Tornado Cash - MerkleTreeWith History Audit

By John Nguyen (jooohn.eth)

General Info

Resources:
Github repo which consists of the project's core smart-contracts, tests, user interface and documentation.
Project author:
Tornado Cash Community
Audit author:
John Nguyen (jooohn.eth)

Table of contents

Summary

Tornado Cash is a non-custodial Ethereum and ERC20 privacy solution based on zkSNARKs. It improves transaction privacy by breaking the on-chain link between the recipient and destination addresses. It uses a smart contract that accepts ETH deposits that can be withdrawn by a different address. Whenever ETH is withdrawn by the new address, there is no way to link the withdrawal to the deposit, ensuring complete privacy.

The MerkleTreeWithHistory.sol contract was reviewed.

The project was reviewed manually and with the help of tools.

Scope:

Github Repo
Commit

The commit reviewed was 5cb9d6017878f4d4237fcc42c978fb34a2192b0c. The review covered the repository at the specific commit and focused on the contracts directory.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Access control provided where needed

Compiler	Okay	The use of Solidity 0.7.0 might not be the best, since newer versions provide additional security checks
Libraries	Good	No libraries were used, meaning less dependency of external contracts
Documentation	Okay	Natspec comments were provided, but not all functions were commented.
Testing	Good	All tests passed with a good percentage of code coverage.
Decentralization	Good	No external party access provided.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas Savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

No Critical, High, Medium or Low Findings

Informational Findings

1. **Unused function**

- Proof of concept: _insert() function declared but never used (contracts/MerkleTreeWithHistory.sol#68)

```
 function _insert(bytes32 _leaf) internal returns (uint32 index) {
   uint32 _nextIndex = nextIndex;
   require(_nextIndex != uint32(2)**levels, "Merkle tree is full. No more leaves can
be added");
   uint32 currentIndex = _nextIndex;
   bytes32 currentLevelHash = _leaf;
   bytes32 left;
   bytes32 right;

   for (uint32 i = 0; i < levels; i++) {
     if (currentIndex % 2 == 0) {
       left = currentLevelHash;
       right = zeros(i);
       filledSubtrees[i] = currentLevelHash;
     } else {
       left = filledSubtrees[i];
       right = currentLevelHash;
     }
     currentLevelHash = hashLeftRight(hasher, left, right);
     currentIndex /= 2;
   }

   uint32 newRootIndex = (currentRootIndex + 1) % ROOT_HISTORY_SIZE;
   currentRootIndex = newRootIndex;
   roots[newRootIndex] = currentLevelHash;
   nextIndex = _nextIndex + 1;
   return _nextIndex;
 }
```

- Impact: unused function are unnecessary and might be confusing
- Recommendation: remove the unused function

2. **Solidity version not recommended for deployment**

- Proof of concept: solidity and solc version 0.7.0 are not recommended for deployment(contracts/MerkleTreeWithHistory.sol#13)

```
pragma solidity ^0.7.0;
```

- Impact: newer versions provide additional security checks that are absent in 0.7.0
- Recommendation: deploy on newer version of solidity

3. **Functions could be declared external**

- Proof of concept: isKnownRoot() and getLastRoot() are never called by the contract itself, could be declared external(contracts/MerkleTreeWithHistory.sol#99-115, #120-122)

```
 function isKnownRoot(bytes32 _root) public view returns (bool) {
  if (_root == 0) {
    return false;
  }
  uint32 _currentRootIndex = currentRootIndex;
  uint32 i = _currentRootIndex;
  do {
    if (_root == roots[i]) {
      return true;
    }
    if (i == 0) {
      i = ROOT_HISTORY_SIZE;
    }
    i--;
  } while (i != _currentRootIndex);
  return false;
}

 function getLastRoot() public view returns (bytes32) {
  return roots[currentRootIndex];
}
```

- Impact: extra gas payed
- Recommendation: modify function visibility to external

4. **Solidity naming conventions are not followed: function name should be in mixedCase**

- Proof of concept: MiMcSponge() function not in mixedCase(contracts/MerkleTreeWithHistory.sol#16)

```
  function MiMCSponge(uint256 in_xL, uint256 in_xR) external pure returns
(uint256 xL, uint256 xR);
```

- Impact: might be confusing for contract readers and auditors, generally language naming conventions should always be used
- Recommendation: rename the function according to Solidity Naming Conventions

5. **Solidity naming conventions are not followed: function parameters should be in mixedCase**

- Proof of concept: MiMCSponge(in_xL, in_xR), hashLeftRight(_hasher, _left, _right), isKnownRoot(_root) function parameters are not in mixedCase(contracts/MerkleTreeWithHistory.sol#16,#53,#99)

```
  function MiMCSponge(uint256 in_xL, uint256 in_xR) external pure returns
(uint256 xL, uint256 xR);

 function hashLeftRight(
   IHasher _hasher,
   bytes32 _left,
   bytes32 _right
 ) public pure returns (bytes32) {}

  function isKnownRoot(bytes32 _root) public view returns (bool) {}
```

- Impact: might be confusing for contract readers and auditors, generally language naming conventions should always be used

- Recommendation: rename the function according to Solidity Naming Conventions

Final Remarks

After reviewing the MerkleTreeWithHistory smart contract, no critical, high, medium or low vulnerabilities were found, a few informational findings that would benefit if they were modified. Unit tests were reviewed - no anomalies found, the tests were accurate.