QuillAudits

Audit Report
July, 2021

SYMMETRIC

Contents

Overview

Symmetric is built to drive mass adoption of DeFi. With a focus on simplicity, it is designed with newcomers in mind, removing technical barriers by providing seamless connectivity. In addition by leveraging lower and predictable gas prices on networks like xDai & Celo, Symmetric makes DeFi more cost effective than other networks. Its unique risk fund provides a protective layer to users and liquidity providers of the platform, reducing risk of loss due to malicious attacks.

Symmetric makes DeFi accessible to everyone, regardless of the size of their portfolio, technical knowledge or risk appetite.

Scope of Audit

CentToken: ERC20 Token with snapshot mechanism
Commit: fb315312ff1544e3af47d8d2246bcc0c75eadf59
Fixed In: e06c4fd50d021607c8c73b7bda3cdf16647e0399

SymmCoin: ERC20 Token with snapshot mechanism
Commit: 9d6ba6f3f48f99cc604db4901e4d39eecef55d87
Fixed In: b81bb791e303d88051839e1d8047dc8b634a44f4

CToken: ERC20 Token used for accounting purpose
Commit: 0a10c72ed57c372d7861c02a9c58384b89d58de6

PoolState: Helper contract to efficiently return information on pools rather than have multiple separate calls
Commit: 4f3e44edf968b1b224d06b9dcdc444f9ef3fe46b

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- EIP712 Structure
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Mythril, Slither, SmartCheck, Surya, Solhint.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	9	0
Closed	0	1	0	0

Test Cases

- Only **SNAPSHOT_ROLE** should be able to take a Snapshot of the state.
  -- > PASS

- Only **MINTER_ROLE** should be able to Mint tokens.
  --> PASS

- Shouldn’t Mint to **Zero Address**.
  --> PASS

- Shouldn’t be able to **Burn Tokens** more than the available balance
  --> PASS

- **Spender** shouldn’t be able to **burn** tokens more than the **allowance** from **owner.**
  --> PASS

- Shouldn’t be able to **Transfer Tokens** more than the available balance
  --> PASS

- **Spender** shouldn’t be able to **transfer** tokens more than the **allowance** from **owner.**
  --> PASS

- Only **DEFAULT_ADMIN_ROLE** can grant or revoke role.
  --> PASS

- **staticcall** should return a **status 0**, if the function called at address **addr** is doing some **state modifying operations**, or if the function doesn’t exist at the specified address, and hence the **result** returned by function **getUint()** should be 0.
  --> PASS

- Function **getPoolInfo()** should return for all the supplied Pools: **Swap Fee** for a **Pool Address** and the **balances** of all the **token addresses** of the pool.
  --> PASS

Suggestion

PoolState.sol

[#L26-48] function getPoolInfo():

A **require** check for the **length** parameter can be added: If the supplied length is a number that is equal or greater than the length of **pools** 2D array, or in other words a number enough to hold all the values that is the **swap fee** of the **pool** and **balance** of **each token** address, then the function will work as intended. But if it supplied less than that(meaning a number less than the total values), it will result in Invalid Opcode.

Issues Found

High severity issues

No issues were found.

Medium severity issues

CentToken.sol & symmCoin.sol

1. [FIXED] **[71-80] hashStruct**

   The order of concatenation of member values doesn't match with the **PERMIT_TYPEHASH**

   Recommendation
   Consider using draft-ERC20Permit extension from openzeppelin for Permit function

References

- https://eips.ethereum.org/EIPS/eip-712
- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/draft-ERC20Permit.sol

Low level severity issues

2. **Older Versions of the solidity compiler have been used in all the contracts:** Use newer versions so as to avoid bugs introduced in the older compilers.

3. **Multiple Pragma Directives have been used:** Use one solidity compiler.

4. **ERC20 approve() race:**
   The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice.

   Reference:

   - https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit
   - https://medium.com/mycrypto/bad-actors-abusing-erc20-approval-to-steal-your-tokens-c0407b7f7c7c
   - https://eips.ethereum.org/EIPS/eip-20

CToken.sol

5. Contract BColor should be marked as abstract

```
contract BColor {
    function getColor()
        external view
        returns (bytes32);
}

contract BBronze is BColor {
    function getColor()
        external view
        returns (bytes32) {
            return bytes32("BRONZE");
        }
}
```

6. **Missing Virtual/Override:** User Virtual & Override keywords while overriding functions

```
15  abstract contract BColor {
16      function getColor()
17          external view
18          returns (bytes32);
19  }
20
21  contract BBronze is BColor {
22      function getColor()
23          external view
24          returns (bytes32) {
25              return bytes32("BRONZE");
26          }
27  }
```

```
271
272     function allowance(address src, address dst) external view returns (uint) {
273         return _allowance[src][dst];
274     }
275
276     function balanceOf(address whom) external view returns (uint) {
277         return _balance[whom];
278     }
279
280     function totalSupply() public view returns (uint) {
281         return _totalSupply;
282     }
283
284     function approve(address dst, uint amt) external returns (bool) {
285         _allowance[msg.sender][dst] = amt;
286         emit Approval(msg.sender, dst, amt);
287         return true;
288     }
289
290     function increaseApproval(address dst, uint amt) external returns (bool) {
291         _allowance[msg.sender][dst] = badd(_allowance[msg.sender][dst], amt);
292         emit Approval(msg.sender, dst, _allowance[msg.sender][dst]);
293         return true;
294     }
295
296     function decreaseApproval(address dst, uint amt) external returns (bool) {
297         uint oldValue = _allowance[msg.sender][dst];
298         if (amt > oldValue) {
299             _allowance[msg.sender][dst] = 0;
300         } else {
301             _allowance[msg.sender][dst] = bsub(oldValue, amt);
302         }
303         emit Approval(msg.sender, dst, _allowance[msg.sender][dst]);
304         return true;
305     }
306
307     function transfer(address dst, uint amt) external returns (bool) {
308         _move(msg.sender, dst, amt);
309         return true;
310     }
311
312     function transferFrom(address src, address dst, uint amt) external returns (bool) {
313         require(msg.sender == src || amt <= _allowance[src][msg.sender], "ERR_CTOKEN_BAD_CALLER");
314         _move(src, dst, amt);
```

