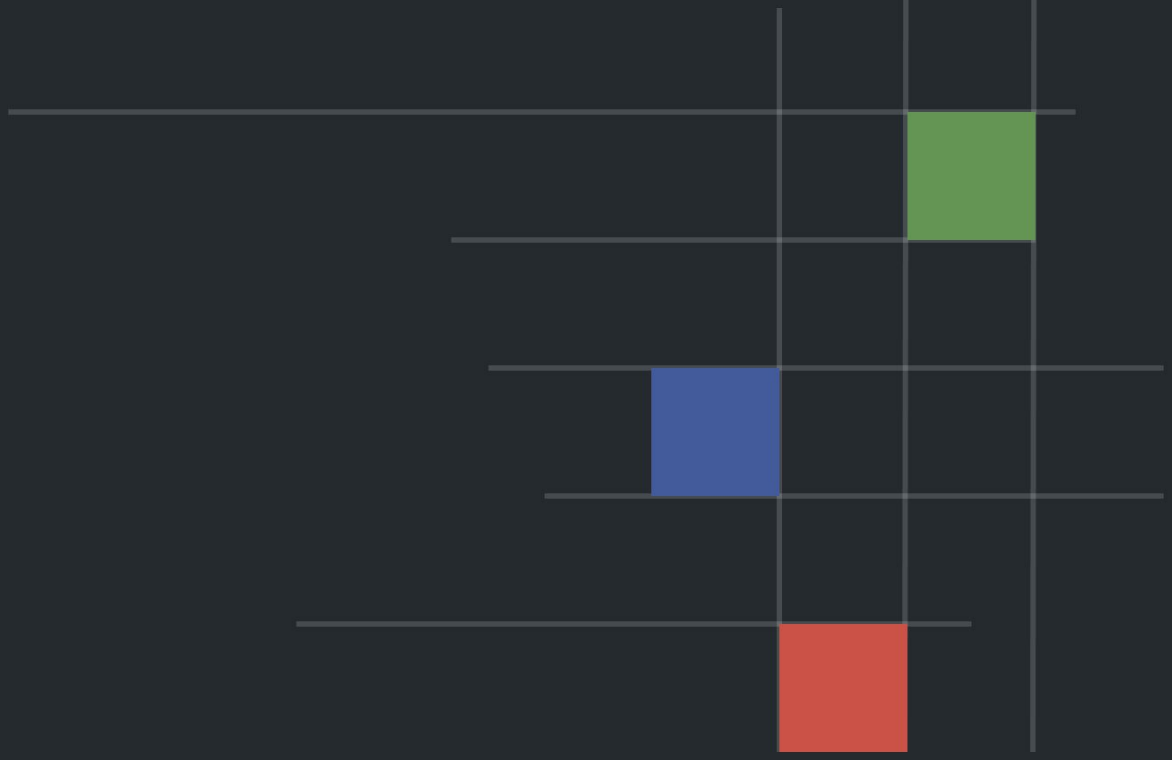
ETH-BSC-Swap

Security Assessment

Mar 30th 2021



Summary

This report has been prepared for the Binance Smart Chain team's smart contracts on the ETH-BSC-Swap, to discover issues and vulnerabilities in the source code as well as any 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 security experts.

The security assessment resulted in 7 findings that are categorized as informational. We recommend to address these findings as potential improvements that can benefit the long run as both smart contracts would have state change effects on ERC20 based token assets. Overall the source code is well written and the logic is straightforward, yet given the fact there is no fully decentralized interchain communication channel between ETH and BSC, the liveness of the bridge functionality greatly relies on centralized monitoring jobs (not in the scope of the assessment) and we suggest BSC team to disclose such information to public and ensure a level of site reliability engineering efforts.

