ethereum is not bitcoin

“The key component is this idea of a Turing-complete blockchain”

--Vitalik Buterin
smart contracts
literally a billion reasons

No DAO funds at risk following the Ethereum smart contract ‘recursive call’ bug discovery

Our team is blessed to have Dr. Christian Reitwießner, Father of Solidity, as its Advisor. During the early development of the DAO Framework 1.1 and thanks to his guidance we were made aware of a generic vulnerability common to all Ethereum smart contracts. We promptly circumvented this so-called “recursive call vulnerability” or “race to empty” from the DAO Framework 1.1 as can be seen on line 580:
30 million reasons will do

PARITY TECHNOLOGIES PRESENTS

PARITY

ETHEREUM BROWSER

Parity Technologies is proud to present our powerful new Parity Browser. Integrated directly into your Web browser, Parity is the fastest and most secure way of interacting with the Ethereum network.
caveats
solidity
dev tools

• .sol files > bytecode > blockchain
• Atom with plugins:
  • language-ethereum
  • etheratom
• Remix: browser based
contract MyContract {
    uint balance;

    function MyContract() {
        Mint(1000000);
    }

    function Mint(uint amount) internal {
        balance = amount;
    }

    function withdraw() {
        msg.sender.transfer(balance);
    }

    function GetBalance() constant returns(uint) {
        return balance;
    }
}
oyente
basic methodology

• Interview devs
• Review .sol file
• Try compiling
• Dissect code flow—optional solgraph
• Run oyente (cross fingers)
• Manually verify 3/4 vuln yay/nays
• Manually check for following vulns...
contract ReEntrancy {

    mapping (address => uint) private expendableTokens;

    function stealTokens() public {
        uint amountToLose = expendableTokens[msg.sender];
        if (!(msg.sender.call.value(amountToLose)()) == throw; }
        expendableTokens[msg.sender] = 0;
    }
}
contract Entrance {

    mapping (address => uint) private expendableTokens;

    function stealTokens() public {
        uint amountToLose = expendableTokens[msg.sender];
        expendableTokens[msg.sender] = 0;
        if (!(msg.sender.call.value(amountToLose)())) { throw; }
    }
}
reentrancy (and irony) in the dao code

```solidity
// Burn DAO Tokens
Transfer(msg.sender, 0, balances[msg.sender]);
withdrawRewardFor(msg.sender); // be nice, and get his rewards
totalSupply -= balances[msg.sender];
balances[msg.sender] = 0;
paidOut[msg.sender] = 0;
return true;
```
default public – parity wallet hack

```solidity
contract WalletLibrary is WalletEvents {

    // constructor is given number of sigs required to do protected "onlymanyowners" transactions
    // as well as the selection of addresses capable of confirming them.

    function initMultiowned(address[] _owners, uint _required) {
        m_numOwners = _owners.length + 1;
        m_owners[1] = uint(msg.sender);
        m_ownerIndex[uint(msg.sender)] = 1;
    }

    // constructor - stores initial daily limit and records the present day's index.

    function initDaylimit(uint _limit) {
    }

    // constructor - just pass on the owner array to the multiowned and
    // the limit to daylimit

    function initWallet(address[] _owners, uint _required, uint _daylimit) {
    }
}
```
initWallet

Transaction Information

TxHash: 0x9dfe326a0382a3719e27be49a3a9657d23fe6e3d93ade4be383d6f754c
Block Height: 4043800 (29739 block confirmations)
TimeStamp: 6 days 5 hrs ago (Jul-19-2017 12:18:15 PM +UTC)
From: 0xb37e4751c297a8121e79c32e6628ed16b44332 (MultiSigExploit-Hacker)
To: Contract 0xbeec591de75b4859a38a524773438122d0b4cd7e
Value: 0 Ether (0.00)
Gas Limit: 82703
Gas Price: 0.00004021 Ether (21 Gwei)
Gas Used By Txn: 56889
Actual Tx Cost/Fee: 0.001403619 Ether (80.29)
Cumulative Gas Used: 1283734
Nonce: 5
Input Data:

```plaintext
Function: initWallet(address[] _owners, uint256 _required,
uint256 _dayLimit) ***
MethodID: 0xe48d7feb
[0]: 0x0000000000000000000000000000000000000000000000000000000000000000
[1]: 0x0000000000000000000000000000000000000000000000000000000000000000
```
execute

