Security Assessment

Conjure Finance

Jun 10th, 2021

Table of Contents

Summary

This report has been prepared for Conjure Finance 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	Conjure Finance
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/ConjureFi/contracts-main/tree/main/contracts
Commits	29fca5fcb4cf75cdf620965908e6c49cc6820139

Audit Summary

Delivery Date	Jun 10, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	14
Critical	0
Major	6
Medium	0
Minor	1
Informational	7
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
CCF	Conjure.sol	0e1a5d9800b67832039c279be777c34b5a7c6ed0ea05886740c3b7acd49c84f4
CFC	ConjureFactory.sol	11bd27229ef3c586e5955f08f55563c5ff793716eb5d1bde84472c84ae6cb313
CRC	ConjureRouter.sol	e5b46c9cd2cd749e949601b171afc4764491153af79bcc0c2341982bf543f002
ECC	EtherCollateral.sol	a4cad4870610987ca607f5353763105e771cab4715833f8997d64a0083920cd8
OCF	Owned.sol	66e5b42ed8619ab0c7bf9d9656b454ab63c904618bde995c07566aea3f0f90a5
PCF	Pausable.sol	f4a6c7948face94dde5facbdef3ba920a911e51d718f7e28922229612ebd0cd9
RDR	RewardsDistributionRecipient.sol	90fa2f7541f01188e269276137366a398cbc1ca4a2331fe390ee637c1b29799c
SDM	SafeDecimalMath.sol	d0786bcacd76c44b842813fdb60e07aa0e726bcc930f3e112ec82b61b03e6f1e
SRC	StakingRewards.sol	ce37a20bee9a4a60a1c5b3223ed2b1f6989f6384c217da8bb16fb4710a3a1547
ICC	interfaces/IConjure.sol	e2acc1a968e430a15730324ac6e31bec75b5111bf9d9875e1b398c87260052af
ICF	interfaces/IConjureFactory.sol	19d172edf55627cc34e283e89c39a245a3f289054b59ee630971ca9864b2c361
ICR	interfaces/IConjureRouter.sol	b9be246cdfea785cc7e89169c2a0d3af7e84a7300efe34dbe8d6ab10ae007a56
IER	interfaces/IERC20.sol	e01e8b39e1854e07bad5f90094e29369839e683ffe24a6499dcc6192c2b8a72b
IEC	interfaces/IEtherCollateral.sol	bb2b6e97fd33a6de291f65421db1097cbe5b4f7d03ffe0f6798142b6f9d6b841
ISR	interfaces/IStakingRewards.sol	a144c04943a4bd85aa51e290fdda223f625e6e58a5be273cc2f41bb7064a5b67
CLC	lib/CloneLibrary.sol	38bf8f5526dd6658d922e8a2d6e60c73098a4b9ede0e8e43ed59cbe343de2799
FPC	lib/FixedPoint.sol	ab686bbce375be1b8eee6898fb1e388ead4e2de229837bc9cef45a2b2256430f

System Overviews

Conjure allows anyone to create tokenized assets on Ethereum representing any asset you can imagine. The resulting Synths are fully collateralized by Ether and have their collateralization and value pegged via a powerfully flexible ability to combine multiple arbitrary price sources to produce a single reference price. This protocol has external dependencies. It depends on 3 types of price feeding protocols including Chainlink, Uniswap and Custom oracles. The system should only be used if the third-party services are appropriately trusted. And these external protocols are not in the scope of this audit.

Centralized Risks

The owner of contract `ConjureFactory` has the permission to:

- Admin can use functions `newConjureImplementation()` and `newEtherCollateralImplementation()` to change the implementation of `Conjure` and `EtherCollateral` with any contracts.

without obtaining the consensus of the community.

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multi-sig governing procedure and let the community monitor in respect of transparency considerations.

Additionally, according to the white-paper , the CNJ governance organization is responsible for:

- Deploying new factory implementations for more synthetic asset types and use cases
- Approving new collateral assets for synths that users can select from
- Managing the treasury for the Conjure DAO
- Managing the token list for Conjure's dapp

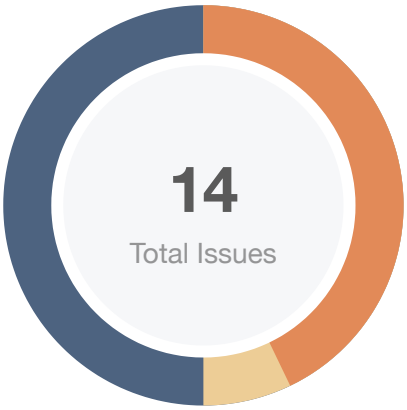
The system should only be used if the above responsibilities are fully fulfilled.

refer to: https://docs.conjure.finance/protocol/governance

Financial Models

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.

