Security Assessment

AppleFinance

Jun 15th, 2021

Table of Contents

Summary

This report has been prepared for AppleFinance 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	AppleFinance
Platform	Polygon
Language	Solidity
Codebase	https://github.com/AppleFinance-Lending/contracts
Commit	0d3bc9d79f2be86ecc6179179aac7d685c2a25c0

Audit Summary

Delivery Date	Jun 15, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

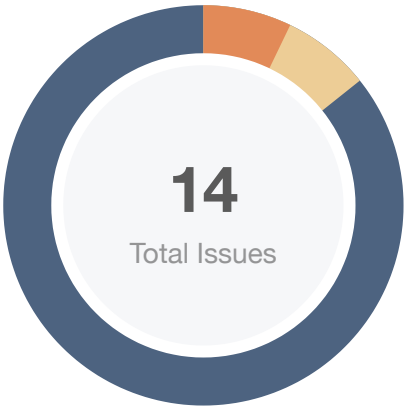
Total Issues	14
Critical	0
Major	1
Medium	0
Minor	1
Informational	12
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
CEA	CErc20.sol	0fc72edd2c29f18d635f562aea2f70aa266b32f9c68225f1c57fcd2505b4c7e4
CEF	CEther.sol	63cbd16d6a2fff1e3d62fc2812f6fbf128c10217931377a3a6ab574c4ec9cd35
CTA	CToken.sol	095be9bca968fa1a3d8a77e0b1c402365e07f4315e1d12cc385da83767128e83
CAF	Comp.sol	1c2f31720db642b64477d5bb32d88c8cd267c572167200f64bd4a449edd7a8b3
CLA	CompoundLens.sol	abcf989706b923b7f5511e2aeb8adf3b088a5318089bae1ea52f04c61692e2a9
CAL	Comptroller.sol	37b2ef0acbf6c80913224fcdae69a2ba546d0ab2f6c6b3363b3fc3ca261f8b87
CSA	ComptrollerStorage.sol	e22b3cdefdc6991a7e8e25b10814f28c604eacec93e26c5590e6e8fe724c6f33
DAF	Delegate.sol	988208fbc27d207a695876c5f6245401a2a66b15c14df47843feb5c265c2f036
JRM	JumpRateModelV3.sol	00877de63435c6aa08926a10122c6464ac1931359b979b9bccf0f6693895fb6b
MAF	Maxmillion.sol	19e5004c44c909b565ee0b6319c148e1e67203968769901b46b1d448ad370b28
POA	PriceOracle.sol	759ede10f6f75304e4f7f279e409ce4ec3bf4ea609b27ee41f187aa8d0879141
UAF	Unitroller.sol	8a2da5e0319c14485ed68c8c2109c9a04fa54c451d934529bb810f0529baab37
TAA	cToken-A.sol	8d41c8af1aad7e16cf12e6547480c0f824f2744831ea900093d1ecf27fe165e7
CTI	interfaces/CTokenInterfaces.sol	ada57bd9d68c3c8a1c4c5bca064a9f24a822c28d97319e05af2a3479f0a14037
CIA	interfaces/ComptrollerInterface.sol	fd798c05357efd45407008adff54cf2f143843a783ed74d30c23e503893aacf5
EIP	interfaces/EIP20Interface.sol	31131c00948885eaaefa7942a6bd1f3db5c40b328290c54e55846990992cde1d
EIN	interfaces/EIP20NonStandardInterface.sol	340ac8a992903d1c0ef2f71d4a2b51c5b1064dbab826d1afcf8849c6d57d9bb7
IRM	interfaces/InterestRateModel.sol	110b526aa1637c317d6c4935a6bfefbf6891cf18119f54161a76620c7f40abfd
CMA	libs/CarefulMath.sol	ffac3a93bc1b316760b8344f8f2c6f2107f4272797261650729c7706f4dd11b4
ERA	libs/ErrorReporter.sol	83b2db1f3a7dd053516544c19f138ca62e4960adc41d3ca4ee5bee89f18eca99

ID	file	SHA256 Checksum
EAF	libs/Exponential.sol	4927f74513e0c9df9a85bf721b9c4ccfe7e934bae1be1a78662baa775ba4e13c
ENE	libs/ExponentialNoError.sol	ccf8f454666481d55e294ecc3582e0045b263c85fe92b70ec3d2bd718c358a9b
OAF	libs/Operator.sol	ff87a28d0f5ed0f347e6e7ca09b0e2880cb7e0dfcf90bddc9474e36696f7cf89
OAL	libs/Ownable.sol	1a474ea9d3c199128dc436c1d2adde2ef437fcb2546e83d30e8e9e4dc7b92f42

