CompliFi

Automated Market Maker

Security Assessment

March 23rd, 2021

Audited By:
Sheraz Arshad @ CertiK
sheraz.arshad@certik.org
Reviewed By:
Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	CompliFi - Automated Market Maker
Description	The audited codebase comprise the Complifi's AMM Pool contract, PoolFactory and a number of contracts performing Mathematical calculations. Anyone is able to create Pool and finalize it, which can be later joined by the users by depositing primary and complement tokens and receiving pool tokens, and can be exited by the users by depositing pool tokens and getting primary and complement tokens.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 892e76d89f3de92d8d2522a674bd7a14b5a7d1db 2. 48fb4fb5a4d188eed90991d822997440261044aa

Audit Summary

Delivery Date	March 23rd, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	March 10th, 2021 - March 23rd, 2021

Vulnerability Summary

Issue Severity	Vulnerabilities Found	Vulnerabilities Resolved
🔴 Critical	0	0
🟠 Major	0	0
🟡 Medium	2	2
🔵 Minor	4	4
🟢 Informational	19	16

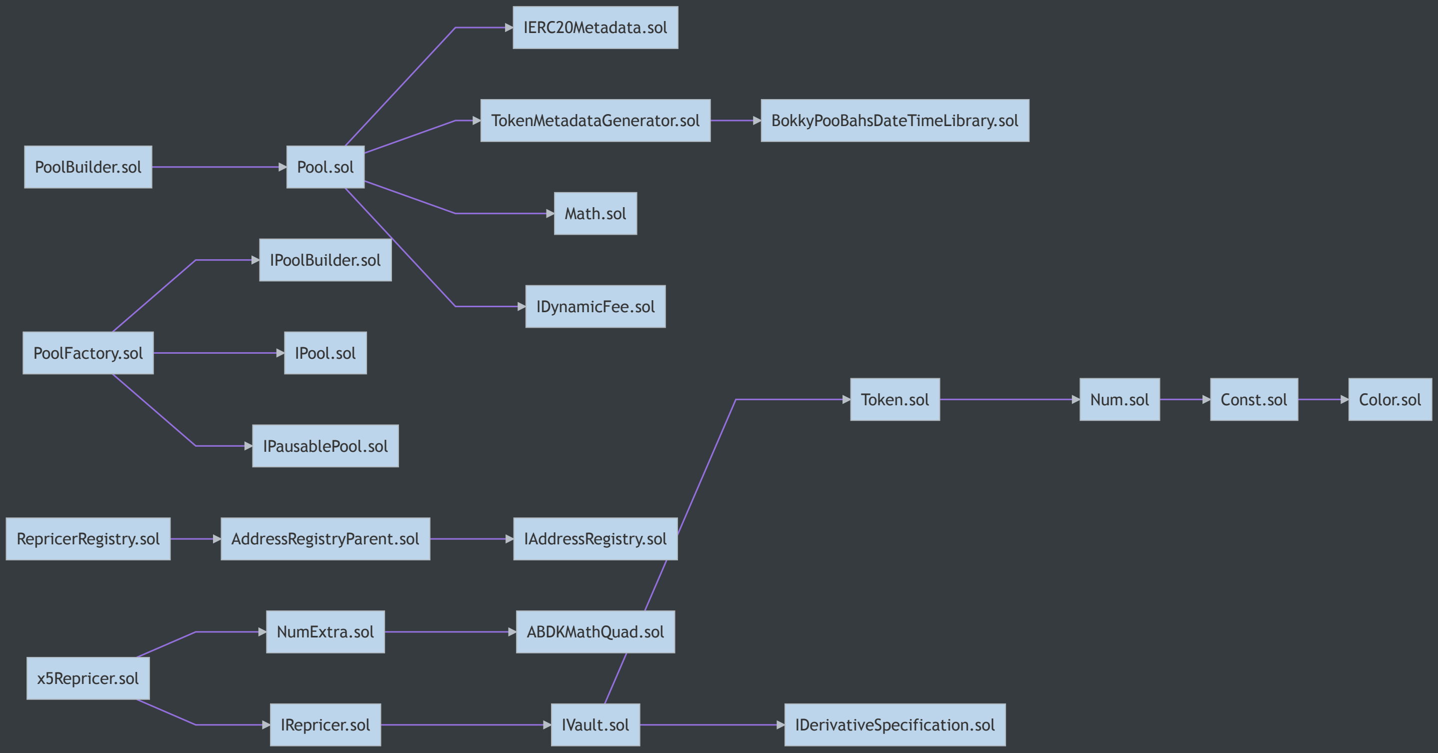
Executive Summary

This report represents the results of CertiK's engagement with CompliFi on their implementation of Automated Market Maker (AMM). The manual and static analysis were performed in the audit. Our findings mainly refer to optimizations issues, with a few minor and medium issues. Majority of the issues are remediated except a few informational issues.

