Cyberscope

Audit Report

Merchant Comptroller

May 2024

Network	Merlin
Address	0xa3cCe569546bCa5eE618174bb83740c61721230a
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Comptroller
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://scan.merlinchain.io/address/0xa3cCe569546bCa5eE618174bb83740c61721230a
Address	0xa3cCe569546bCa5eE618174bb83740c61721230a
Network	Merlin

Audit Updates

Initial Audit	09 May 2024

Source Files

Filename	SHA256
Unitroller.sol	e9f46495fb0c8eb9fb8b573e2a2907230ec0c4a35bf6a6fa3672e6a9045f301f
PriceOracle.sol	28eaed5913198dfe65150ad5b73ba415c7910e5a773bcf566d198e3ec1e95241
InterestRateModel.sol	8ca958179765a9ef12f955a76afdd6ac8bdacbe0be61216b6329e674b5739e7d

ExponentialNoError.sol	ef79b0e99297f924296b136e84d89861704 09eb233820b3e7733c2a6387707e9
ErrorReporter.sol	97062c28c271dac34dd5b46e74bfa31d6d 75f52b4c8c34653f275b2578a5e000
EIP20NonStandardInterface.sol	a2345b0d95fefe5a7256c2f7fb3bd3e1982 0b06d38c27235b8820116920a5894
EIP20Interface.sol	e1670bdc5381a0a06e605adf1d24226d25 132a43b560f2a190faf2d2cbf60aed
ComptrollerStorage.sol	b9127b3192073721e4240021b6fa99ab94 995db7855b327c9d1491ff46700df9
ComptrollerInterface.sol	98e442b63ace6598e6a4c46b539b5f2b41 a9fbe486c284e7e8acf11165b3be95
Comptroller.sol	bdadfc6e984d60b19b0643ed6f48e21f4c1 4400638454fdcfd498bc06342af86
CTokenInterfaces.sol	7016d4ab7b6a30e571002f8172a38746b0 f3025d94935945f8cfa792a1d9920a
CToken.sol	c681be80c13f47d3ba2dd91b625e708246 8c19e0f846be8135cda9edca2e7390
Governance/Comp.sol	20fc9139509172b8c6a7f7ca27deefb41b9 adbd39a7b8b0583f0debd98889686

Overview

The Merchant Comptroller contract's primary functions include managing the protocol's core operations such as liquidity and collateral checks, interest rate calculations, and the enforcement of borrowing and lending conditions. This contract determines which assets are available in the market, sets and adjusts collateral factors, and ensures adequate liquidity and collateral levels to support borrowing activities. It also calculates interest rates dynamically based on asset utilization rates to balance supply and demand within the market.

Additionally, the contract is responsible for the governance aspect of the platform. It facilitates the proposal and voting processes, allowing MECH token holders to participate in protocol governance by submitting and voting on proposals that influence the protocol's rules and future development. Another significant role of the Comptroller is the distribution of MECH tokens, which are awarded to users based on their interaction with the protocol, such as borrowing, lending, and providing liquidity, thereby incentivizing continued and meaningful participation in the ecosystem. This contract ensures that all operations adhere to the established protocols and risk parameters, maintaining the overall health and stability of the system.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	DI	Documentation Inconsistency	Unresolved
●	LO	Loop Optimization	Unresolved
●	MMN	Misleading Method Naming	Unresolved
●	RV	Redundant Validation	Unresolved
●	SMU	Suboptimal Memory Usage	Unresolved
●	URV	Unused Return Value	Unresolved
●	L03	Redundant Statements	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

DI - Documentation Inconsistency

Criticality	Minor / Informative
Location	Comptroller.sol#L1918
Status	Unresolved

Description

The `getCompAddress` function is intended to return the address of a specific token. The current documentation for the function describes it as returning the address of the SONNE token. However, upon examination, the address actually returned by the function (`0xf34CFB7c220f71dcfD3E94fFf48a428815c1fE99`) corresponds to the MECH token. This inconsistency between the function's documented purpose and its actual output creates a documentation mismatch that can lead to misunderstandings about which token is being referred to.

```
function getCompAddress() public view virtual returns (address)
{
    return 0xf34CFB7c220f71dcfD3E94fFf48a428815c1fE99;
}
```

Recommendation

To resolve this documentation inconsistency, the comments within the `getCompAddress` function should be updated to accurately reflect the token associated with the returned address. Specifically, the comment should state that the function returns the address of the MECH token instead of the SONNE token. Correcting this documentation will align the function’s description with its actual behavior, ensuring clarity and preventing potential confusion or errors in the use of the contract.