Findings



Critical	**0** (0.00%)
Major	**1** (7.14%)
Medium	**0** (0.00%)
Minor	**1** (7.14%)
Informational	**12** (85.71%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	DAO missing	**Centralization / Privilege**	**Informational**	**Resolved**
CAF-01	Logical Issue in `Comp.sol`	Logical Issue	Informational	Resolved
CAF-02	Does Not Move Delegates While Mint Token	**Centralization / Privilege**	**Major**	**Resolved**
CAL-01	Unlocked Compiler Version Declaration	Language Specific	Informational	Resolved
CAL-02	Proper usage of “pure” and “view”	Coding Style	Informational	Resolved
CAL-03	Incorrect Naming Convention Utilization	Coding Style	Informational	Acknowledged
CAL-04	Proper Usage of “public” and “external” type	Coding Style	Informational	Resolved
CAL-05	Misuse of a Boolean Constant	Coding Style	Informational	Resolved
CAL-06	Return value not stored	Gas Optimization	Informational	Resolved
CAL-07	Boolean Equality	Gas Optimization	Informational	Resolved

ID	Title	Category	Severity	Status
CLA-01	Proposal related functions removed from `CompoundLens`	Logical Issue	Minor	Resolved
CTI-01	Incorrect Naming Convention Utilization	Coding Style	Informational	Acknowledged
DAF-01	Misuse of a Boolean Constant	Coding Style	Informational	Resolved
ENE-01	Incorrect Naming Convention Utilization	Coding Style	Informational	Acknowledged

GLOBAL-01 | DAO missing

Category	Severity	Location	Status
Centralization / Privilege	Informational	Global	Resolved

Description

The Compound protocol is governed and upgraded by COMP token-holders, using three distinct components; the COMP token, governance module (Governor Bravo), and Timelock. Together, these contracts allow the community to propose, vote, and implement changes through the administrative functions of a cToken or the Comptroller. Proposals can modify system parameters, support new markets, or add entirely new functionality to the protocol. The `GovernorAlpha` and `Timelock` contracts are removed, additionally functions `getGovReceipts`, `setProposal` and `getGovProposals` in contract `CompoundLens` are also removed, hence the DAO Governance implementation is disabled in this protocol.

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CAF-01 | Logical Issue in `Comp.sol`

Category	Severity	Location	Status
Logical Issue	Informational	Comp.sol: 109	Resolved

Description

The below codes are not matching with the specifications.

```
    uint public totalSupply = 200000e18; // 1 billion comp
```

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CAF-02 | Does Not Move Delegates While Mint Token

Category	Severity	Location	Status
Centralization / Privilege	**Major**	Comp.sol: 548	**Resolved**

Description

In essence, Comp Token lets token holders delegate their voting power to another entity. However, if the governance mints tokens to someone, the governance still maintains delegates. The bug is that the mint function does not call `_moveDelegates()`.

Recommendation

Consider adding call of `_moveDelegates()` in the function `mint()` and other functions that affects the token balance. Also make sure that `_delegates` mapping is correctly initialized, otherwise, delegation will be moved to address 0.

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CAL-01 | Unlocked Compiler Version Declaration

Category	Severity	Location	Status
Language Specific	Informational	Comptroller.sol: 1	Resolved

Description

The compiler version utilized throughout the project uses the "^" prefix specifier, denoting that a compiler version which is greater than the version will be used to compile the contracts. Recommend the compiler version should be consistent throughout the codebase.

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 e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CAL-02 | Proper usage of “pure” and “view”

Category	Severity	Location	Status
Coding Style	Informational	Comptroller.sol: 1764	Resolved

Description

Function state mutability can be restricted to pure, functions can be declared pure in which case they promise not to read from or modify the state.

Examples: function `getCompAddress()`

Recommendation

Consider declaring this function as pure.

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CAL-03 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	Informational	Comptroller.sol	Acknowledged

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

Functions other than constructors should use mixedCase

Variables should use mixedCase

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

Examples:

Constants like : `compClaimThreshold`, `compInitialIndex`, `closeFactorMinMantissa`, `closeFactorMaxMantissa`, `collateralFactorMaxMantissa`, `expScale`, `doubleScale`, `halfExpScale`, `mantissaOne`, `borrowRateMaxMantissa`, `reserveFactorMaxMantissa`, `isCToken`

Functions like : `mul_ScalarTruncate()`, `mul_ScalarTruncateAddUInt()`

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.

CAL-04 | Proper Usage of “public” and “external” type

Category	Severity	Location	Status
Coding Style	Informational	Comptroller.sol	Resolved

Description

“public” functions that are never called by the contract should be declared “external” . When the inputs are arrays, “external” functions are more efficient than “public” functions.

