ANCILIA

Sumer AUDITING REPORT

v1.3

June 2024

Prepared for

Sumer.money

Prepared by

Ancilia, Inc.

Index

Version History

Version	Description	Date
v1.3	Reviewed one more branch	06/21/2024
v1.2	Final report	06/18/2024
v1.1	Add new findings	06/03/2024
v1.0	Initial report	05/20/2024

Executive Summary

The Sumer.money team shared their smart contract source code via github. We have listed tags of smart contracts to ensure the entirety of the audit can be tied to a given contract version. The Ancilia team has worked with the Sumer.money team on all potential findings and issues. The audit scope includes checking for smart contracts with attack vulnerabilities such as re-entry attacks, logic flaws, authentication bypasses, DoS attacks, etc. Our researchers primarily focused on CToken, Comptroller and interest models. The deployment scripts, price-oracle, staking and underlying token contracts are not in the scope of this audit.

Disclaimer

Note that security audit services do not guarantee to find all possible security issues in the given smart contracts. A repeating code audit or incremental code audit is encouraged. Multiple audits with several auditors are recommended. Product owners are still required to have their own test cases and regular code review process. A threat intelligence system may help to discover or prevent a potential attack which can further reduce risk. Additionally, a bug bounty program for the community will help improve the security of products. Last but not least, Security is complicated! A strong smart contract does not guarantee your product is safe from all cybersecurity attacks.

Contracts overview

The contracts repository was shared through github, the tag names and commit revisions are attached below. The last tag we have audit is "refs/tags/audit-final".

Repo	Tags	Revision
sumer-project	refs/tags/audit	32cd5d87079c64e68efc8a2faf4f427137db3e62
	refs/tags/audit-fix1	0319c31c2703548484886492d8612f2325cdd541
	refs/tags/audit-fix2	bf952f2e726dfe151b3f65aa86b0e0ec96cd8993
	refs/tags/audit-fix3	f59a83e0cebd69c8130d16dbe7691ebe4d2d46e8
	refs/tags/audit2	dd885c4d607da0ea1d7b3d1fac4332752ef64c78
	refs/tags/audit2-fix1	1280aebcc45033eb7e8825d15c282d53fc3cf6c5
	refs/tags/audit2-fix2	9492ad9131fddf4fcec585cf286ab1d930e85287
	refs/tags/audit2-fix3	370d151c12287babc6fca82f8d37a3e9724ca1ab
	refs/tags/audit2-fix4	7216cc4329d8d0e837ded83c8df7e9c8e9325b72
	refs/tags/audit2-fix5	08a0e01502b5549a77bea13ff6ae7be1bee967d4
	refs/tags/audit2-fix6	f20ad32f2341f7a571fdaaa01c65812417fc73d5
	refs/tags/audit2-fix7	c268541360b81a0fe361f623054a75b104f67a64
	refs/tags/audit2-final	f950102bbf860a1eab4b9bac4e710db4f8dd3716
	refs/tags/audit2-final2	c3597c6a1f6749da06e69487a0d9f2734e561bb3
	refs/tags/v2-deployed	5c48ba95cdc3f5a1f2bd9eb512e03a8aabb91167
	refs/tags/audit-final	7f7754f238a13f79c7bacd3cc993d159454c394a
	refs/tags/audit-final-dev1	c7c86a37317e79cba09c3e0e9114c1e4528a22c9

The findings

Results

ID	Description	Severity	Product Impact	Status
Sumer-A-01	Arbitrary borrow and collateral for any account	Critical	Critical	Fixed
Sumer-A-02	The assetGroup index starts from 0	Medium	High	Fixed