LO - Loop Optimization

Criticality	Minor / Informative
Location	Comptroller.sol#L929,1274,1452,1771
Status	Unresolved

Description

The contract code exhibits inefficient loop structures across several functions. In these loops, the size of the array being iterated over is recalculated in each iteration rather than being stored as a variable at the beginning. This approach results in unnecessary operations depending on the storage type of the array: an additional `sload` operation for storage arrays, costing an extra 100 gas per iteration due to EIP-2929, and an additional `mload` or `calldataload` operation for memory or calldata arrays, respectively, each adding 3 gas per iteration. These repetitive and redundant operations increase the overall gas cost of transactions, which could have been avoided.

```
function claimComp(
    address[] memory holders,
    CToken[] memory cTokens,
    bool borrowers,
    bool suppliers
) public {
    for (uint256 i = 0; i < cTokens.length; i++) {
        CToken cToken = cTokens[i];
        require(markets[address(cToken)].isListed, "market must
be listed");
        if (borrowers == true) {
            Exp memory borrowIndex = Exp({mantissa:
cToken.borrowIndex()});
            updateCompBorrowIndex(address(cToken),
borrowIndex);
            for (uint256 j = 0; j < holders.length; j++) {
                distributeBorrowerComp(
                    address(cToken),
                    holders[j],
                    borrowIndex
                );
            }
        }
        if (suppliers == true) {
            updateCompSupplyIndex(address(cToken));
            for (uint256 j = 0; j < holders.length; j++) {
                distributeSupplierComp(address(cToken),
holders[j]);
            }
        }
    }
    for (uint256 j = 0; j < holders.length; j++) {
        compAccrued[holders[j]] = grantCompInternal(
            holders[j],
            compAccrued[holders[j]]
        );
    }
}
```

Recommendation

To optimize gas usage and improve the efficiency of the smart contract, it is recommended to refactor the loops within the affected functions. Specifically, the length of the array should be stored in a local variable at the start of the loop. This change will prevent repeated length calculations on each iteration, reducing the number of costly sload, mload, or calldataload operations. Implementing this small but impactful optimization will decrease the gas costs and enhance the contract's performance.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	Comptroller.sol#L1910
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Specifically, the `getBlockNumber` function is currently implemented to return `block.timestamp` instead of `block.number`. The name of the function suggests that it should return the number of the current block in the blockchain, which is a sequential identifier. However, the function's implementation returns the timestamp of the current block, which indicates the time at which the block was mined.

```
function getBlockNumber() public view virtual returns (uint256)
{
    return block.timestamp;
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

RV - Redundant Validation

Criticality	Minor / Informative
Location	Comptroller.sol#L230,381
Status	Unresolved

Description

The `exitMarket` function in the contract includes a redundant check to verify whether the sender is already active within the market. This validation is unnecessarily duplicated, as the internal function `redeemAllowedInternal`, which is invoked by `exitMarket`, performs the same check.

```
function exitMarket(address cTokenAddress)
    external
    override
    returns (uint256)
{
    ...
        /* Fail if the sender is not permitted to redeem all of
their tokens */
        uint256 allowed = redeemAllowedInternal(
            cTokenAddress,
            msg.sender,
            tokensHeld
        );
    ...
    return uint256(Error.NO_ERROR);
}

function redeemAllowedInternal(
    address cToken,
    address redeemer,
    uint256 redeemTokens
) internal view returns (uint256) {
    if (!markets[cToken].isListed) {
        return uint256(Error.MARKET_NOT_LISTED);
    }