Overview

Project Summary

Name	ETH-BSC-Swap
Codebase	release: audit1.0

Engagement Summary

Delivery Date	March 30th, 2021
Methodology	Static analysis, manual review and simulation
Contracts in Scope	4
Contract - BSC-Impl	release: audit1.0 | BSCSwapAgentImpl
Contract - BSC-Proxy	release: audit1.0 | BSCSwapAgentUpgrdableProxy
Contract - ETH-Impl	release: audit1.0 | ETHSwapAgentImpl
Contract - ETH-Proxy	release: audit1.0 | ETHSwapAgentUpgrdableProxy

Finding Summary

Total	7
Critical	0
Medium	0
Minor	0
Informational	7

Understanding of Core Logics

BSCSwapAgentImpl

On Binance Smart Chain, an instance of `BSCSwapAgentImpl` would be deployed (or via a proxy wrapper) to handle three major functionalities:

1. `createSwapPair()`: Privileged call by owner only and invoked via the monitoring job. Create a swap pair whenever a erc20 token is registered on ETH side. The bep20 token mapping the erc20 is not a mirror since it's initialized with 0 totalSupply and a proxy wrapper;
2. `fillETH2BSCSwap()`: Privileged call by owner only and invoked via the monitoring job. Mint the corresponding amount of bep20 tokens to the swap caller on the ETH side. It ensures that the same txhash on ETH won't be executed twice with the help of the state mapping of `filledETHTx`;
3. `swapBSC2ETH()`: payable call by public of those who wish to swap back bep20 tokens and get erc20 format on the ETH side. A `swapFee` is required and forwarded to the owner for the sake of reducing malicious spams that may drain out the operational funds of the monitoring jobs quicker than expected. A successful swap would result in a burn of the bep20 token.

BSCSwapAgentUpgradableProxy

This is a typical implementation of a proxy wrapper based on OpenZeppelin's standard `TransparentUpgradeableProxy` library. Based on the unit tests and migration files provided, we did not see that there is any proxy on top of the agent contracts, nevertheless such pattern would not introduce security concerns wherera correct migration steps are well in-place.

ETHSwapAgentImpl

On Ethereum, an instance of `ETHSwapAgentImpl` would be deployed (or via a proxy wrapper) to handle three major functionalities:

1. `registerSwapPairToBSC()`: Public callable function and any user could invoke to register a erc20 token. Once succeeded, such an event would be subscribed by the monitoring job and make corresponding action on the BSC side;
2. `fillBSC2ETHSwap()`: Privileged call by owner only and invoked via the monitoring job. Transfer the amount (validated off-chain) of erc20 tokens from the agent

contract to the user. With the `filledBSCTx`, no same txhash on BSC would be executed twice causing the `double-send`;

3. `swapETH2BSC()`: payable call by public of those who wish to swap erc20 tokens and get bep20 format on the BSC side. A `swapFee` is required and forwarded to the owner for the sake of reducing malicious spams that may drain out the operational funds of the monitoring jobs quicker than expected. A successful swap would result in a `transferFrom` of the user to the agent contract.

ETHSwapAgentUpgradableProxy

Same understanding referring to `BSCSwapAgentUpgradableProxy`.

Findings

ID	Title	Severity	Response
CTK-BSC-1	Reentrancy on BSCSwapAgentImpl	Informational	Acknowledged
CTK-BSC-2	Handling of Return Values	Informational	Acknowledged
CTK-BSC-3	Multiple Versions of Solidity	Informational	Resolved
CTK-BSC-4	Stricter Function Visibilities	Informational	Acknowledged
CTK-BSC-5	Potential Denial of Service	Informational	Acknowledged
CTK-BSC-6	Centralization Dependency	Informational	Acknowledged
CTK-BSC-7	SPDX License Identifiers	Informational	Acknowledged

CTK-BSC-1 | Reentrancy on BSCSwapAgentImpl

Type	Severity	Location
Volatile Code	Info	BSC-Impl: L93-L101

Description

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

```
    function createSwapPair(...) onlyOwner external returns (address) {
        ...
        BEP20UpgradeableProxy proxyToken = new
BEP20UpgradeableProxy(bep20Implementation, bep20ProxyAdmin, "");
        IProxyInitialize token = IProxyInitialize(address(proxyToken));
        token.initialize(name, symbol, decimals, 0, true, address(this));

        swapMappingETH2BSC[erc20Addr] = address(token);
        swapMappingBSC2ETH[address(token)] = erc20Addr;
        ...
    }
```

Recommendation

Given the fact that token is a brand new instance initiated via proxy template, it would not introduce malicious external calls. Yet still it is considered as a good practice to follow the pattern and move `token.inialize()` after the below state changes.