Sumer-A-03	force to sync balance may lead to donation attack	Medium	Medium	Acked
Sumer-A-04	Unprotected privilege function	Critical	Critical	Fixed
Sumer-A-05	SortedBorrower on arbitrary token and borrower	High	High	Deprecated
Sumer-A-06	incorrect msg.sender after an external call	Medium	Medium	Fixed
Sumer-A-07	Native token address is inconsistent	Medium	Medium	Fixed
Sumer-A-08	Not following Checks-effects-interactions best practices	Medium	Medium	Fixed
Sumer-A-09	High gas used when updating the borrower	Medium	Medium	Deprecated
Sumer-A-10	Multiple Agreement Indexes might not work as expected	Medium	Medium	Fixed
Sumer-A-11	Timelock admin functions parameter checks	Low	Low	Fixed
Sumer-A-12	utilizationRate() may not be safe in WhitePaperInterestRateModel	High	Low	Fixed
Sumer-A-13	sendValue() does not have gas limits	Low	Low	Fixed
Sumer-A-14	New redeemFaceValue() function needs to verify deadline and chain ID	Low	Low	Fixed
Sumer-A-15	consumeValue() allowed to consume any CToken amount	Critical	Critical	Fixed
Sumer-A-16	createAgreement() can drain the pool	Critical	Critical	Fixed

Details

Sumer-A-01 [Critical] Arbitrary borrow and collateral for any account

In the contract CToken.sol, the "isCToken()" check in the function borrowAndDepositBack() can be bypassed. This allows anyone to add borrow/collateral on behalf of any account.

```
ftrace | funcSig
function borrowAndDepositBack(address borrower⬆, uint256 borrowAmount⬆) external nonReentrant returns (uint256) {
  // only allowed to be called from su token
  if (CToken(msg.sender).isCToken()) {
    revert NotSuToken();
  }
  // only cToken has this function
  if (!isCToken) {
    revert NotCToken();
  }
  return borrowAndDepositBackInternal(payable(borrower⬆), borrowAmount⬆);
}

/**
 * @notice Sender borrows assets from the protocol and deposit all of them back to the protocol
 * @param borrowAmount The amount of the underlying asset to borrow and deposit
 * @return uint 0=success, otherwise a failure (see ErrorReporter.sol for details)
 */
ftrace | funcSig
function borrowAndDepositBackInternal(address payable borrower⬆, uint256 borrowAmount⬆) internal returns (uint256) {
  accrueInterest();
  borrowFresh(borrower⬆, borrowAmount⬆, false);
  mintFresh(borrower⬆, borrowAmount⬆, false);
  return uint256(0);
}
}
```

Suggestion: Only allow suToken to call this function.

Update: Fixed

Sumer-A-04 [critical] Unprotected privileged function

In the contract Comptroller.sol, the privileged function cleanAssetGroup() does not have any protection which allows anyone to reset(delete) all assetGroups. The missing assetGroup could impact the user's liquidity check.

```
593  function cleanAssetGroup() external {
594    for (uint8 i = 0; i < _eqAssetGroups.length; i++) {
595      uint8 groupId = _eqAssetGroups[i].groupId;
596      delete assetGroupIdToIndex[groupId];
597    }
598
599    uint8 len = uint8(_eqAssetGroups.length);
600    for (uint8 i = 0; i < len; i++) {
601      _eqAssetGroups.pop();
602    }
603  }
```

Suggestion: Allows only admins to call this function.

Update: Only allows calls from DEFAULT_ADMIN_ROLE role.

Sumer-A-15 [Critical] consumeValue() allowed to consume any CToken amount

In the contract Timelock.sol, the onlyListedCToken() modifier for the function consumeValue() ensures the cToken parameter is a valid token. However, there is no permission check on who can call this function. It allows anyone to consume any amount of underlying tokens on any valid cToken.

```
58  function consumeValue(uint256 underlyAmount, address cToken) external onlyListedCToken(cToken) {
59    consumeValueInternal(underlyAmount, cToken);
60  }
61
```

Suggestion: Check msg.sender to ensure it is authorized.
Update: Only allows msg.sender is a valid CToken.