    /* If the redeemer is not 'in' the market, then we can
bypass the liquidity check */
    if (!markets[cToken].accountMembership[redeemer]) {
        return uint256(Error.NO_ERROR);
    }

    /* Otherwise, perform a hypothetical liquidity check to
guard against shortfall */
    (
        Error err,
        ,
        uint256 shortfall
    ) = getHypotheticalAccountLiquidityInternal(
            redeemer,
            CToken(cToken),
            redeemTokens,
            0
        );
    if (err != Error.NO_ERROR) {
        return uint256(err);
    }
    if (shortfall > 0) {
        return uint256(Error.INSUFFICIENT_LIQUIDITY);
    }
```

```
    return uint256(Error.NO_ERROR);
}
```

Recommendation

It is recommended to remove the redundant validation from the `exitMarket` function. Since the necessary check is already performed by `redeemAllowedInternal`, eliminating the duplicate validation in `exitMarket` will reduce the overall execution cost and simplify the function's logic.

SMU - Suboptimal Memory Usage

Criticality	Minor / Informative
Location	Comptroller.sol#L171,1592,1675,1771,1847
Status	Unresolved

Description

Several functions utilize `memory` for parameters that are strictly read-only. This implementation choice introduces inefficiencies. Using `memory` for such parameters involves unnecessary data copying when `calldata` could be used instead. `calldata` is a non-modifiable, non-persistent area where function arguments are stored and behaves more efficiently for read-only data because it avoids the extra steps and gas costs associated with copying data to memory.

```
function enterMarkets(address[] memory cTokens)
    public
    override
    returns (uint256[] memory)
{
    uint256 len = cTokens.length;

    uint256[] memory results = new uint256[](len);
    for (uint256 i = 0; i < len; i++) {
        CToken cToken = CToken(cTokens[i]);

        results[i] = uint256(addToMarketInternal(cToken,
msg.sender));
    }

    return results;
}

...
```

Recommendation

It is advisable to review the function signatures in the affected functions to utilize calldata for parameters that are only read and not modified. This adjustment will make the contract more gas-efficient, reducing the cost of calls to these functions.

URV - Unused Return Value

Criticality	Minor / Informative
Location	Comptroller.sol#L1258
Status	Unresolved

Description

There is an issue where the `isCToken` getter from the cToken contract is called during the process of listing a new market. However, the return value of this call is not checked, which renders the verification step ineffective. If the called address lacks an `isCToken` getter and the target is not an externally owned account (EOA), the fallback function will be executed instead. This could potentially allow the addition of invalid cToken contracts to the market. If the `isCToken` call returns false—indicating that the contract is not a valid cToken—the market listing should be prevented, but currently, it proceeds without this check.

```
function _supportMarket(CToken cToken) external returns
(uint256) {
    if (msg.sender != admin) {
        return
            fail(
                Error.UNAUTHORIZED,
                FailureInfo.SUPPORT_MARKET_OWNER_CHECK
            );
    }

    if (markets[address(cToken)].isListed) {
        return
            fail(
                Error.MARKET_ALREADY_LISTED,
                FailureInfo.SUPPORT_MARKET_EXISTS
            );
    }

    cToken.isCToken(); // Sanity check to make sure its really
a CToken

    // Note that isComped is not in active use anymore
    Market storage newMarket = markets[address(cToken)];
    newMarket.isListed = true;
    newMarket.isComped = false;
    newMarket.collateralFactorMantissa = 0;

    _addMarketInternal(address(cToken));
    _initializeMarket(address(cToken));

    emit MarketListed(cToken);

    return uint256(Error.NO_ERROR);
}
```

Recommendation

It is recommended to ensure that return values from such verification calls are explicitly checked. The function should include a check of the return value from the `isCToken` call. If the return value is false or the function call fails to reach a valid contract with the necessary getter, the transaction should revert to prevent the listing of invalid markets.

L03 - Redundant Statements

Criticality	Minor / Informative
Location	Comptroller.sol#L16
Status	Unresolved

Description

Redundant statements are statements that are unnecessary or have no effect on the contract's behavior. These can include declarations of variables or functions that are not used, or assignments to variables that are never used.

As a result, it can make the contract's code harder to read and maintain, and can also increase the contract's size and gas consumption, potentially making it more expensive to deploy and execute.

