SMART CONTRACT AUDIT REPORT

Amplify Tokenized Solutions Pte. Ltd.

By: Ankit Raj

Introduction

This audit report highlights the overall security of the [Amplify protocol](#) and their [token](#). With this report, I have tried to ensure the reliability of the smart contract by completing the assessment of their system's architecture and smart contract codebase.

Auditing approach and Methodologies applied

In this audit, I consider the following crucial features of the code.

- Whether the implementation of protocol standards.
- Whether the code is secure.
- Whether the code meets the best coding practices.
- Whether the code meets the SWC Registry issue.

The audit has been performed according to the following procedure:

• Manual audit

1. Inspecting the code line by line and revert the initial algorithms of the protocol and then compare them with the specification
2. Manually analyzing the code for security vulnerabilities.
3. Assessing the overall project structure, complexity & quality.
4. Checking SWC Registry issues in the code.
5. Unit testing by writing custom unit testing for each function.
6. Checking whether all the libraries used in the code of the latest version.
7. Analysis of security on-chain data.
8. Analysis of the failure preparations to check how the smart contract performs in case of bugs and vulnerability.

• Automated analysis

1. Scanning the project's code base with [Mythril](#), [Slither](#), [Echidna](#) , [Manticore](#) other's.
2. Manually verifying (reject or confirm) all the issues found by tools.
3. Performing Unit testing.
4. Manual Security Testing (SWC-Registry, Overflow)
5. Running the tests and checking their coverage.

Report: All the gathered information is described in this report.

Audit details

Project Name: Amplify Protocol with commit [86822b6](#)

Token symbol: [AMPT](#)

Token Supply: 100 million AMPT

Language: Solidity

Platform and tools: Remix, VScode, securify and other tools mentioned in the automated analysis section.

Audit Goals

The focus of this audit was to verify whether the smart contract is secure, resilient, and working properly according to the specs. The audit activity can be grouped in three categories.

Security: Identifying the security-related issue within each contract and system of contracts.

Sound architecture: Evaluating the architect of a system through the lens of established smart contract best practice and general software practice.

Code correctness and quality: A full review of contract source code. The primary area of focus includes.

- Correctness.
- Section of code with high complexity.
- Readability.
- Quantity and quality of test coverage.

Security

Every issue in this report was assigned a severity level from the following:

High severity issues
Issues mentioned here are critical to smart contract performance and functionality and should be fixed before moving to mainnet.

Medium severity issues
This could potentially bring the problem in the future and should be fixed.

Low severity issues
These are minor details and warnings that can remain unfixed but would be better if it got fixed in the future.

No. of issue per severity

Severity	High	Medium	Low
Open	0	0	6

Manual audit
Following are the reports from our manual analysis.

High severity issues

No High Severity Issue found.

Medium severity issues

No Medium Severity Issue found.

Low Severity Issues :
There were 6 low severity issues found in protocol and token contract.

1. **Unsafe Assumptions About Average Time Between Blocks**

The current implementation of the protocol uses *blocks* rather than *seconds* to measure the time between interest accruals. This makes the implementation highly sensitive to changes in the average time between Ethereum blocks.
On [line 19 of WhitePaperInterestRateModel.sol](#) it is implicitly assumed that the time between blocks is 15 seconds. However, the average time between blocks can change dramatically.

For example, the average time between blocks may increase by significant factors due to the difficulty bomb or decrease by significant factors during the transition to Serenity. The difference between the actual time between blocks and the assumed time between blocks causes proportional differences between the intended interest rates and the actual interest rates.

While the admin can combat this by adjusting the interest rate model when the average time between blocks changes, such adjustments are manual and happen only after-the-fact. Errors in blocktime assumptions are cumulative, and fixing the model after-the-fact does not make users whole – it only prevents incorrect interest calculations moving forward (until the next change in blocktime).

Consider refactoring the implementation to use *seconds* rather than *blocks* to measure the time between accruals. While block.timestamp can be manipulated by miners within a narrow window, these errors are small and, importantly, are not cumulative. This would decouple the interest rate model from Ethereum's average blocktime.