Files In Scope

ID	Contract	Location
COL	Color.sol	contracts/Color.sol
CON	Const.sol	contracts/Const.sol
DFE	DynamicFee.sol	contracts/DynamicFee.sol
IDF	IDynamicFee.sol	contracts/IDynamicFee.sol
IPP	IPausablePool.sol	contracts/IPausablePool.sol
IPL	IPool.sol	contracts/IPool.sol
IPB	IPoolBuilder.sol	contracts/IPoolBuilder.sol
IVT	IVault.sol	contracts/IVault.sol
MAT	Math.sol	contracts/Math.sol
NUM	Num.sol	contracts/Num.sol
NEA	NumExtra.sol	contracts/NumExtra.sol
POO	Pool.sol	contracts/Pool.sol
PBR	PoolBuilder.sol	contracts/PoolBuilder.sol
PFY	PoolFactory.sol	contracts/PoolFactory.sol
PVW	PoolView.sol	contracts/PoolView.sol
TOK	Token.sol	contracts/Token.sol
RRY	RepricerRegistry.sol	contracts/registries/RepricerRegistry.sol
IRR	IRepricer.sol	contracts/repricers/IRepricer.sol
X5R	x5Repricer.sol	contracts/repricers/x5Repricer.sol
IDS	IDerivativeSpecification.sol	contracts/libs/complifi/IDerivativeSpecification.sol
ARP	AddressRegistryParent.sol	contracts/libs/complifi/registries/AddressRegistryParent.sol
IAR	IAddressRegistry.sol	contracts/libs/complifi/registries/IAddressRegistry.sol
IER	IERC20Metadata.sol	contracts/libs/complifi/tokens/IERC20Metadata.sol
TMG	TokenMetadataGenerator.sol	contracts/libs/complifi/tokens/TokenMetadataGenerator.sol
BPB	BokkyPooBahsDateTimeLibrary.sol	contracts/libs/complifi/libs/BokkyPooBahsDateTimeLibrary/BokkyPooBahsDateTimeLibrary.sol

File Dependency Graph



Manual Review Findings

ID	Title	Type	Severity	Resolved
MAT-01M	Redundant `return` keyword	Language Specific	🟢 Informational	✓
MAT-02M	Redundant `return` statement	Language Specific	🟢 Informational	✓
MAT-03M	Redundant `return` statement	Language Specific	🟢 Informational	✓
NUM-01M	Explicitly returning local variable	Gas Optimization	🟢 Informational	✓
POO-01M	Lack of verification of the function parameter	Volatile Code	🟡 Medium	✓
POO-02M	`require` statement performs incorrect comparison	Inconsistency	🟡 Medium	✓
POO-03M	No event emitted for external state variable change	Language Specific	🔵 Minor	✓
POO-04M	Incorrect code	Logical Issue	🔵 Minor	✓
POO-05M	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
POO-06M	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	🔵 Minor	✓
POO-07M	Inefficient data structure	Gas Optimization	🟢 Informational	✓
POO-08M	`require` statements can be subsituted with a modifier	Language Specific	🟢 Informational	⏱
POO-09M	Inefficient function parameter	Gas Optimization	🟢 Informational	✓
POO-10M	Inefficient function parameter	Gas Optimization	🟢 Informational	✓
POO-11M	Inefficient storage read	Gas Optimization	🟢 Informational	✓
POO-12M	`require` statements can be subsituted with a modifier	Language Specific	🟢 Informational	✓

POO-13M	`require` statements can be subsituted with a modifier	Language Specific	● Informational	✓
POO-14M	Redundant casting to type `address`	Gas Optimization	● Informational	✓
PFY-01M	Functions declared before state variables	Language Specific	● Informational	✓
TOK-01M	Mutability Specifiers Missing	Gas Optimization	● Informational	✓
TOK-02M	Inefficient storage read	Gas Optimization	● Informational	✓
X5R-01M	Reundant casting to `int`	Gas Optimization	● Informational	✓
X5R-02M	Explicitly returning local variable	Gas Optimization	● Informational	✓

Static Analysis Findings

ID	Title	Type	Severity	Resolved
PFY-01S	Function visibility can be changed from `public` to `external`	Language Specific	● Informational	ⓘ
TOK-01S	Function visibility can be changed from `public` to `external`	Language Specific	● Informational	ⓘ

CERTIK

MAT-01M: Redundant `return` keyword

Type	Severity	Location
Language Specific	● Informational	Math.sol L36

Description:

The `return` keyword on the aforementioned line is redundant as the function can implicitly return and the return value is assigned to the named parameter of `spotPrice`.