```
contract Comptroller is
    ComptrollerV7Storage,
    ComptrollerInterface,
    ComptrollerErrorReporter,
    ExponentialNoError
{
...
     * @notice Return the address of the SONNE token
     * @return The address of SONNE
     */
    function getCompAddress() public view virtual returns
(address) {
        return 0xf34CFB7c220f71dcfD3E94fFf48a428815c1fE99;
    }
}
```

Recommendation

To avoid redundant statements, it's important to carefully review the contract's code and remove any statements that are unnecessary or not used. This can help to improve the clarity and efficiency of the contract's code.

By removing unnecessary or redundant statements from the contract's code, the clarity and efficiency of the contract will be improved. Additionally, the size and gas consumption will be reduced.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Comptroller.sol#L123,126,129,1089,1117,1138,1207,1241,1315,1342,1360,1384,1400,1416,1428,1440,1834,1847,1871
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint256 internal constant closeFactorMinMantissa = 0.05e18
uint256 internal constant closeFactorMaxMantissa = 0.9e18
uint256 internal constant collateralFactorMaxMantissa = 0.9e18

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	Comptroller.sol#L123,126
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
uint256 internal constant closeFactorMinMantissa = 0.05e18
uint256 internal constant closeFactorMaxMantissa = 0.9e18
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	Comptroller.sol#L205,1393,1409,1421,1433,1780,1791,1904
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
if (marketToJoin.accountMembership[borrower] == true) {

require(msg.sender == admin || state == true, "only admin can
unpause");

...
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Comptroller.sol#L1467
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 newAccrual;
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Comptroller.sol#L1349,1376
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
borrowCapGuardian = newBorrowCapGuardian;

pauseGuardian = newPauseGuardian;
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Comptroller.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.10;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Comptroller.sol#L1820
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
comp.transfer(user, amount);
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Unitroller	Implementation	UnitrollerAdminStorage, ComptrollerErrorReporter		
		Public	✓	-
	_setPendingImplementation	Public	✓	-
	_acceptImplementation	Public	✓	-
	_setPendingAdmin	Public	✓	-
	_acceptAdmin	Public	✓	-