2. Require Statement Without Error Message

There is a require statement on [line 126 of CEther.sol](#) with no failure message. Consider adding a message to inform users in case of a revert. This msg will be displayed during failed operation.

3. 0 Address for Mints & Burns

Although not technically part of the EIP20 specification, it is common practice to use the zero address as the source for all Transfer events after minting, and as the destination for Transfers upon burning tokens. The Transfer events in functions [mintFresh](#) and [redeemFresh](#) use the address of the CToken contract instead. Consider using the zero address instead or informing users and developers of this feature.

4. Use assert

The **require** statement on line 1329 of CToken.sol is confirming a property that should never fail for any user input. In such a situation, consider using an **assert** statement instead as part of the right software practice.

In Token AMPT.sol

5. Costly loop [line 209]

Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of gas consumed in a block. If **array.length** is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed:

```
for (uint256 i = 0; i < array.length ; i++) {
    cosltyFunc();
}
```

This becomes a security issue, if an external actor influences array.length. E.g., if the array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.

6. Pure-functions should not read/change state line: [299-303]

In Solidity, functions that do not read from the state or modify it can be declared as pure.

Recommendation:

Do not declare functions that read from or modify the state as pure.

The following statements are considered modifying the state:

1. Writing to state variables

2. Emitting events;
3. Creating other contracts;
4. Using selfdestruct;
5. Sending Ether via calls;
6. Calling any function not marked view or pure;
7. Using low-level calls;
8. Using inline assembly that contains certain opcodes.

The following statements are considered reading from the state:

1. Reading from state variables;
2. Accessing this.balance or <adress>.balance;
3. Accessing any of the members of block, tx, msg (with the exception of msg.sig and msg.data);
4. Calling any function not marked pure;
5. Using inline assembly that contains certain opcodes

Automated test :

We have used multiple automated testing frameworks. This makes code more secure common attacks. The results are below.

Smart Check:

SmartCheck automatically checks Smart Contracts for vulnerabilities and bad practices. Automated tests have been conducted and got the following report. Few errors were found both in protocol and token contract. Detailed can be checked on the link below

https://tool.smartdec.net/scan/c1fa193ad119409289279a2e29dfff68

Unsafe array's length manipulation	⌄
Use of call function with no data	⌄
Deprecated constructions	⌄
Multiplication after division	⌄
Using approve function of the ERC-20 token standard	⌄
Strict comparison with block.timestamp or now	⌄
Extra gas consumption	⌄
Non-initialized return value	⌄
Costly loop	⌄
Blockhash function misuse	⌄
Locked money	⌄
msg.value == 0 check	⌄
Overpowered role	⌄
Compiler version not fixed	⌄

Private modifier	⌄
Revert inside the if-operator	⌄
Use of SafeMath	⌄
Send instead of transfer	⌄
Pure-functions should not read/change state	⌄
View-function should not change state	⌄
Replace multiple return values with a struct	⌄
Using tx.origin for authorization	⌄
Unchecked low-level call	⌄
Prefer external to public visibility level	⌄
Upgrade code to Solidity 0.5.x	⌄
Use of assembly	⌄
Unsafe type inference	⌄
Implicit visibility level	⌄

1. Hardcoded Address:

contracts/ComptrollerG4.sol [1379, 1433]

The contract contains unknown address. This address might be used for some malicious activity. Please check hardcoded address and it's usage.

Recommendation:

It is required to check the address. Also, it is required to check the code of the called contract for vulnerabilities.

2. Unsafe array's length manipulation

- contracts/ComptrollerG4.sol [Line: 220-220]
- contracts/Comptroller.sol [Line: 226-226]
- contracts/ComptrollerG2.sol [Line: 220-220]
- contracts/ComptrollerG3.sol [Line: 220-220]
- contracts/ComptrollerG1.sol [Line: 224-224]

The length of the dynamic array is changed directly. In this case, the appearance of gigantic arrays is possible and it can lead to a storage overlap attack (collisions with other data in storage).