Findings



Critical	**0** (0.00%)
Major	**6** (42.86%)
Medium	**0** (0.00%)
Minor	**1** (7.14%)
Informational	**7** (50.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
ConjureFi-01	Usage of `synth`	Logical Issue	Informational	Acknowledged
ConjureFi-02	Missing Emit Events	Gas Optimization	Informational	Resolved
ConjureFi-03	Flash Loan Attack Risk	Logical Issue	Major	Acknowledged
CCF-01	Logic Issue of `_factoryContract`	Logical Issue	Major	Acknowledged
CCF-02	Boolean Equality	Gas Optimization	Informational	Resolved
CCF-03	Logic Issue of Inverse Asset	Logical Issue	Major	Acknowledged
CCF-04	Safety Usage of Custom Oracle	Logical Issue	Major	Acknowledged
CCF-05	Uniswap Oracle TWAP Price	Logical Issue	Informational	Acknowledged
CFC-01	Privileged Ownership	**Centralization / Privilege**	**Major**	**Acknowledged**
CRC-01	Address of `treasury`	Logical Issue	Informational	Acknowledged
CRC-02	Issue in Fallback Functions	Logical Issue	Major	Resolved
ECC-01	Incorrect Naming Convention Utilization	Coding Style	Informational	Resolved
ECC-02	Inconsistency Implementation of `liquidationRatio`	Logical Issue	Minor	Acknowledged
OCF-01	Unlocked Compiler Version Declaration	Language Specific	Informational	Resolved

ConjureFi-01 | Usage of `synth`

Category	Severity	Location	Status
Logical Issue	Informational	Global	Acknowledged

Description

As per the development team stating: "These assets allow Ethereum access to any financial market that exists and opens access to everyone."

Can you please describe how to use the synths more specifically, since we haven't seen the contracts for exchanging these synths.

Alleviation

The team replied: The primary use would be through using other markets like uniswap to trade the assets. There is no direct use for just making a synth if there are no markets but comes down to the creator to provide liquidity to make the assets useful. We also have our own LP rewards to create markets for select assets.

ConjureFi-02 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	Informational	Global	Resolved

Description

Several sensitive actions are defined without event declarations.

1. Fucntions `setNewOwner()`, `newTreasury()` and `newStakingrewards()` in contract `ConjureRouter` can change several important metrics of the contract.
2. Fucntions `constructor()`, `newConjureImplementation()`, `newEtherCollateralImplementation()` and `newConjureRouter()` in contract `ConjureFactory` will decide the important metrics.
3. Functions `initialize()` and `init()` in contract `Conjure` did not emit several important metrics.

Recommendation

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

Alleviation

The team heeded our advice and resolved this issue in commit 89e83b30456306f3b62cd8990cad4e39e2141479.

ConjureFi-03 | Flash Loan Attack Risk

Category	Severity	Location	Status
Logical Issue	Major	Global	Acknowledged

Description

The protocol supports 4 oracle types:

1. Chainlink
2. Uniswap TWAP
3. Custom Oracle
4. Conjure

The TWAP price is more robust for assets to prevent flash loan attack. And the mix usages of multiple oracles also help prevent flash loan attack.

But the protocol has no restrictions on the number and weight of the usage of these oracles when creating assets. e.g. select only 1 custom oracle and take 100% weight, this could lead to risks.

Recommendation

Consider to add more restrictions on the creation of conjure assets for the selection of oracles.

Alleviation

The team replied: Oracles are unrestricted by design as we don't want to create barriers for innovation, and yes there may be malicious synths, but we expect people to verify the oracles used themselves which would be why we display the oracle addresses on the dapp and for the most popular oracles to have been checked by the community, causing the best to have the highest TVL and be the most used.

CCF-01 | Logic Issue of `_factoryContract`

Category	Severity	Location	Status
Logical Issue	Major	Conjure.sol: 127, 145	Acknowledged

Description

As per the logic in the constructor, `_factoryContract` is set to `address(1)`.

But in the function `initialize()`, `_factoryContract` is required to equal to `address(0)` for the initialization. As a result, the initialization will fail.

Recommendation

Consider changing the constructor as following example:

```
constructor() {
    // Don't allow implementation to be initialized.
    _factoryContract = address(0);
}
```

Alleviation

The team replied: Just to clarify clones don't execute the constructor when you use the delegatecall proxy clone pattern. Because you are just calling the implementation for what to do but storage will be whatever on the contract copied from.