Recommendation:

We advise to remove the `return` keyword on the aforementioned line.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

MAT-02M: Redundant `return` statement

Type	Severity	Location
Language Specific	● Informational	Math.sol L61

Description:

The `return` statement on the aforementioned line is redundant as it explicitly returns `tokenAmountOut` which is already implicitly returned by the function.

Recommendation:

We recommend to remove the `return` statement on the aforementioned line.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

MAT-03M: Redundant `return` statement

Type	Severity	Location
Language Specific	● Informational	Math.sol L87

Description:

The `return` statement on the aforementioned line is redundant as it explicitly returns `tokenAmountIn` which is already implicitly returned by the function.

Recommendation:

We recommend to remove the `return` statement on the aforementioned line.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

NUM-01M: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	● Informational	Num.sol L34, L43, L63, L75, L128

Description:

The functions on the aforementioned lines explicitly return local variable which increase overall gas of cost.

Recommendation:

Since named return variables can be declared in the signature of a function, consider refactoring to remove the local variable declaration and explicit return statement in order to reduce the overall cost of gas.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

POO-01M: Lack of verification of the function parameter

Type	Severity	Location
Volatile Code	● Medium	Pool.sol L206

Description:

The function parameter `manager` on the aforementioned line is not validated against zero address and it can result in unwanted state of the contract if a zero address value is passed.

Recommendation:

We advise to add check to validate the `manager` parameter against zero address value.

```
require(
   manager != address(0),
   "manager cannot be zero"
);
```

Alleviation:

The relevant code part is removed rendering this exhibit ineffective.

POO-02M: `require` statement performs incorrect comparison

Type	Severity	Location
Inconsistency	● Medium	Pool.sol L610

Description:

The `require` statement on the aforementioned line redundantly validates the outToken's leverage being greater zero than instead of validating inToken's leverage.

Recommendation:

We advise to rectify the `require` statement on the aforementioned line by validating the inToken's leverage such that it is greater than zero.

```
require(
   inToken.leverage > 0,
   "ZERO_IN_LEVERAGE"
);
```

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

POO-03M: No event emitted for external state variable change

Type	Severity	Location
Language Specific	● Minor	Pool.sol L206

Description:

The function `setController` assigns the new controller's address to the `_controller` state variable without emitting an event, which makes it difficult to track off-chain.

Recommendation:

Consider creating and emitting an event in order to track when the `_controller` state variable changes.

Alleviation:

The relevant code part is removed rendering this exhibit ineffective.

POO-04M: Incorrect code

Type	Severity	Location
Logical Issue	● Minor	Pool.sol L235-L236, 239, L259

Description:

The `require` statement on `L259` is ineffectual as `qMin` on `L259` always evaluates to `0`.

Recommendation:

We advise to update the `qMin` state variable before the `bind` function calls on `L235-L236`, so that the updated value for `qMin` is used by the `require` statement on `L259`.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

POO-05M: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	Pool.sol L643

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true` , many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa` . The code was added to check the both compliant and non-compliant ERC20 transfer.

POO-06M: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	Pool.sol L650

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true` , many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa` . The code was added to check the both compliant and non-compliant ERC20 transfer.

POO-07M: Inefficient data structure

Type	Severity	Location
Gas Optimization	● Informational	Pool.sol L101

Description:

The aforementioned line declared dynamic array of type `address` to store tokens' addresses. As the Pool contract only needs to store two tokens in a given instance, hence the dynamic array can be substituted with a fixed length array of size two which will cost less gas.

Recommendation:

We advise to subsitute the dynamic address type array on the aforementioned line with a fixed length array of size two to save gas cost.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

POO-08M: `require` statements can be subsituted with a modifier

Type	Severity	Location
Language Specific	● Informational	Pool.sol L211, L229

Description:

The `require` statements on the aforementioned lines can be subsituted with a modifier to increase the legibility of the codebase.

Recommendation:

We advise to subsitute the `require` statements on the aforementioned lines with a modifier.

```
modifier onlyController() {
   require(
      msg.sender == _controller,
      "NOT_CONTROLLER"
   );
}
```

Alleviation:

No alleviations.

POO-09M: Inefficient function parameter

Type	Severity	Location
Gas Optimization	● Informational	Pool.sol L273

Description:

The function parameter `maxAmountsIn` is a dynamic array of type `uint` . As the dynamic array always expects to have size of two, the dynamic array can be substituted with a fixed length array of size two which will be cheaper in terms of gas cost.

Recommendation:

We advise to utilize fixed length `uint` array of size two in place of dynamic `uint` array for the function parameter `maxAmountsIn` .

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa` .

POO-10M: Inefficient function parameter

Type	Severity	Location
Gas Optimization	● Informational	Pool.sol L299

Description:

The function parameter `minAmountsOut` is a dynamic array of type `uint` . As the dynamic array always expects to have size of two, the dynamic array can be substituted with a fixed length array of size two which will be cheaper in terms of gas cost.

