks	Logical Issue	Informational	Acknowledged
MCB-04	Proper Usage on migrator.migrate()	**Logical Issue, Centralization / Privilege**	**Medium**	**Acknowledged**
MCB-05	Privileged ownerships on MasterChef	**Centralization / Privilege**	**Minor**	**Acknowledged**

BFB-01 | Unmatch function parameter

Category	Severity	Location	Status
Logical Issue	Informational	core/BiswapFactory.sol: 54	Acknowledged

Description

_devFee was defined as "uint32" in function setDevFee(uint32 _devFee), while "uint8" in BiswapFactory.sol: function setDevFee(address _pair, uint8 _devFee) external

Recommendation

We recommend always using the same type definition.

BRS-01 | Third-party dependencies

Category	Severity	Location	Status
Logical Issue	Minor	periphery/BiswapRouter02.sol: 272, 391	Acknowledged

Description

Third party function ISwapFeeReward(swapFeeReward).swap(msg.sender, input, output, amountOut) was called in _swap(). The contract is serving as the underlying entity to interact with third party functions. 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 encourage the team to constantly monitor the status of those 3rd parties to mitigate the side effects when unexpected activities are observed.

BSW-01 | Role of onlyMinter() is not set correctly

Category	Severity	Location	Status
Logical Issue	Minor	staking/BSWToken.sol: 1284	Acknowledged

Description

Role of onlyMinter() is not set correctly, no initial Minter was set.

Recommendation

We recommend to set an initial minter(MasterChef).

BSW-02 | Privileged ownerships on BSWToken

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	staking/BSWToken.sol	**Acknowledged**

Description

BSWToken is the standard BEP20 implementation that contains the mint functionality with ownership controls, which means whoever obtained access to the minter account would be able to tamper with the integrity of the token economics.

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. Specifically for this scenario, we assume the owner will be transferred to the vault (MasterChef) on top of the token. We recommend that the team maintains a high level of transparency on such a transaction taking place.

BSW-03 | Delegation Not Moved Along With `transfer()`

Category	Severity	Location	Status
Logical Issue	Medium	staking/BSWToken.sol: 1	Acknowledged

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

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

MCB-01 | `add()` function not restricted

Category	Severity	Location	Status
Volatile Code	Medium	staking/MasterChef.sol: 227	Acknowledged

Description

The total amount of reward BSWReward 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 is not reflected in the comment behaviors as there isn't any valid restriction on preventing this issue.

The current implementation relies on the owner's trust to avoid repeatedly adding the 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.

MCB-02 | Missing emit events

Category	Severity	Location	Status
Gas Optimization	Informational	staking/MasterChef.sol: 207~210, 253, 424	Acknowledged

Description

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

- `dev()`
- `setMigrator()`
- `updateBswPerBlock()`

Recommendation

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

MCB-03 | Recommended explicit pool validity checks

Category	Severity	Location	Status
Logical Issue	Informational	staking/MasterChef.sol: 244	Acknowledged

Description

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

Recommendation

We advise the client to adopt following modifier `validatePoolByPid` to functions `set()`, `migrate()`, `deposit()`, `withdraw()`, `emergencyWithdraw()`, `pendingBSW()` and `updatePool()`.

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

MCB-04 | Proper Usage on migrator.migrate()

Category	Severity	Location	Status
Logical Issue, Centralization / Privilege	**Medium**	staking/MasterChef.sol: 254, 264	**Acknowledged**

Description

`setMigrator()` function can set migrator contract to any contract that is implemented from `IMigratorChef` interface by the owner. As result, invocation of `migrator.migrate()` in function migrate() may bring dangerous effects as it is unknown to the user. However, the project may lose the ability to upgrade and migrate if `setMigrator()` and `migrate()` are removed.

Recommendation

We recommend to either removing such functionality on migrate; or having a timelock as the controlling party with enough delay for such privileged function invocations. The project shall also make clear statement or documents on how they plan to leverage the migrate functionality for proper usage.

MCB-05 | Privileged ownerships on MasterChef

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	staking/MasterChef.sol	**Acknowledged**

Description

The owner of MasterChef has permission to add and set pools that could update the parameters on rewards 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.

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.

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.