CCF-02 | Boolean Equality

Category	Severity	Location	Status
Gas Optimization	Informational	Conjure.sol: 180	Resolved

Description

Boolean constants can be used directly and do not need to be compared to true or false.

Example:

```
        require(_inited == false, "Contract already inited");
```

Recommendation

Consider changing it as following example:

```
        require(!_inited, "Contract already inited");
```

Alleviation

The team heeded our advice and resolved this issue in commit 89e83b30456306f3b62cd8990cad4e39e2141479.

CCF-03 | Logic Issue of Inverse Asset

Category	Severity	Location	Status
Logical Issue	Major	Conjure.sol: 437	Acknowledged

Description

According to the way of calculating the inverse asset price, the price could become 0 in case the underlying asset price changes dramatically. Hence this could lead to transaction failures when liquidating the loan.

Refer to : https://blog.synthetix.io/inverse-synths-are-back/

Additionally, each iSynth has three important points: an entry point, an upper limit, and a lower limit. Its entry point is the price at which it is entered into the system. The primary reason inverse assets need upper and lower limits is because the further an iSynth gets from its entry point, the more effective leverage there is for each movement due to design.

Recommendation

Consider adding upper and lower price limit on inverse asset and freezing the price when the price reaches the limit.

Alleviation

[Conjure Finance Team]:

If the price goes below zero or to zero we close the asset Conjure.sol 457. After that, no new loans can be opened and no new collateral can be deposited. The asset also won't get liquidated cause the collateral amount will always be above the liquidation ratio.

[Certik Response]:

- The primary reason inverse assets need upper and lower limits is because the further an iSynth gets from its entry point, the more effective leverage there is for each movement due to design. To use an extreme example, if iETH's entry point was at 5000 USD, and ETH got up to 9999 USD, leaving iETH at 1 USD. A user opens a loan by depositing 1 ETH and borrowing 5000 iETH now. Then, if the ETH goes higher, the user can repay iETH and close the loan. In the other case, if the ETH goes back to 9000 USD, the user can give up his collateral as the value of his iETHs is 5000000 USD now. Simply put, in this case, the user can take advantage of very high multiples of leverage without risk. Refer to: https://docs.synthetix.io/tokens/#inverse-isynths

[Conjure Finance Team]: OK acknowledged, we will consider that and will update this in the code to have boundaries also.

CCF-04 | Safety Usage of Custom Oracle

Category	Severity	Location	Status
Logical Issue	Major	Conjure.sol: 194	Acknowledged

Description

According to the white-paper: "If there is no template available for the price it is possible to send custom call data to any contract needed in order to retrieve a price. Finally, it is also possible to supply an already existing Conjure address as a price feed. The contract will then look it up in the internal system and takes the price from the arbitrary asset."

In function `updateInternalPrice`, the created Conjure asset can be based on any custom price oracles. The price retrieved from any contract can be maliciously manipulated. Additionally, the existing Conjure address based on custom price oracle is also not safe to be a price feeding.

```
500 else {
501       prices[i] = abi.decode(data, (uint));
502
503       // norming price
504       if (MAXIMUM_DECIMALS != _oracleData[i].decimals) {
505           prices[i] = prices[i] * 10 ** (MAXIMUM_DECIMALS - _oracleData[i].decimals);
506       }
507 }
```

Recommendation

Consider to support trustable custom oracle and describe the solution to prove the custom oracle is trustable.

Alleviation

[Conjure Finance Team]:

Oracles are unrestricted by design as we don't want to create barriers for innovation, and yes there may be malicious synths, but we expect people to verify the oracles used themselves which would be why we display the oracle addresses on the dapp and for the most popular oracles to have been checked by the community, causing the best to have the highest TVL and be the most used.

[CertiK response]:

We understand the team doesn't want to create barriers for innovations. However, it is difficult to verify the security of a custom oracle especially by users themselves. Consider to verify the custom oracles by the development team or the community, and then mark this new asset with different flags like "admin verified" or "community verified" to build sincere trustness.

[Conjure Finance Team]: We will build this into the ui and mark them once we whitelist them, good suggestion.

CCF-05 | Uniswap Oracle TWAP Price

Category	Severity	Location	Status
Logical Issue	Informational	Conjure.sol: 62, 496	Acknowledged

Description

According to the white-paper, the `Conjure` contract will use Uniswap TWAP price for the Uniswap oracle type.

However, the oracle address will be set in the initialization so we are unable to verify this point.

We would love to double-confirm, is the implementation reflecting as in the white-paper, they are using TWAP oracle pricing strategy?

Recommendation

Consider proving that the oracle address is really Uniswap TWAP oracle.