The best practice to avoid Reentrancy weaknesses is to make sure all internal state changes are performed before the call is executed. This is known as the Check-Effects-Interaction Pattern.

Alleviation

The team confirmed that there is no reentrancy attack here, and only get token addr only after the external call.

CTK-BSC-2 | Handling of Return Values

Type	Severity	Location
Volatile Code	Info	BSC-Impl: L115, L129

Description

`mintTo()` and `burn()` are inherited from `BEP20TokenImplementation` and they have return values that are supposed to check when interacting with.

```
    function fillETH2BSCSwap(...) onlyOwner external returns (bool) {
        ...
        ISwap(bscTokenAddr).mintTo(amount, toAddress);
        ...
        return true;
    }

    function swapBSC2ETH(...) payable external notContract returns (bool) {
        ...
        ISwap(bep20Addr).burn(amount);
        ...
        return true;
    }
```

Recommendation

Consider adding the return value checks.

Alleviation

The team confirmed that there is no need given that It is not possible that the BEP20 contract will return false.

CTK-BSC-3 | Multiple Versions of Solidity

Type	Severity	Location
Inconsistency	Info	Multiple files

Description

Files on `BSCSwapAgentImpl` and `ETHSwapAgentImpl` are using version of `0.6.4`, while other files are not anchored to the specific version.

Recommendation

Consider aligning the versions or give a range of 0.6.x for compatibility.

Alleviation

The team will use a fixed 0.6.4 version.

CTK-BSC-4 | Stricter Function Visibilities

Type	Severity	Location
Language Specific	Info	BSC-Impl: L48, L68, L77

Description

Three functions's visibilities are wider than their usage:

- `initialize()`
- `renounceOwnership()`
- `transferOwnership()`

Recommendation

When functions are intended to be invoked via contracts or transactions, consider using external instead of public.

Alleviation

The team confirmed that it is referenced from public libraries thus remain as it is.

CTK-BSC-5 | Potential Denial of Service

Type	Severity	Location
Business Model	Info	ETH-Impl: L79

Description

The visibility of registerSwapPairToBSC() is external, which means anyone can invoke this function to register a ERC20 token address. Given the ETH-BSC swap heavily relies on the centralized monitoring job that subscribes to the events emitted from such function, it is possible that malicious users may spam the swap system by keeping posting transactions with random or newly created ERC20 contracts. In result, the monitoring job would be introduced with more overhead and corresponding invocations on the BSC side.

Recommendation

Given the current high gas price on ETH, we do not think the chance for such scenario would be high (meanwhile no financial incentives) enough to consider a code change. Some level of precautions could be applied at monitoring jobs to mitigate the happening, i.e. identifying addresses with repeated function calls or setup a queue to batch handle the invocations on BSC side.

On the other hand, it's also practical to build a BSC-ETH swap by deploying the ETH-Impl on BSC and BSC-Impl on ETH. If that is the case, more cautions should be considered give the gas on BSC is much affordable.

Alleviation

The team confirmed that it is by design that this system is open for anyone to register as a swap pair.

CTK-BSC-6 | Centralization Dependency

Type	Severity	Location
Business Model	Info	N/A

Description

Given the incompatibility of inter blockchain communication between BSC and ETH, the swap would be maintained and run via a mix of smart contracts and off-chain monitoring processes. The integrity of asset transfers is not guaranteed on-chain solely.

Recommendation

There is no 100% decentralized way of solving, and we consider the current approach to be aligned with the standards and other similar functional swaps or bridges. More efforts would be applied to the availability and reliability of the monitoring processes running off the chain.

CTK-BSC-7 | SPDX License Identifiers

Type	Severity	Location
Language Specific	Info	N/A

Description

Source code files do not contain the one-liner info on the license of the contracts.

Recommendation

Since 0.6.8, Solidity introduced SPDX license identifiers and developers could specify the types to address their contract usages. It is recommended to have such specification on the Solidity files that intend to be open-source.

```
// SPDX-License-Identifier: MIT
```

Appendix | Finding Categories

Gas Optimization

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

Refer to exhibits that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

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

Control Flow

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

Volatile Code

Refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

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

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

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

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

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.

Business Model

Refer to contract or function logics that are debatable or not clearly implemented according to the design intentions.

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 CertiK

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.

CERTIK

Provable Trust For All