7. Multiple declarations of Events **Approval** and **Transfer** found as the **CToken** contract inherits from **IERC20** and **CTokenBase.** The **CToken** contract finds the event declaration twice.

```
201 interface IERC20 {
202     event Approval(address indexed src, address indexed dst, uint amt);
203     event Transfer(address indexed src, address indexed dst, uint amt);
204
205     function totalSupply() external view returns (uint);
206     function balanceOf(address whom) external view returns (uint);
207     function allowance(address src, address dst) external view returns (uint);
208
209     function approve(address dst, uint amt) external returns (bool);
210     function transfer(address dst, uint amt) external returns (bool);
211     function transferFrom(
212         address src, address dst, uint amt
213     ) external returns (bool);
214 }
215
216 contract CTokenBase is BNum {
217
218     mapping(address => uint)                   internal _balance;
219     mapping(address => mapping(address=>uint)) internal _allowance;
220     uint internal _totalSupply;
221
222     event Approval(address indexed src, address indexed dst, uint amt);
223     event Transfer(address indexed src, address indexed dst, uint amt);
224
225     function _mint(uint amt) internal {
226         _balance[address(this)] = badd(_balance[address(this)], amt);
227         _totalSupply = badd(_totalSupply, amt);
228         emit Transfer(address(0), address(this), amt);
229     }
230
```

8. Explicit type conversion not allowed from **"int_const -1"** to **"uint256"**: **Use type(uint256).max**

```
310 ▾    function transferFrom(address src, address dst, uint amt) external override returns (bool) {
311          require(msg.sender == src || amt <= _allowance[src][msg.sender], "ERR_CTOKEN_BAD_CALLER");
312          _move(src, dst, amt);
313 ▾        if (msg.sender != src && _allowance[src][msg.sender] != uint256(-1)) {
314              _allowance[src][msg.sender] = bsub(_allowance[src][msg.sender], amt);
315              emit Approval(msg.sender, dst, _allowance[src][msg.sender]);
316          }
317          return true;
```

9. **Missing Minting/Burning Implementations:** The contract has **Minting** and **Burning** internal functions but no public/external functions to accompany them. Also, the contract doesn't mint the initial tokens.

```
function _mint(uint amt) internal {
    _balance[address(this)] = badd(_balance[address(this)], amt);
    _totalSupply = badd(_totalSupply, amt);
    emit Transfer(address(0), address(this), amt);
}

function _burn(uint amt) internal {
    require(_balance[address(this)] >= amt, "ERR_INSUFFICIENT_BAL");
    _balance[address(this)] = bsub(_balance[address(this)], amt);
    _totalSupply = bsub(_totalSupply, amt);
    emit Transfer(address(this), address(0), amt);
}
```

CentToken.sol & symmCoin.sol

10. The contract doesn't take snapshots of the state **(Balance, TotalSupply)** automatically while minting or token transfers.

Informational

No issues were found.

Gas Optimization

Public functions that are never called by the contract should be declared external to save gas.

CToken.sol

```
name() should be declared external:
        - CToken.name() (CToken.sol#258-260)
symbol() should be declared external:
        - CToken.symbol() (CToken.sol#262-264)
decimals() should be declared external:
        - CToken.decimals() (CToken.sol#266-268)
totalSupply() should be declared external:
        - CToken.totalSupply() (CToken.sol#278-280)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Automated Testing

Slither

Slither didn’t detect any high severity issues.

Mythril

Mythril didn’t detect any high severity issues.

Smartcheck

Smartcheck didn’t detect any high severity issues.

Solhint

```
CentFinance/CentToken.sol
  15:2  error  Line length must be no more than 120 but current length is 163  max-line-length
  21:2  error  Line length must be no more than 120 but current length is 141  max-line-length
  68:2  error  Line length must be no more than 120 but current length is 124  max-line-length

CentFinance/symmcoin.sol
  15:2  error  Line length must be no more than 120 but current length is 163  max-line-length
  21:2  error  Line length must be no more than 120 but current length is 141  max-line-length
  68:2  error  Line length must be no more than 120 but current length is 124  max-line-length

✖ 6 problems (6 errors, 0 warnings)
```

Disclaimer

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

Closing Summary

Some issues of medium and low severity have been reported during the audit. The medium severity issue has been fixed in the new commit. No critical or high severity issues have been reported. Some suggestions have also been made to improve the code quality and gas optimisation.

SYMMETRIC

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com