Examples:

Functions like : `enterMarkets()`, `getAccountLiquidity()`, `getHypotheticalAccountLiquidity()`, `refreshCompSpeeds()`, `claimComp()`, `getAllMarkets()`, `propose()`, `queue()`, `execute()`, `cancel()`, `getActions()`, `getReceipt()`, `castVote()`, `castVoteBySig()`, `__acceptAdmin()`, `__abdicate()`, `__queueSetTimelockPendingAdmin()`, `__executeSetTimelockPendingAdmin()`, `_setImplementation()`, `borrowBalanceStored()`, `exchangeRateCurrent()`, `exchangeRateStored()`, `accrueInterest()`, `_setComptroller()`, `delegateToViewImplementation()`, `_setPriceOracle()`, `_setPauseGuardian()`, `_setMintPaused()`, `_setBorrowPaused()`, `_setTransferPaused()`, `_setSeizePaused()`, `_grantComp()`, `_setCompRate()`, `_addCompMarkets()`, `_dropCompMarket()`

Recommendation

Consider using the “external” attribute for functions never called from the contract.

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CAL-05 | Misuse of a Boolean Constant

Category	Severity	Location	Status
Coding Style	Informational	Comptroller.sol	Resolved

Description

Boolean constants in code have only a few legitimate uses. Other uses (in complex expressions, as conditionals) indicate either an error or, most likely, the persistence of faulty code.

Examples:

```
if (false) {
            maxAssets = maxAssets;
}
```

Recommendation

Consider removing the ineffectual code.

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CAL-06 | Return value not stored

Category	Severity	Location	Status
Gas Optimization	Informational	Comptroller.sol	Resolved

Description

The return value of an external call is not stored in a local or state variable.

Examples:

```
function _supportMarket(CToken cToken) external returns (uint) {
...
cToken.isCToken();
...
}
```

Recommendation

Ensure that all the return values of the function calls are used.

Consider adding “require” statement for isCToken:

```
require(cToken.isCToken(),”This is not a CToken contract!”);
```

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CAL-07 | Boolean Equality

Category	Severity	Location	Status
Gas Optimization	Informational	Comptroller.sol	Resolved

Description

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

Example:

```
if (marketToJoin.accountMembership[borrower] == true) {
            // already joined
            return Error.NO_ERROR;
        }
```

Recommendation

Consider changing it as following:

```
 if (marketToJoin.accountMembership[borrower]) {...
}
```

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CLA-01 | Proposal related functions removed from `CompoundLens`

Category	Severity	Location	Status
Logical Issue	Minor	CompoundLens.sol	Resolved

Description

Functions `getGovReceipts()`, `setProposal()`, `getGovProposals()` were removed from CompoundLens. These functions reads data for governance.

Recommendation

Consider adding these functions back.

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

CTI-01 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	Informational	interfaces/CTokenInterfaces.sol	Acknowledged

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

Functions other than constructors should use mixedCase

Variables should use mixedCase

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

Examples:

Constants like : `compClaimThreshold`, `compInitialIndex`, `closeFactorMinMantissa`, `closeFactorMaxMantissa`, `collateralFactorMaxMantissa`, `expScale`, `doubleScale`, `halfExpScale`, `mantissaOne`, `borrowRateMaxMantissa`, `reserveFactorMaxMantissa`, `isCToken`

Functions like : `mul_ScalarTruncate()`, `mul_ScalarTruncateAddUInt()`

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.

DAF-01 | Misuse of a Boolean Constant

Category	Severity	Location	Status
Coding Style	Informational	Delegate.sol	Resolved

Description

Boolean constants in code have only a few legitimate uses. Other uses (in complex expressions, as conditionals) indicate either an error or, most likely, the persistence of faulty code.

Examples:

```
if (false) {
            maxAssets = maxAssets;
}
```

Recommendation

Consider removing the ineffectual code.

Alleviation

The team heeded our advice and resolved this issue in commit e288a690990b0d04b6c994e6e07afffa4f4fa4fc.

ENE-01 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	Informational	libs/ExponentialNoError.sol	Acknowledged

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

Functions other than constructors should use mixedCase

Variables should use mixedCase

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

Examples:

Constants like : `compClaimThreshold`, `compInitialIndex`, `closeFactorMinMantissa`, `closeFactorMaxMantissa`, `collateralFactorMaxMantissa`, `expScale`, `doubleScale`, `halfExpScale`, `mantissaOne`, `borrowRateMaxMantissa`, `reserveFactorMaxMantissa`, `isCToken`

Functions like : `mul_ScalarTruncate()`, `mul_ScalarTruncateAddUInt()`

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.

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.