CERTIK

Security Assessment

ConvO

Jun 22nd, 2021

Table of Contents

Summary

This report has been prepared for ConvO 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	ConvO
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/Convergence-Finance/convergenceO/tree/master/contracts
Commit	5e8312f71d6b2be33d3cffc46b1587c2ab2e8259 937bbda226204db6fead1a76377c8c5f54d6504b

Audit Summary

Delivery Date	Jun 22, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	9
● Critical	0
● Major	0
● Medium	0
● Minor	3
● Informational	6
● Discussion	0

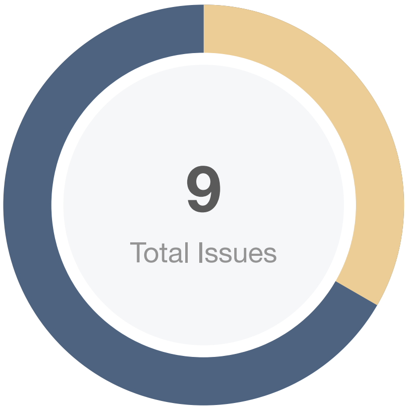
Audit Scope

ID	file	SHA256 Checksum
FSC	FixedSwap.sol	509023e623ba9e3a06fe7892dea60b72a421f44aa5dbc908f33cd49c7cfd0a64

It should be noted that the system design includes a number of economic arguments and assumptions. These were explored to the extent that they clarified the intention of the code base, but we did not audit the mechanism design itself.

Additionally, financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol. The correctness of the financial model is not in the scope of the audit.

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**0** (0.00%)
🟨 **Medium**	**0** (0.00%)
🟨 **Minor**	**3** (33.33%)
🟦 **Informational**	**6** (66.67%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
FSC-01	Unlocked Compiler Version	Language Specific	● Informational	ⓘ Acknowledged
FSC-02	Proper Usage of `public` and `external`	Gas Optimization	● Informational	ⓘ Acknowledged
FSC-03	Lack of Error Message	Coding Style	● Informational	ⓘ Acknowledged
FSC-04	Set `immutable` to Variables	Gas Optimization	● Informational	ⓘ Acknowledged
FSC-05	Redundant Code	Logical Issue	● Informational	ⓘ Acknowledged
FSC-06	Check Effect Interaction Pattern Violated	Logical Issue	● Minor	✓ Resolved
FSC-07	Owner Capacity	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
FSC-08	Multiple Storage Reads	Gas Optimization	● Informational	ⓘ Acknowledged
FSC-09	Logic Issue of `fund()`	Logical Issue	● Minor	ⓘ Acknowledged

FSC-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational	FixedSwap.sol: 3	ⓘ Acknowledged

Description

The contract has unlocked compiler versions. An unlocked compiler version in the contract's source code permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging as compiler-specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation

No alleviation.

FSC-02 | Proper Usage of `public` and `external`

Category	Severity	Location	Status
Gas Optimization	● Informational	FixedSwap.sol: 157, 227, 231, 240, 203	ⓘ Acknowledged

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to use the `external` attribute for functions never called within the contract.

Alleviation

No alleviation.

FSC-03 | Lack of Error Message

Category	Severity	Location	Status
Coding Style	● Informational	FixedSwap.sol: 345	ⓘ Acknowledged

Description

The convenience function `require` can be used to check for conditions and throw an exception if the condition is not met. If you do not provide a string argument to require, it will revert with empty error data, not even including the error selector. (LINK)

Recommendation

We advise the client to add error messages.

Alleviation

No alleviation.

FSC-04 | Set `immutable` to Variables

Category	Severity	Location	Status
Gas Optimization	● Informational	FixedSwap.sol: 25, 26, 27, 33, 34, 21	ⓘ Acknowledged

Description

The variables `erc20`,`tradeValue`, `startDate`,`endDate`,`isTokenSwapAtomic` and `feeAddress` are only changed once in the `constructor` function.

Recommendation

We advise the client to set `erc20`,`tradeValue`, `startDate`,`endDate`,`isTokenSwapAtomic` and `feeAddress` as `immutable` variables.

Alleviation

No alleviation.

FSC-05 | Redundant Code

Category	Severity	Location	Status
Logical Issue	● Informational	FixedSwap.sol: 79	ⓘ Acknowledged

Description

If the conditions that `block.timestamp < _startDate` and `_startDate < _endDate` are satisfied ,then `block.timestamp < _endDate` would be true. Therefore there is no need to validate this condition.

Recommendation

The following code can be removed:

```
79     require(block.timestamp < _endDate, "End Date should be further than current date");
```

Alleviation

No alleviation.

FSC-06 | Check Effect Interaction Pattern Violated

Category	Severity	Location	Status
Logical Issue	● Minor	FixedSwap.sol: 240	✓ Resolved

Description

The sequence of external call/transfer and storage manipulation must follow a check effect interaction pattern. For example:

- `swap()`

Recommendation

We advise the client to adopt `nonReentrant` modifier from `openzeppelin` library to the function `swap()` to prevent any reentrancy issue.(LINK)

Alleviation

The team heeded our advice and resolved this issue in commit : 937bbda226204db6fead1a76377c8c5f54d6504b.

FSC-07 | Owner Capacity

Category	Severity	Location	Status
Centralization / Privilege	● **Minor**	FixedSwap.sol	ⓘ **Acknowledged**

Description

The owner of contract `Convergence0` has the privilege to:

- `pause()`
- `safePull()`

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.

Alleviation

The team responds that they will need this admin ownership and will follow the advice to renounce it at the right timing.

FSC-08 | Multiple Storage Reads

Category	Severity	Location	Status
Gas Optimization	● Informational	FixedSwap.sol: 316	ⓘ Acknowledged

Description

In functions `purchases[purchase_id]` is repeatedly read from storage, which is very gas inefficient.

Recommendation

We recommend assigning the values to memory variables first before using, as a call from storage costs 200 gas and a call from memory costs only 3 gas.

Alleviation

No alleviation.

FSC-09 | Logic Issue of `fund()`

Category	Severity	Location	Status
Logical Issue	● Minor	FixedSwap.sol: 240	ⓘ Acknowledged

Description

Currently, in order for a sale to be successfully funded, the contract token balance needs to be checked within the `fund()` function. However, one can also increase the balance of the contract by `transfer()` instead of `fund()`. This may result in failure to proceed through `fund()` function beyond L242 if the contract balance is already larger than the proposed `tokenForSale` because the requirement that "transferred tokens is no more than needed" is not met. As a consequence, the `swap` will not be able to execute.

Recommendation

We advise the client to define a variable to replace `availableTokens()`.

Alleviation

The team responds that the design for IDO is that all the token would be initially held by them only, so it is impossible for an external user to transfer that erc20 token to this contract.

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.

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 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.

CERTIK