		External	Payable	-
PriceOracle	Implementation			
	getUnderlyingPrice	External		-
InterestRateModel	Implementation			
	getBorrowRate	External		-
	getSupplyRate	External		-
ExponentialNoError	Implementation			
	truncate	Internal		
	mul_ScalarTruncate	Internal		

	mul_ScalarTruncateAddUInt	Internal		
	lessThanExp	Internal		
	lessThanOrEqualExp	Internal		
	greaterThanExp	Internal		
	isZeroExp	Internal		
	safe224	Internal		
	safe32	Internal		
	add_	Internal		
	add_	Internal		
	add_	Internal		
	sub_	Internal		
	sub_	Internal		
	sub_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	div_	Internal		
	div_	Internal		
	div_	Internal		

	div_	Internal		
	div_	Internal		
	div_	Internal		
	div_	Internal		
	fraction	Internal		
ComptrollerErrorReporter	Implementation			
	fail	Internal	✓	
	failOpaque	Internal	✓	
TokenErrorReporter	Implementation			
EIP20NonStandardInterface	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	allowance	External		-
EIP20Interface	Interface			
	name	External		-
	symbol	External		-

	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	allowance	External		-
UnitrollerAdminStorage	Implementation			
ComptrollerV1Storage	Implementation	UnitrollerAdminStorage		
ComptrollerV2Storage	Implementation	ComptrollerV1Storage		
ComptrollerV3Storage	Implementation	ComptrollerV2Storage		
ComptrollerV4Storage	Implementation	ComptrollerV3Storage		
ComptrollerV5Storage	Implementation	ComptrollerV4Storage		
ComptrollerV6Storage	Implementation	ComptrollerV5Storage		

ComptrollerV7Storage	Implementation	ComptrollerV6Storage		
ComptrollerInterface	Implementation			
	enterMarkets	External	✓	-
	exitMarket	External	✓	-
	mintAllowed	External	✓	-
	mintVerify	External	✓	-
	redeemAllowed	External	✓	-
	redeemVerify	External	✓	-
	borrowAllowed	External	✓	-
	borrowVerify	External	✓	-
	repayBorrowAllowed	External	✓	-
	repayBorrowVerify	External	✓	-
	liquidateBorrowAllowed	External	✓	-
	liquidateBorrowVerify	External	✓	-
	seizeAllowed	External	✓	-
	seizeVerify	External	✓	-
	transferAllowed	External	✓	-
	transferVerify	External	✓	-
	liquidateCalculateSeizeTokens	External		-

Comptroller	Implementation	ComptrollerV7Storage, ComptrollerInterface, ComptrollerErrorReporter, ExponentialNoError		
		Public	✓	-
	getAssetsIn	External		-
	checkMembership	External		-
	enterMarkets	Public	✓	-
	addToMarketInternal	Internal	✓	
	exitMarket	External	✓	-
	mintAllowed	External	✓	-
	mintVerify	External	✓	-
	redeemAllowed	External	✓	-
	redeemAllowedInternal	Internal		
	redeemVerify	External	✓	-
	borrowAllowed	External	✓	-
	borrowVerify	External	✓	-
	repayBorrowAllowed	External	✓	-
	repayBorrowVerify	External	✓	-
	liquidateBorrowAllowed	External	✓	-