Recommendation:

We advise to utilize fixed length `uint` array of size two in place of dynamic `uint` array for the function parameter `minAmountsOut` .

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa` .

POO-11M: Inefficient storage read

Type	Severity	Location
Gas Optimization	● Informational	Pool.sol L284, L313

Description:

The aforementioned lines perform inefficient storage read where the length of `_tokens` array are read repeatedly from the contract's storage.

Recommendation:

We advise to store the length of the array in a local variable as it will cost less gas compared to reading from contract's storage upon each iteration of the `for` loop.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

POO-12M: `require` statements can be subsituted with a modifier

Type	Severity	Location
Language Specific	● Informational	Pool.sol L278, L304, L438, L504

Description:

The `require` statements on the aforementioned lines can be subsituted with a modifier to increase the legibility of the codebase.

Recommendation:

We advise to subsitute the `require` statements on the aforementioned lines with a modifier.

```
modifier onlyFinalized() {
    require(
        _finalized,
       "NOT_FINALIZED"
    );
}
```

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

POO-13M: `require` statements can be subsituted with a modifier

Type	Severity	Location
Language Specific	● Informational	Pool.sol L233, L439, L505

Description:

The `require` statements on the aforementioned lines can be subsituted with a modifier to increase the legibility of the codebase.

Recommendation:

We advise to subsitute the `require` statements on the aforementioned lines with a modifier.

```
modifier onlySettled() {
   require(
      block.timestamp < derivativeVault.settleTime(),
      "SETTLED"
   );
}
```

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

POO-14M: Redundant casting to type `address`

Type	Severity	Location
Gas Optimization	● Informational	Pool.sol L392, L393, L444-L445, L509-L510

Description:

The aforementioned lines perform redundant castings of address type variables to type `address`

Recommendation:

We recommend to remove the redundant castings to type `address` on the aforementioned lines to save gas cost associated with it.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

PFY-01M: Functions declared before state variables

Type	Severity	Location
Language Specific	● Informational	PoolFactory.sol L39, L45

Description:

The functions on the aforementioned lines are declared before the contract's state variables and constructor.

Recommendation:

We advise to declare the functions on the aforementioned after the contract's constructor to increase the legibility of the codebase and to comply with standard Solidity contract's convention.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

TOK-01M: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	● Informational	Token.sol L75

Description:

The linked variables are assigned to only once, either during their contract-level declaration or during the `constructor`'s execution.

Recommendation:

For the former, we advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable. For the latter, we advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

TOK-02M: Inefficient storage read

Type	Severity	Location
Gas Optimization	● Informational	Token.sol L137

Description:

The function `transferFrom` on the aforementioned line performs inefficient storage read for `_allowance[src][msg.sender]` where it reads the same value multiple times from the contract's storage.

Recommendation:

We advise to store `_allowance[src][msg.sender]` in a local variable as it will cost less gas compared to reading from contract's storage and utilize it on `L138, L140, L141`.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa`.

X5R-01M: Reundant casting to `int`

Type	Severity	Location
Gas Optimization	● Informational	x5Repricer.sol L46-L47

Description:

The aforementioned lines redundantly cast `volatility` to type `int` .

Recommendation:

We advise to remove the redundant casting to `int` on the aforementioned lines to save gas cost associated with it.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa` .

X5R-02M: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	● Informational	x5Repricer.sol L87

Description:

The function on the aforementioned line explicitly returns local variable which increase overall gas of cost.

Recommendation:

Since named return variables can be declared in the signature of a function, consider refactoring to remove the local variable declaration and explicit return statement in order to reduce the overall cost of gas.

Alleviation:

Alleviations were applied as of commit hash `48fb4fb5a4d188eed90991d822997440261044aa` .

PFY-01S: Function visibility can be changed from `public` to `external`

Type	Severity	Location
Language Specific	🟢 Informational	PoolFactory.sol L104, L108

Description:

The functions on the aforementioned lines are never called within the contract and hence their visibilities can be changed to `external` to increase the legibility of the codebase.

Recommendation:

We recommend to change the functions' visibilities on the aforementioned lines from `public` to `external`.

Alleviation:

No alleviations.

TOK-01S: Function visibility can be changed from `public` to `external`

Type	Severity	Location
Language Specific	● Informational	Token.sol L85, L89, L93

Description:

The functions on the aforementioned lines are never called within the contract and hence their visibilities can be changed to `external` to increase the legibility of the codebase.

Recommendation:

We recommend to change the functions' visibilities on the aforementioned lines from `public` to `external`.

Alleviation:

No alleviations.

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.

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.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

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.