Alleviation

The team replied: Oracles are unrestricted by design as we don't want to create barriers for innovation, and yes there may be malicious synths, but we expect people to verify the oracles used themselves which would be why we display the oracle addresses on the dapp and for the most popular oracles to have been checked by the community, causing the best to have the highest TVL and be the most used.

CFC-01 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	**Major**	ConjureFactory.sol: 115, 127	**Acknowledged**

Description

The owner of contract `ConjureFactory` has the permission to:

- Admin can use functions `newConjureImplementation()` and `newEtherCollateralImplementation()` to change the implementation of `Conjure` and `EtherCollateral` with any contracts.

without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multi-sig governing procedure and let the community monitor in respect of transparency considerations.

Alleviation

The team replied: "The owner for the contracts will be set to a timelock."

CRC-01 | Address of `treasury`

Category	Severity	Location	Status
Logical Issue	Informational	ConjureRouter.sol: 79	Acknowledged

Description

In the function `distribute()`, half of the fees will be transferred to the address `treasury`.

However, the address `treasury` is not specified in the protocol. And the function `newTreasury()` which can change the address `treasury`, does not emit events.

Recommendation

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. The ownership of the address `treasury` is better to be handed over to DAO, and any plan to change the address `treasury` is better to move to the execution queue of the Timelock contract and emitting events as well.

Alleviation

The team replied: "The treasury address will also be the timelock address."

CRC-02 | Issue in Fallback Functions

Category	Severity	Location	Status
Logical Issue	Major	ConjureRouter.sol: 61~71	Resolved

Description

In the Ethereum, `send/transfer/call` can be used for ETH transfer. In the worst case, the fallback function can only rely on 2300 gas being available (for example when `send` or `transfer` is used), leaving little room to perform other operations except basic logging. Therefore, the callback function of the current contract is not suitable for doing much.

```
61      fallback() external payable {
62          distribute();
63      }
64
65      /**
66      * fallback function for collection funds
67      * only executes the distribution logic if the contract balance is more than 0.1
ETH
68      */
69      receive() external payable {
70          distribute();
71      }
```

refer to: https://docs.soliditylang.org/en/v0.7.0/contracts.html?highlight=2300#receive-ether-function

Recommendation

Consider to test the gas consumption of below codes:

```
61      fallback() external payable {
62          distribute();
63      }
64
65      /**
66      * fallback function for collection funds
67      * only executes the distribution logic if the contract balance is more than 0.1
ETH
68      */
69      receive() external payable {
70          distribute();
71      }
```

In case the gas consumption is over 2300, the external ether transfer (when `send` or `transfer` is used) will fail. Each opcode supported by the EVM has an associated gas cost. Pay attention the gas costs aren't arbitrary. Gas costs can and will change.

Alleviation

The team heeded our advice and removed the function in commit 89e83b30456306f3b62cd8990cad4e39e2141479.

ECC-01 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	Informational	EtherCollateral.sol: 37, 40, 46	Resolved

Description

Solidity defines a naming convention that should be followed. In general, the following naming conventions should be utilized in a Solidity file:

Constants should be named with all capital letters with underscores separating words UPPER_CASE_WITH_UNDERSCORES

Refer to https://solidity.readthedocs.io/en/v0.5.17/style-guide.html#naming-conventions

Examples:

Constants like : `minLoanCollateralSize`, `accountLoanLimit`, `liquidationPenalty`

Recommendation

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation

The team heeded our advice and resolved this issue in commit 89e83b30456306f3b62cd8990cad4e39e2141479.

ECC-02 | Inconsistency Implementation of `liquidationRatio`

Category	Severity	Location	Status
Logical Issue	Minor	EtherCollateral.sol: 161	Acknowledged

Description

According to the white-paper, the liquidationRatio is set to 120%. While in the contract `EtherCollateral`, liquidationRatio can be set from 110% to 1000%. Is this designed intentionally? Additionally, the lower the liquidationRatio is set to, the greater the risk becomes.

Recommendation

Consider to publish a twitter or medium to announce the changes on the `liquidationRatio`.

Alleviation

The team replied: "We wanted to give users the option to set whatever liquidation Ratio they want to. We will update the documentation for that and also make things clear in a tutorial."

OCF-01 | Unlocked Compiler Version Declaration

Category	Severity	Location	Status
Language Specific	Informational	Owned.sol: 2	Resolved

Description

The compiler version utilized uses the “^” prefix specifier, denoting that a compiler version which is greater than the version will be used to compile the contracts.

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

The team heeded our advice and resolved this issue in commit 89e83b30456306f3b62cd8990cad4e39e2141479.

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.