	liquidateBorrowVerify	External	✓	-
	seizeAllowed	External	✓	-
	seizeVerify	External	✓	-
	transferAllowed	External	✓	-

	transferVerify	External	✓	-
	getAccountLiquidity	Public		-
	getAccountLiquidityInternal	Internal		
	getHypotheticalAccountLiquidity	Public		-
	getHypotheticalAccountLiquidityInternal	Internal		
	liquidateCalculateSeizeTokens	External		-
	_setPriceOracle	Public	✓	-
	_setCloseFactor	External	✓	-
	_setCollateralFactor	External	✓	-
	_setLiquidationIncentive	External	✓	-
	_supportMarket	External	✓	-
	_addMarketInternal	Internal	✓	
	_initializeMarket	Internal	✓	
	_setMarketBorrowCaps	External	✓	-
	_setBorrowCapGuardian	External	✓	-
	_setPauseGuardian	Public	✓	-
	_setMintPaused	Public	✓	-
	_setBorrowPaused	Public	✓	-
	_setTransferPaused	Public	✓	-
	_setSeizePaused	Public	✓	-
	_become	Public	✓	-
	fixBadAccruals	External	✓	-
	adminOrInitializing	Internal		

	setCompSpeedInternal	Internal	✓	
	updateCompSupplyIndex	Internal	✓	
	updateCompBorrowIndex	Internal	✓	
	distributeSupplierComp	Internal	✓	
	distributeBorrowerComp	Internal	✓	
	updateContributorRewards	Public	✓	-
	claimComp	Public	✓	-
	claimComp	Public	✓	-
	claimComp	Public	✓	-
	grantCompInternal	Internal	✓	
	_grantComp	Public	✓	-
	_setCompSpeeds	Public	✓	-
	_setContributorCompSpeed	Public	✓	-
	getAllMarkets	Public		-
	isDeprecated	Public		-
	getBlockNumber	Public		-
	getCompAddress	Public		-
CTokenStorage	Implementation			
CTokenInterface	Implementation	CTokenStorage		
	transfer	External	✓	-
	transferFrom	External	✓	-

	approve	External	✓	-
	allowance	External		-
	balanceOf	External		-
	balanceOfUnderlying	External	✓	-
	getAccountSnapshot	External		-
	borrowRatePerBlock	External		-
	supplyRatePerBlock	External		-
	totalBorrowsCurrent	External	✓	-
	borrowBalanceCurrent	External	✓	-
	borrowBalanceStored	External		-
	exchangeRateCurrent	External	✓	-
	exchangeRateStored	External		-
	getCash	External		-
	accrueInterest	External	✓	-
	seize	External	✓	-
	_setPendingAdmin	External	✓	-
	_acceptAdmin	External	✓	-
	_setComptroller	External	✓	-
	_setReserveFactor	External	✓	-
	_reduceReserves	External	✓	-
	_setInterestRateModel	External	✓	-
CErc20Storage	Implementation			

CErc20Interface	Implementation	CErc20Storage		
	mint	External	✓	-
	redeem	External	✓	-
	redeemUnderlying	External	✓	-
	borrow	External	✓	-
	repayBorrow	External	✓	-
	repayBorrowBehalf	External	✓	-
	liquidateBorrow	External	✓	-
	sweepToken	External	✓	-
	_addReserves	External	✓	-
CDelegationStorage	Implementation			
CDelegatorInterface	Implementation	CDelegationStorage		
	_setImplementation	External	✓	-
CDelegateInterface	Implementation	CDelegationStorage		
	_becomeImplementation	External	✓	-