Recommendation

If possible, avoid changing the length of the dynamic array directly.

- Use uint[] storage arrayName = new uint[](7) to create a dynamic array of the desired length.
- Use delete arrayName to clear a dynamic array.
- Use .push() (instead of .length++) to write to the end of the dynamic array.
- Starting with version 0.5.0 of the Solidity compiler, use .pop() (instead of .length--) to delete the last element of the dynamic array.

3. Multiplication after division

Solidity operates only with integers. Thus, if the division is done before the multiplication, the rounding errors can increase dramatically.

- contracts/DAIInterestRateModelV3.sol [line: 83-86]
- contracts/DAIInterestRateModelV3.sol [line: 94-94]

Recommendation:

Multiplication before division may increase the rounding precision.

4. Using approve function of the ERC-20 token standard

The approve function of ERC-20 is vulnerable. Using front-running attack one can spend approved tokens before change of allowance value.

- contracts/CErc20Delegator.sol [line: 183-186]
- contracts/CToken.sol [line: 158-163]

Recommendation:

Only use the approve function of the ERC-20 standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved).

5. Extra gas consumption

State variable, .balance, or .length of non-memory array is used in the condition of for or while loop. In this case, every iteration of loop consumes extra gas.

- contracts/Governance/GovernorAlpha.sol [line: 210-212, 180-182, 196,198]
- contracts/ComptrollerG4.sol   [line: 1018-1020]
- contracts/Lens/CompoundLens.sol [line: 230-253, 301-306,]
- contracts/Comptroller.sol [Line: 1034-1036]

Recommendation:

If a state variable, .balance, or .length is used several times, holding its value in a local variable is more gas efficient. If .length of calldata-array is placed into a local variable, the optimisation will be less significant.

6. Costly Loop

Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of gas consumed in a block. If array.length is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed:

```
for (uint256 i = 0; i < array.length ; i++) {

    cosltyFunc();

}
```

This becomes a security issue, if an external actor influences array.length. E.g., if array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.

- contracts/Governance/AMPT.sol [Line: 212-212]
- contracts/Governance/Comp.sol [Line:209-209]
- contracts/ComptrollerG4.sol [Line 1315-1317, 1318-1320, 1273-1289, 1285-1287, 122-126, 725-777, 1279-1281, 1109-1114, 1128-1133, 1118-1126, 207-212]
- contracts/Lens/CompoundLens.sol [Line: 301-306, 177-185, 230-253, 75-77, 144-146, 120-122]
- contracts/Comptroller.sol [Line: 128-132, , 1369-1371, 1034-1036, 1333-1335, 1055-1058, 741-793, 1163-1168, 1339-1341, 1182-1187, 213-218, 1172-1180]
- contracts/ComptrollerG2.sol [Line: 207-212, 122-126, 706-758]
- contracts/ComptrollerG3.sol [Line 1298-1300, 725-777, 1292-1294, 207-212, 1094-1096]

Recommendation:

Avoid loops with a big or unknown number of steps.

7. Locked money

● contracts/Unitroller.sol [Line 10-148]

Contracts programmed to receive ether should implement a way to withdraw it, i.e., call transfer (recommended), send, or call.value at least once.

Recommendation

Implement a withdraw function or reject payments (contracts without a fallback function do it automatically).

8. msg.value == 0 check

● contracts/CErc20Delegator.sol [Line: 453-453]

The msg.value == 0 condition check is meaningless in most cases.

Recommendation:

Avoid meaningless checks.

9. Overpowered role

● contracts/SimplePriceOracle.sol [Line: 44-47, 25-42]

This function is callable only from one address. Therefore, the system depends heavily on this address. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g. if the private key of this address becomes compromised.

Recommendation

We recommend designing contracts in a trustless manner. For instance, this functionality can be implemented in the contract's constructor. Another option is to use MultiSig wallet at this address.