Sumer-A-16 [Critical] createAgreement() can drain the pool

In the contract Timelock.sol, the onlyListedCToken() modifier for the function createAgreement() ensures the cToken parameter is a valid token. However, there is no permission check on who could call this function. It allows anybody to create a timelock agreement for any amount of underlying tokens on any valid cToken.

```
136  function createAgreement(
137      TimeLockActionType actionType⬆,
138      address cToken⬆,
139      uint256 underlyAmount⬆,
140      address beneficiary⬆
141  ) external onlyListedCToken(cToken⬆) returns (uint256) {
142      require(beneficiary⬆ != address(0), 'Beneficiary cant be zero address');
143      uint256 underlyBalance;
144      address underlying = ICToken(cToken⬆).underlying();
145      if (underlying == address(0)) {
146          underlyBalance = address(this).balance;
147      } else {
148          underlyBalance = IERC20(underlying).balanceOf(address(this));
149      }
150      require(underlyBalance >= balances[underlying] + underlyAmount⬆, 'balance error');
151      balances[underlying] = underlyBalance;
152
153      uint256 agreementId = agreementCount++;
154      uint48 timestamp = uint48(block.timestamp);
```

Suggestion: Check msg.sender to ensure it is authorized.
Update: Only allows msg.sender is a valid CToken.

Sumer-A-02 [High] The assetGroup index starts from 0

In the contract Comptroller.sol, an index(assetGroupIdToIndex) is used to manage the asset group. However, the valid index starts from '0', which means an non-existent groupId will be valid. For example, an arbitrary groupId in the function removeAssetGroup() will cause the group index 0 to be removed.

```
    ftrace | funcSig
579  function removeAssetGroup(uint8 groupId⬆) external onlyRole(DEFAULT_ADMIN_ROLE) returns (uint256) {
580      uint8 length = uint8(_eqAssetGroups.length);
581      uint8 lastGroupId = _eqAssetGroups[length - 1].groupId;
582      uint8 index = assetGroupIdToIndex[groupId⬆];
583
584      _eqAssetGroups[index] = _eqAssetGroups[length - 1];
585      assetGroupIdToIndex[lastGroupId] = index;
586      _eqAssetGroups.pop();
587      delete assetGroupIdToIndex[groupId⬆];
588
589      emit RemoveAssetGroup(groupId⬆, length);
590      return uint256(0);
591  }
592
```

Suggestion: Start the group index from 1, use 0 to differentiate an unexistant groupId.

Update: Fixed

Sumer-A-05 [High] SortedBorrower on arbitrary token and borrower

In the contract Comptroller.sol, a couple of functions are intended to be protected by the "onlyCToken()" modifier to prevent public calls. However, the check is not robust enough. It allows anyone to call redemptionManager.updateSortedBorrows() on an arbitrary Token and borrower address, which could corrupt the sortedBorrows storage. Thus, could be used to target redemption.

```
660  modifier onlyCToken() {
661    require(isContract(msg.sender), 'only ctoken');
662    ICToken(msg.sender).isCToken();
663    _;
664  }
665
```

```
1282 function borrowVerify(address cToken, address borrower, uint256
       borrowAmount) external onlyCToken {
1283   // Shh - currently unused
1284   cToken;
1285   borrower;
1286   borrowAmount;
1287   redemptionManager.updateSortedBorrows(cToken, borrower);
1288 }
1289
```

Suggestion: Update onlyCToken modifier to ensure it has a strong and robust check.

Update: Took out the code and used pass in providers for redemption.

Sumer-A-12 [High] utilizationRate() may not be safe in WhitePaperInterestRateModel

In the contract WhitePaperInterestRateModel.sol, the function utilizationRate() may not be safe if the cash is less than the reserves. This contract does not appear to be used anywhere, but please be aware of the potential vulnerability.

```
55  function utilizationRate(
56    uint256 cash⬆,
57    uint256 borrows⬆,
58    uint256 reserves⬆
59  ) public pure returns (uint256) {
60    // Utilization rate is 0 when there are no borrows
61    if (borrows⬆ == 0) {
62      return 0;
63    }
64
65    return borrows⬆.mul(1e18).div(cash⬆.add(borrows⬆).sub(reserves⬆));
66  }
```

Suggestion: Don't use this code without updating interest rate model
Update: Fixed