	_resignImplementation	External	✓	-

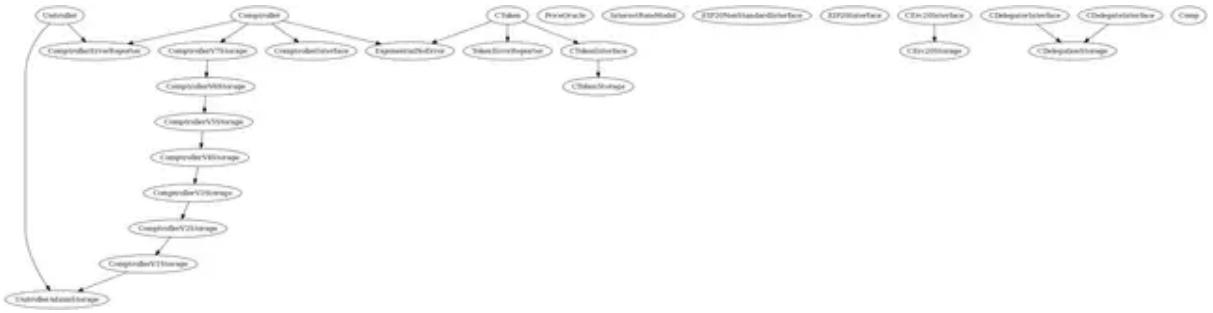
CToken	Implementation	CTokenInterface, Exponential NoError, TokenErrorReporter		
	initialize	Public	✓	-
	transferTokens	Internal	✓	
	transfer	External	✓	nonReentrant
	transferFrom	External	✓	nonReentrant
	approve	External	✓	-
	allowance	External		-
	balanceOf	External		-
	balanceOfUnderlying	External	✓	-
	getAccountSnapshot	External		-
	getBlockNumber	Internal		
	borrowRatePerBlock	External		-
	supplyRatePerBlock	External		-
	totalBorrowsCurrent	External	✓	nonReentrant
	borrowBalanceCurrent	External	✓	nonReentrant
	borrowBalanceStored	Public		-
	borrowBalanceStoredInternal	Internal		
	exchangeRateCurrent	Public	✓	nonReentrant
	exchangeRateStored	Public		-
	exchangeRateStoredInternal	Internal		
	getCash	External		-
	accrueInterest	Public	✓	-

	mintInternal	Internal	✓	nonReentrant
	mintFresh	Internal	✓	
	redeemInternal	Internal	✓	nonReentrant
	redeemUnderlyingInternal	Internal	✓	nonReentrant
	redeemFresh	Internal	✓	
	borrowInternal	Internal	✓	nonReentrant
	borrowFresh	Internal	✓	
	repayBorrowInternal	Internal	✓	nonReentrant
	repayBorrowBehalfInternal	Internal	✓	nonReentrant
	repayBorrowFresh	Internal	✓	
	liquidateBorrowInternal	Internal	✓	nonReentrant
	liquidateBorrowFresh	Internal	✓	
	seize	External	✓	nonReentrant
	seizeInternal	Internal	✓	
	_setPendingAdmin	External	✓	-
	_acceptAdmin	External	✓	-
	_setComptroller	Public	✓	-
	_setReserveFactor	External	✓	nonReentrant
	_setReserveFactorFresh	Internal	✓	
	_addReservesInternal	Internal	✓	nonReentrant
	_addReservesFresh	Internal	✓	
	_reduceReserves	External	✓	nonReentrant
	_reduceReservesFresh	Internal	✓	

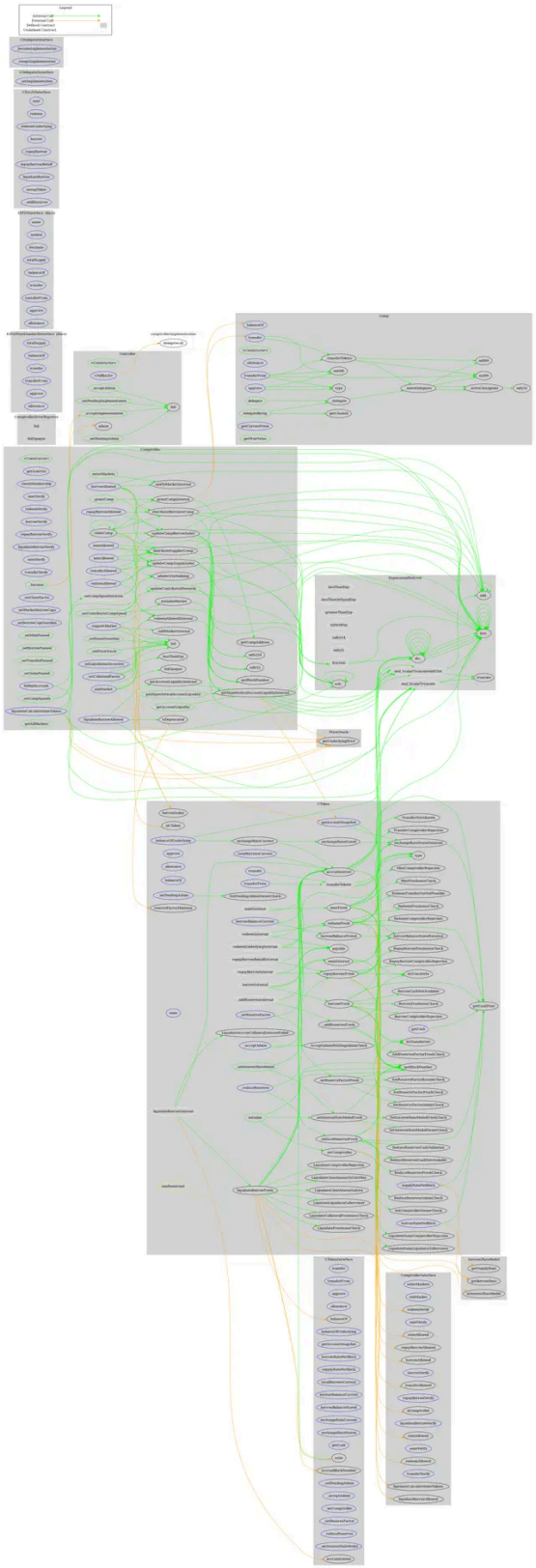
	_setInterestRateModel	Public	✓	-
	_setInterestRateModelFresh	Internal	✓	
	getCashPrior	Internal		
	doTransferIn	Internal	✓	
	doTransferOut	Internal	✓	
Comp	Implementation			
		Public	✓	-
	allowance	External		-
	approve	External	✓	-
	balanceOf	External		-
	transfer	External	✓	-
	transferFrom	External	✓	-
	delegate	Public	✓	-
	delegateBySig	Public	✓	-
	getCurrentVotes	External		-
	getPriorVotes	Public		-
	_delegate	Internal	✓	
	_transferTokens	Internal	✓	
	_moveDelegates	Internal	✓	
	_writeCheckpoint	Internal	✓	
	safe32	Internal		
	safe96	Internal		

	add96	Internal		
	sub96	Internal		
	getChainId	Internal		

Inheritance Graph



Flow Graph



Summary

Merchant Comptroller contract implements a central governance mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope'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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 represents an extensive assessment 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io