10. Compiler version not fixed

Solidity source files indicate the versions of the compiler they can be compiled with.

```
pragma solidity ^0.4.17; // bad: compiles w 0.4.17 and above
pragma solidity 0.4.24; // good : compiles w 0.4.24 only
```

It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Recommendation:

Specify the exact compiler version (pragma solidity x.y.z;).

The detailed report of all above and other errors/notes can be found at this link.
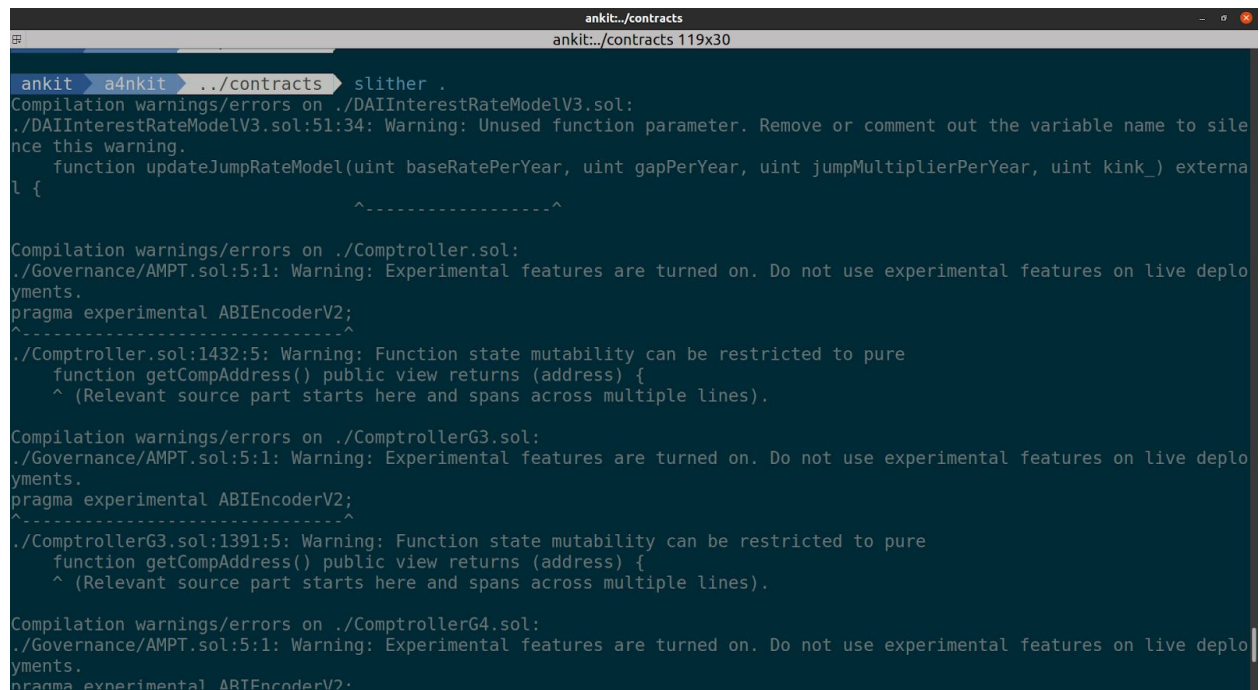
Slither:

Slither is a Solidity static analysis framework which runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to write custom analyses quickly. Slither enables developers to find vulnerabilities, enhance their code comprehension, and promptly prototype custom analyses. Each solidity file and project together has been analyzed. We got a report with a few warnings and errors.