Transaction Information

TxHash: 0x8ef10fe51705539b68e4c0d44448a2a56b87221325f6265d6183633a7be7c
Block Height: 4048802 (28738 block confirmations)
Time Stamp: 6 days 5 hrs ago (Jul-19-2017 12:19:36 PM -UTC)
From: 0xb3764761e297d6f1216793c32ad66825cdddb4d32 (MultisigEther-Hacker)
To: 0x59c591de75eb990e3b5a52b673420022d0b6c0d7e
Value: 0 Ether ($0.00)
Gas Limit: 78925
Gas Price: 0.00000021 Ether (21 Gwei)
Gas Used By Tx: 59433
Actual Tx Cost/Eur: 0.001227093 Ether ($0.25)
Cumulative Gas Used: 1821381
Nonce: 6

Input Data:

Function: execute(address to, uint256 value, bytes data)

MethodID: 0x62d2716
(0): 06000000000000000000000000000000000000000000000000000000003764761e297d6f1216793c32ad66825cdddb4d32
(1): 060000000000000000000000000000000000000000000000000000000001677796ee0304146000

Convert To Ascii
unchecked send in king of the ether

```solidity
uint compensation = valuePaid - wizardCommission;
if (currentMonarch.etherAddress != wizardAddress) {
    currentMonarch.etherAddress.send(compensation);
} else {
    // When the throne is vacant, the fee accumulates for the wizard.
}
```
unchecked send

```java
if (kingOfLosingDone && !(compensationSent)) {
    monarch.send(500);
    compensationSent = True;
}
```

```java
if (kingOfLosingDone && !(compensationSent)) {
    if (monarch.send(500))
        compensationSent = True;
    else throw;
}
```
gas limits
contract SendContract {
    address public richest;
    uint public mostSent;

    function SendContract() payable {
        richest = msg.sender;
        mostSent = msg.value;
    }

    function becomeRichest() payable returns (bool) {
        if (msg.value > mostSent) {
            richest.transfer(msg.value);
            richest = msg.sender;
            mostSent = msg.value;
            return true;
        } else {
            return false;
        }
    }
}
withdawn not sent

```
contract WithdrawalContract {
    address public richest;
    uint public mostSent;

    mapping (address => uint) pendingWithdrawals;

    function WithdrawalContract() payable {
        richest = msg.sender;
        mostSent = msg.value;
    }

    function becomeRichest() payable returns (bool) {
        if (msg.value > mostSent) {
            pendingWithdrawals[richest] += msg.value;
            richest = msg.sender;
            mostSent = msg.value;
            return true;
        } else {
            return false;
        }
    }

    function withdraw() {
        uint amount = pendingWithdrawals[msg.sender];
        pendingWithdrawals[msg.sender] = 0;
        msg.sender.transfer(amount);
    }
}
```
encryption
transaction-ordering dependence

```solidity
contract Puzzle{
    address public owner;
    bool public locked;
    uint public reward;
    bytes32 public diff;
    bytes public solution;

    function Puzzle(){
        owner = msg.sender;
        reward = msg.value;
        locked = false;
        diff = bytes32(11111);
    }

    function (){  
        if (msg.sender == owner){
            if (locked) throw;
            owner.send(reward);
            reward = msg.value;
        }
        else
            if (msg.data.length > 0){
                if (locked) throw;
                if (sha256(msg.data) < diff){
                    msg.sender.send(reward);
                    solution = msg.data;
                    locked = true;
                }
            }
    
```
call-stack depth limit

Announcement of imminent hard fork for EIP150 gas cost changes:

During the last couple of weeks, the Ethereum network has been the target of a sustained attack. The attacker(s) have been very crafty in locating vulnerabilities in the client implementations as...
variable or function ambiguity

```
1  Player[] public persons;
2  
3  uint public payoutCursor_Id_ = 0;
4  uint public balance = 0;
5  
6  address public owner;
7  
8  uint public payoutCursor_Id = 0;
9  ...
10  while (balance > persons[payoutCursor_Id_].deposit / 100 * 115) {
11  uint MultipliedPayout = persons[payoutCursor_Id_].deposit / 100 * 115;
12  persons[payoutCursor_Id].etherAddress.send(MultipliedPayout);
13  balance -= MultipliedPayout;
14  payoutCursor_Id_++;
```
input validation

throw -- being deprecated

require (condition) – check external conditions (invalid inputs or errors in external components)

assert (condition) – internal errors
odds and ends

- Timestamp dependence
- Business logic flaws
- Separating public/private data
get involved
dox me ... or just keep in touch

@konstanthacker

konstantinos.karagiannis@bt.com