CERTIK

Security Assessment

ConvX

Apr 12th, 2021

Summary

This report has been prepared for ConvX 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 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.

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	ConvX
Description	The decentralized interchangeable asset protocol.
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/CertiKProject/certik-audit-contracts/tree/convergenceX/contracts/convergenceX/contracts
Commits	91f982a82ff754c7b23cd32fef224c0b623d3124

Audit Summary

Delivery Date	Apr 12, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

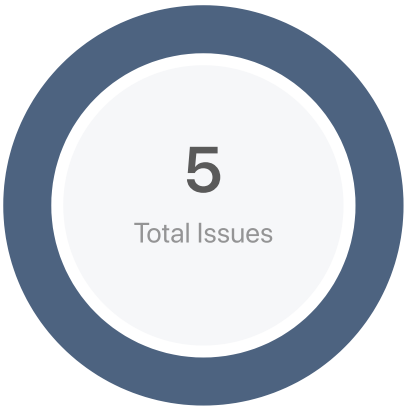
Total Issues	5
Critical	0
Major	0
Minor	0
Informational	5
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
IER	interfaces/IERC20.sol	601f298284e7c6489173ac75eb82eb852b3198e50e44ad4b331743160ad02c5b
ISC	interfaces/ISwapCallee.sol	3731c6ae122ade29120e5bbfa938208e086674c8b3efc480e4cf17448fb8061f
ISF	interfaces/ISwapFactory.sol	3e079d4cd1ab08539552f168621cefb3058454f43087d77845274912237d58c6
ISP	interfaces/ISwapPair.sol	cd4e51b96dfa722916d134a7e6389644bc357274a778c272d9b236d3d6362f64
ISR	interfaces/ISwapRouter.sol	3a8520256b776e8a8e8540680a3f579a1a69b688a22058a0aaca0dec2ad910b3
IWE	interfaces/IWETH.sol	cec1aaf363cc97ae523254961e0bb2b246686e1a632e0dff5285d322cdd7dca1
MXC	libraries/Math.sol	a553dd23aa798c18e1b2a19b2f64a2ba8144df56e212f20bab346be5c37287bb
SMX	libraries/SafeMath.sol	a3ea60a0335eada3ce9d3e6066ac9aed03271d2efc76f6f2d188d658472734c7
SSM	libraries/SignedSafeMath.sol	6047c5f20b7a310796ba510ff8c3e8c155f737828283470d2646d1e5cacd01b8
SLX	libraries/SwapLibrary.sol	2dd4667ee3be4927a612757e38de0496713093a5934874e545a5679a8224b677
THX	libraries/TransferHelper.sol	00e6bcd0ba6343d9dc40fe63f736f3ac2a51203becafa3d39af33c6f7bb1107c
UQX	libraries/UQ112x112.sol	24283a562d299a5e4133d3f05304eec8e75a1b18c6907dd2a8f399eea0b16524
SER	swap/SwapERC20.sol	490280a00135108370263e3de19f8974b487c30d1342714e4462f0768b094a58
SFX	swap/SwapFactory.sol	b8d51bd187052412efde684de3eccaf5ec56a218815cf9071c75fffd52566f11

ID	file	SHA256 Checksum
SPX	swap/SwapPair.sol	782ee06c0b2cfae6c14c63b0a046acc53923516103d238f734cc72513e25cbb7
SRX	swap/SwapRouter.sol	97ba11b9bbf32c3dacd27f380c21362afecaabc3e6fd5a9776f63a970f36cb81

Findings



Critical	0	(0.00%)
Major	0	(0.00%)
Minor	0	(0.00%)
Informational	5	(100.00%)
Discussion	0	(0.00%)

ID	Title	Category	Severity	Status
SFX-1	Lack of Input Validation	Volatile Code	Informational	Acknowledged
SFX-2	Missing Emit Events	Volatile Code	Informational	Acknowledged
SLX-1	Different Solidity Version	Volatile Code	Informational	Resolved
SPX-1	Unknown Implementation of `migrator`	Logical Issue	Informational	Acknowledged
SRX-1	Lack of Input Validation	Volatile Code	Informational	Acknowledged

SFX-1 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	swap/SwapFactory.sol: 19	Acknowledged

Description

The assigned value to `_feeToSetter` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in constructor of contract `SwapFactory.sol`. Violation of this may cause losing ownership of `feeToSetter` authorization.

Recommendation

Check that the address is not zero by adding following checks in the constructor of contract `SwapFactory.sol`.

```
require(_feeToSetter != address(0), "_feeToSetter is zero address");
```

Alleviation

The development team responded that they will control the address to be non zero in their deployment scripts.

SFX-2 | Missing Emit Events

Category	Severity	Location	Status
Volatile Code	Informational	swap/SwapFactory.sol: 47~60	Acknowledged

Description

Several key actions are defined without event declarations. Example:

```
47 setFeeTo(address _feeTo)
```

```
52 setMigrator(address _migrator)
```

```
57 setFeeToSetter(address _feeToSetter)
```

Recommendation

Consider emitting events for key actions.

Alleviation

The development team responded that no use case for that to emit events

SLX-1 | Different Solidity Version

Category	Severity	Location	Status
Volatile Code	Informational	libraries/SwapLibrary.sol: 3	Resolved

Description

There are 2 different solidity versions in the source codes, i.e. `pragma solidity >=0.5.0;` and `pragma solidity 0.6.12;`

Recommendation

Consider keeping source codes the same solidity version.

Alleviation

The development team responded that they support multiple solidity versions via their hardhat environment config.

SPX-1 | Unknown Implementation of `migrator`

Category	Severity	Location	Status
Logical Issue	Informational	swap/SwapPair.sol: 127~128	Acknowledged

Description

The implementation of `migrator` is unknown, but its function `desiredLiquidity()` affects on the execution of function `mint()`.

Recommendation

Consider exposing the implementation of `migrator`.

Alleviation

The development team responded that it is gonna be implemented in a separate contract which design is similar to sushiswap.

SRX-1 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	swap/SwapRouter.sol: 24~25	Acknowledged

Description

The assigned value to `_factory` , `_WETH` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in constructor of contract `SwapRouter.sol` . Violation of this may cause losing ownership of `factory` , `WETH` authorization.

Recommendation

Check that the address is not zero by adding following checks in the constructor of contract `SwapRouter.sol` .

```
require(_factory != address(0), "_factory is zero address");
require(_WETH != address(0), "_WETH is zero address");
```

Alleviation

The development team responded that they control the creation of this contract in their deployment script so they will have safeguard there.

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 in storage 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.

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.