Individual solidity file analysis at gist:

https://gist.github.com/aj07/ea3cfdd9be662c6396fdd44930ab3a80

We did the analysis of the project altogether. Below are the results.

```
Function CErc20._addReserves(uint256) (CErc20.sol#117-119) is not in mixedCase
Function CToken._setPendingAdmin(address) (CToken.sol#1102-1118) is not in mixedCase
Function CToken._acceptAdmin() (CToken.sol#1125-1145) is not in mixedCase
Function CToken._setComptroller(ComptrollerInterface) (CToken.sol#1152-1169) is not in mixedCase
Function CToken._setReserveFactor(uint256) (CToken.sol#1176-1184) is not in mixedCase
Function CToken._reduceReserves(uint256) (CToken.sol#1283-1291) is not in mixedCase
Function CToken._setInterestRateModel(InterestRateModel) (CToken.sol#1348-1356) is not in mixedCase
Variable CTokenStorage._notEntered (CTokenInterfaces.sol#10) is not in mixedCase
Constant CTokenStorage.borrowRateMaxMantissa (CTokenInterfaces.sol#31) is not in UPPER_CASE_WITH_UNDERSCORES
Constant CTokenStorage.reserveFactorMaxMantissa (CTokenInterfaces.sol#36) is not in UPPER_CASE_WITH_UNDERSCORES
Function CTokenInterface._setPendingAdmin(address) (CTokenInterfaces.sol#235) is not in mixedCase
Function CTokenInterface._acceptAdmin() (CTokenInterfaces.sol#236) is not in mixedCase
Function CTokenInterface._setComptroller(ComptrollerInterface) (CTokenInterfaces.sol#237) is not in mixedCase
Function CTokenInterface._setReserveFactor(uint256) (CTokenInterfaces.sol#238) is not in mixedCase
Function CTokenInterface._reduceReserves(uint256) (CTokenInterfaces.sol#239) is not in mixedCase
Function CTokenInterface._setInterestRateModel(InterestRateModel) (CTokenInterfaces.sol#240) is not in mixedCase
Constant CTokenInterface.isCToken (CTokenInterfaces.sol#123) is not in UPPER_CASE_WITH_UNDERSCORES
Function CErc20Interface._addReserves(uint256) (CTokenInterfaces.sol#265) is not in mixedCase
Function CDelegatorInterface._setImplementation(address,bool,bytes) (CTokenInterfaces.sol#287) is not in mixedCase
Function CDelegateInterface._becomeImplementation(bytes) (CTokenInterfaces.sol#296) is not in mixedCase
Function CDelegateInterface._resignImplementation() (CTokenInterfaces.sol#301) is not in mixedCase
Constant ComptrollerInterface.isComptroller (ComptrollerInterface.sol#5) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Exponential.expScale (Exponential.sol#13) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Exponential.doubleScale (Exponential.sol#14) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Exponential.halfExpScale (Exponential.sol#15) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Exponential.mantissaOne (Exponential.sol#16) is not in UPPER_CASE_WITH_UNDERSCORES
Constant InterestRateModel.isInterestRateModel (InterestRateModel.sol#9) is not in UPPER_CASE_WITH_UNDERSCORES
Constant PriceOracle.isPriceOracle (PriceOracle.sol#7) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
INFO:Detectors:
CDelegationStorage.implementation (CTokenInterfaces.sol#272) should be constant
```

```
            - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.LIQUIDATE_SEIZE_COMPTROLLER_REJECTION,allowed) (CT
oken.sol#1049)
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
            - failOpaque(Error.MATH_ERROR,FailureInfo.LIQUIDATE_SEIZE_BALANCE_DECREMENT_FAILED,uint256(mathErr)) (C
Token.sol#1068)
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
            - failOpaque(Error.MATH_ERROR,FailureInfo.LIQUIDATE_SEIZE_BALANCE_INCREMENT_FAILED,uint256(mathErr)) (C
Token.sol#1073)
        - Transfer(borrower,liquidator,seizeTokens) (CToken.sol#1085)
Reentrancy in CToken.transferTokens(address,address,address,uint256) (CToken.sol#68-127):
        External calls:
        - allowed = comptroller.transferAllowed(address(this),src,dst,tokens) (CToken.sol#70)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),0) (ErrorReporter.sol#194)
            - fail(Error.MATH_ERROR,FailureInfo.TRANSFER_NOT_ALLOWED) (CToken.sol#96)
        - Failure(uint256(err),uint256(info),0) (ErrorReporter.sol#194)
            - fail(Error.MATH_ERROR,FailureInfo.TRANSFER_TOO_MUCH) (CToken.sol#106)
        - Failure(uint256(err),uint256(info),0) (ErrorReporter.sol#194)
            - fail(Error.BAD_INPUT,FailureInfo.TRANSFER_NOT_ALLOWED) (CToken.sol#77)
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
            - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.TRANSFER_COMPTROLLER_REJECTION,allowed) (CToken.so
l#72)
        - Failure(uint256(err),uint256(info),0) (ErrorReporter.sol#194)