Sumer-A-08 [Medium] Not following Checks-effects-interactions best practice

To prevent potential re-entrance issues, developers must follow the Checks-effects-interactions pattern. There are several places in which the state will change after an external call to the user managed address.

```
172  function doTransferOut(address payable to⬆, uint256 amount⬆) internal override {
173    /* Send the Ether, with minimal gas and revert on failure */
174    // to.transfer(amount);
175    (bool success, ) = to⬆.call{gas: 5300, value: amount⬆}('');
176    require(success, 'unable to send value, recipient may have reverted');
177    underlyingBalance -= amount⬆;
178  }
```

```
156  function claim(uint256[] calldata agreementIndexes⬆) external nonReentrant {
157    uint256[] memory sorted = sort_array(agreementIndexes⬆);
158    require(!frozen, 'timeLock frozen');
159
160    for (uint256 i = 0; i < agreementIndexes⬆.length; i++) {
161      Agreement memory agreement = _validateAndDeleteAgreement(msg.sender, sorted[i]);
162      if (agreement.underlying == address(1)) {
163        // payable(agreement.beneficiary).transfer(agreement.amount);
164        Address.sendValue(payable(msg.sender), agreement.amount);
165      } else {
166        IERC20(agreement.underlying).safeTransfer(msg.sender, agreement.amount);
167      }
168      underlyingDetail[agreement.underlying].totalBalance -= agreement.amount;
169    }
170  }
```

```
261  function doTransferOut(address payable to⬆, uint256 amount⬆) internal virtual overr
262    ICToken token = ICToken(underlying);
263    token.transfer(to⬆, amount⬆);
264    underlyingBalance -= amount⬆;
265
266    bool success;
```

Suggestion: Update state before making the call.

Update: Fixed.

Sumer-A-03 [Medium] force to sync balance may lead to donation attack

In the CToken.sol, the function _syncUnderlyingBalance() will force updating the cash(underlyingBalance) with the existing balance in the token contract. Depending on when you call this function, it may lead to a donation attack which is a known existing issue for compound protocols.

Suggestion: Please be cautious when using this call.

Update: "Yes, totally understand. We called it only to fix the bug we had previously that cEther did not track the underlying balance properly"

Sumer-A-06 [Medium]incorrect msg.sender after an external call

In the contract Comptroller.sol, the function redeemFaceValueWithPermit() calls this.redeemFaceValue(). Because of the external call, the msg.sender in

function redeemFaceValue() will be changed to the address of Comptroller. This is no longer the original EOA address which calls redeemFaceValueWithPermit().

```
894  function redeemFaceValueWithPermit(
895    address suToken⬆,
896    uint256 amount⬆,
897    uint256 deadline⬆,
898    bytes memory signature⬆
899  ) external {
900    address underlying = ICToken(suToken⬆).underlying();
901    IEIP712(underlying).permit(msg.sender, suToken⬆, amount⬆, deadline⬆, signature⬆)
902    return this.redeemFaceValue(suToken⬆, amount⬆);
903  }
```

Suggestion: Update function redeemFaceValue() to 'public' and remove the external call by using 'this.'

Update: Fixed

Sumer-A-07 [Medium] Native token address is inconsistent

In the CEther contract, createAgreement() will use the current underlying address which is address(0). But in the timelock contract, the claim() function uses address(1) as the native token. This inconsistency may cause unexpected side effects.

Suggestion: Use the same address for the CEther underlying address.

Update: Fixed

Sumer-A-09 [Medium] High gas used when updating the borrower

The operation on the SortedBorrower storage is quite expensive. For example, the function updateSortedBorrows() will loop the list and find the closest address, then update prevId and nextId. It will load the storage from 'cold' and the gas cost is 2,100. For example, if there are 500 borrowers and need to insert to the end, the minimum gas cost would be 500 * 2 * 2100 = 2,100,000.

Suggestion: Please review the business logic to determine if this cost is necessary.