            - fail(Error.MATH_ERROR,FailureInfo.TRANSFER_NOT_ENOUGH) (CToken.sol#101)
        - Transfer(src,dst,tokens) (CToken.sol#122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
CErc20.doTransferIn(address,uint256) (CErc20.sol#142-167) uses assembly
        - INLINE ASM (CErc20.sol#148-160)
CErc20.doTransferOut(address,uint256) (CErc20.sol#178-197) uses assembly
        - INLINE ASM (CErc20.sol#183-195)
```

```
                - fail(Error.MARKET_NOT_FRESH,FailureInfo.REDEEM_FRESHNESS_CHECK) (CToken.sol#660)
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
                - failOpaque(Error.MATH_ERROR,FailureInfo.REDEEM_NEW_TOTAL_SUPPLY_CALCULATION_FAILED,uint256(vars.mathE
rr)) (CToken.sol#670)
        - Redeem(redeemer,vars.redeemAmount,vars.redeemTokens) (CToken.sol#701)
        - Transfer(redeemer,address(this),vars.redeemTokens) (CToken.sol#700)
Reentrancy in CToken.repayBorrowFresh(address,address,uint256) (CToken.sol#850-916):
        External calls:
        - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (CToken.sol#852)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
                - (failOpaque(Error.MATH_ERROR,FailureInfo.REPAY_BORROW_ACCUMULATED_BALANCE_CALCULATION_FAILED,uint256(
vars.mathErr)),0) (CToken.sol#870)
        - Failure(uint256(err),uint256(info),0) (ErrorReporter.sol#194)
                - (fail(Error.MARKET_NOT_FRESH,FailureInfo.REPAY_BORROW_FRESHNESS_CHECK),0) (CToken.sol#859)
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
                - (failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.REPAY_BORROW_COMPTROLLER_REJECTION,allowed),0) (C
Token.sol#854)
        - RepayBorrow(payer,borrower,vars.actualRepayAmount,vars.accountBorrowsNew,vars.totalBorrowsNew) (CToken.sol#91
0)
Reentrancy in CToken.seizeInternal(address,address,address,uint256) (CToken.sol#1045-1091):
        External calls:
        - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (CToken.sol#104
7)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),0) (ErrorReporter.sol#194)
                - fail(Error.INVALID_ACCOUNT_PAIR,FailureInfo.LIQUIDATE_SEIZE_LIQUIDATOR_IS_BORROWER) (CToken.sol#1054)
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
                - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.LIQUIDATE_SEIZE_COMPTROLLER_REJECTION,allowed) (CT
oken.sol#1049)
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
```

```
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (CToken.sol#940)
        - RepayBorrow(payer,borrower,vars.actualRepayAmount,vars.accountBorrowsNew,vars.totalBorrowsNew) (CToken.sol#91
0)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (CToken.sol#940)
        - Transfer(borrower,liquidator,seizeTokens) (CToken.sol#1085)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (CToken.sol#940)
Reentrancy in CToken.mintFresh(address,uint256) (CToken.sol#497-561):
        External calls:
        - allowed = comptroller.mintAllowed(address(this),minter,mintAmount) (CToken.sol#499)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
                - (failOpaque(Error.MATH_ERROR,FailureInfo.MINT_EXCHANGE_RATE_READ_FAILED,uint256(vars.mathErr)),0) (CT
oken.sol#513)
        - Failure(uint256(err),uint256(info),0) (ErrorReporter.sol#194)
                - (fail(Error.MARKET_NOT_FRESH,FailureInfo.MINT_FRESHNESS_CHECK),0) (CToken.sol#506)
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
                - (failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.MINT_COMPTROLLER_REJECTION,allowed),0) (CToken.so
l#501)
        - Mint(minter,vars.actualMintAmount,vars.mintTokens) (CToken.sol#554)
        - Transfer(address(this),minter,vars.mintTokens) (CToken.sol#555)
Reentrancy in CToken.redeemFresh(address,uint256,uint256) (CToken.sol#613-707):
        External calls:
        - allowed = comptroller.redeemAllowed(address(this),redeemer,vars.redeemTokens) (CToken.sol#653)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
                - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.REDEEM_COMPTROLLER_REJECTION,allowed) (CToken.sol#
655)
        - Failure(uint256(err),uint256(info),0) (ErrorReporter.sol#194)
                - fail(Error.TOKEN_INSUFFICIENT_CASH,FailureInfo.REDEEM_TRANSFER_OUT_NOT_POSSIBLE) (CToken.sol#680)
        - Failure(uint256(err),uint256(info),opaqueError) (ErrorReporter.sol#203)
                - failOpaque(Error.MATH_ERROR,FailureInfo.REDEEM_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED,uint256(vars.ma
thErr)) (CToken.sol#675)
```

16

While analysis with the slither there was some error/warnings/message from the tool on these files. A detailed report on each of the file has been put at gist.

 CToken.sol, CCompLikeDelegate.sol,   BaseJumpRateModelV2.sol, CDaiDelegate.sol, CErc20Delegate.sol, CErc20Immutable.sol, CEther.sol, Comptroller.sol, ComptrollerG2.sol , ComptrollerG4.sol, CTokenInterfaces.sol, DAIInterestRateModelV3.sol , Exponential.sol , JumpRateModel.sol , JumpRateModelV2.sol , LegacyJumpRateModelV2.sol , Maximillion.sol , PriceOracle.sol , SimplePriceOracle.sol ,   Unitroller.sol

Note

The repo given for audit contains both protocol and token contract.
The token contract was written in a standard format. There was no specific function to test recommended in the test case code written on the repo.

Also, at the Timelock contract, the MINIMUM_DELAY =1minutes. Usually, in a standard project, it's of 1/2days.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the code. Besides, a security audit, please don't consider this report as investment advice.

Summary

The use of smart contracts is simple and the code is relatively small. Altogether the code is written and demonstrates effective use of abstraction, separation of concern, and modularity. But there are a few issues/vulnerabilities to be tackled at various security levels, it is recommended to fix them before deploying the contract on the main network. Given the subjective nature of some assessments, it will be up to the Amplify-protocol team to decide whether any changes should be made.

About the Auditor: Ankit Raj

Ankit is a technology expert with many years of expert experience building, managing, and automating systems at scale for blockchain, distributed systems, and storage projects.

Started a career as a developer with Red Hat where he developed the DHT module for GlusterFS. GlusterFS is being used by Facebook and financial institutions for clustering large chunks of data, images, and videos. Later he got a grant from Ethereum Foundation to work on Solidity language. There he wrote solidity code as well as maintained docs for the solidity programming language which is used by the developer across the globe. Then he worked with various crypto startups like Ocean Protocol, Coss exchange leading a full-stack development team. At Ocean, he built the protocol for safe data transfer. Ankit also founded Blockvidhya, a document verification startup service relying on the blockchain, which was incubated at IIT Mandi and part of YC startup school. He was also part of Entrepreneur First in Singapore where he was leveraging his blockchain and Open Finance skills.

Currently, he is actively doing contributions in Ethereum, Polkadot & Near Protocol ecosystem, and doing research around Open Finance.

During his free time, he participates in hackathons. He has won more than 5 international hackathons across the globe organized by Ethglobal, Matic, Near Protocol and Barclays labs. He also actively writes around open finance and DeFi protocols.
Link: Linkedin, Twitter, Medium, website