Update: Deprecated

Sumer-A-10 [Medium] Multiple Agreement Indexes might not work as expected

In the contract Timelock.sol, the function claim() takes a list of indexes for the agreement claim. But the agreement positioning might be changed during the function _validateAndDeleteAgreement() as the last position gets moved to the claimed one. If the index list contains an index which is the last one claimed then this no longer works as expected.

```
135  function _validateAndDeleteAgreement(
136    address beneficiary⬆,
137    uint256 agreementIndex⬆
138  ) internal returns (Agreement memory) {
139    uint256 length = uint256(userAgreements[beneficiary⬆].length);
140    require(agreementIndex⬆ < length, 'agreement index out of bound');
141    Agreement memory agreement = userAgreements[beneficiary⬆][agreementIndex⬆];
142    require(block.timestamp >= agreement.releaseTime, 'release time not reached');
143    require(!agreement.isFrozen, 'agreement frozen');
144
145    // Move the last element to the deleted spot.
146    // Remove the last element.
147    delete userAgreements[beneficiary⬆][agreementIndex⬆];
148    userAgreements[beneficiary⬆][agreementIndex⬆] = userAgreements[beneficiary⬆][userAgreements[beneficiary⬆].length -
149    userAgreements[beneficiary⬆].pop();
150
151    emit AgreementClaimed(beneficiary⬆, agreementIndex⬆, agreement.underlying, agreement.actionType, agreement.amount);
152
153    return agreement;
154  }
155
```

Suggestion: Don't support multiple agreements or combine together by using the address.

Update: Fixed

Sumer-A-11 [Low] Timelock admin functions parameter checks

In the contract Timelock.sol, there are multiple privilege functions(onlyAdmin) that do not have parameter value checks. Adding a value check will prevent future damage by incidental calls. For example, the lockDuration can be set to 0 in the setLockDuration function, if this is never intended to be set to said value, it should be checked.

```
80  function setLockDuration(address underlying⬆, uint48 lockDuration⬆) external onlyAdmin {
81    underlyingDetail[underlying⬆].lockDuration = lockDuration⬆;
82  }
83
```

Suggestion: Please double check the value range for important functions.

Update: Use a new function isAgreementMature() to check the lock time.

Sumer-A-13 [Low] sendValue() does not have gas limits

In the contract Timelock.sol, the function claim() will call the Address.sendValue() function to send native tokens. There are no gas limits which can introduce the potential re-entry vulnerabilities.

```
ftrace | funcSig
function claim(uint256[] calldata agreementIds⬆) external nonReentrant {
  require(!frozen, 'TimeLock is frozen');

  for (uint256 index = 0; index < agreementIds⬆.length; index++) {
    Agreement memory agreement = _validateAndDeleteAgreement(agreementIds⬆[index]);
    address underlying = ICToken(agreement.cToken).underlying();
    if (underlying == address(0)) {
      // payable(agreement.beneficiary).transfer(agreement.amount);
      Address.sendValue(payable(agreement.beneficiary), agreement.underlyAmount);
    } else {
      IERC20(underlying).safeTransfer(agreement.beneficiary, agreement.underlyAmount);
    }
    balances[underlying] -= agreement.underlyAmount;
  }
}
```

Suggestion: Add gas limits, similar to other places in the code base.
Update: Fixed

Sumer-A-14 [Low] New redeemFaceValue() function needs to verify deadline and chain ID

In the contract RedemptionManager.sol, the new function redeemFaceValue() will verify the pass in providers list. However, it does not check if the deadline parameter is expired or not. Furthermore, the chain ID should be part of signature verification data so that users cannot send to signing providers across chains.

```
ftrace | funcSig
function redeemFaceValue(
  address csuToken,
  uint256 amount,
  address[] memory providers,
  uint256 deadline,
  bytes memory signature
) public {
  if (ICToken(csuToken).isCToken() || !comptroller.isListed(csuToken)) {
    revert InvalidSuToken();
  }

  if (signature.length != 65) {
    revert InvalidSignatureLength();
  }
  bytes32 hash = keccak256(abi.encodePacked(deadline, providers));
```

Suggestion: Check deadline and add chain ID into hash function.
Update: Fixed

Summary

Ancilia team has performed both an automated and manual code audit on the Sumer smart contracts mentioned above. All issues have been shared with the Sumer.money team through a telegram channel before this report. Overall, 4 critical, 3 high, 6 medium and 3 low impact issues have been discovered through this audit.
Sumer.money team reacted pretty quickly and fixed all the issues. Ancilia team verified and confirmed the fixes are in the github.