Smart Contract Security
Assessing Solidity smart contracts
About Me

• Evangelos Deirmentzoglou
• Security Consultant
• Smart contract audits

• Nmap/Ncrack contributor
• Certs: OSCE, OSCP, OSWP
• Blockchain Basics
• Front Running
• Reentrancy
• External Calls
• Integer Overflow
• tx.origin
• Gas limit
Ethereum Gas
• Introduced to reduce abuse
• transaction cost (EVM operation)
• gas price (in eth)

Miner receives estimated gas spent* gas price
Remainder of gas cost is returned to the sender
Blockchain Transactions

Miner → Transaction Pool → Blockchain
Frontrunning

- aka Timing attacks
- Affecting: First In First Out Contracts
- Miner frontrunning
- User frontrunning
Examples: DEX, Bancor, random ICOs

```python
function freeEth(){
    msg.sender.transfer(1000 ether);
}
```

Priority Pricing Market for Gas Slots

- Price percentile (since block ~5M)
- Gas Price (gwei)

Reference: https://twitter.com/phildaian/status/1032679149960355840
- Set upper gas limit
- Set a fixed gas limit
- tx.gasprice
- gasleft()

```solidity
function () payable {
    require(tx.gasprice <= max_gas_price);
    require(gasleft() <= max_gas_limit);
    //...
}
```
Commit/Reveal approach

1. Commit
   - Generate a unique hash for your vote
   - Distribute your hash to the public
   - Wait for all votes to be committed….

2. Reveal
   - Distribute your vote to the public
   - The public verifies that your vote & hash match
   - Count votes and declare winner!

Reference: https://karl.tech/learning-solidity-part-2-voting/
Payable functions

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

function () payable {
    balances[msg.sender] += msg.value;
}
```
150$ mil stolen
DAO & numerous other contracts
function deposit() public payable {
    balances[msg.sender] += msg.value;
}

function withdraw(uint256 _withdraw) public payable {
    require(balances[msg.sender] >= _withdraw);
    require(msg.sender.call.value(_withdraw)());
    balances[msg.sender] -= _withdraw;
}
```solidity
function deposit() public payable {
    balances[msg.sender] += msg.value;
}

function withdraw(uint256 _withdraw) public payable {
    require(balances[msg.sender] >= _withdraw);
    require(msg.sender.call.value(_withdraw)());
    balances[msg.sender] -= _withdraw;
}

function attack(address target) payable {
    c = VictimContract(target);
    c.deposit.value(0.1 ether)();
    c.withdraw(0.1 ether);
}

function() payable {
    c.withdraw(0.1 ether);
}
```
- Always use transfer()
- Use sensitive operations before calls to other contracts
- ReentrancyGuard by OpenZeppelin

Reference: https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/ReentrancyGuard.sol
Reentrancy is not really obsolete
“A bridge to connect all bridges”
Connects ERC20 tokens
Not reentrancy per se

```solidity
function gift(ERC20 _token, uint256 _amount, address _receiver) {
    require(_token.transfer(_receiver, _amount);
    nativetoken.mint(msg.sender, reward);
}
```

More like call to the unknown

```solidity
function transfer(address _adress, uint _amount) {
    vuln_contract.gift(_attack_contract, _amount, _attack_contract);
}
```
● Trust that called function will succeed
● Trust that you know what the called function do

```javascript
function destroy(){
    require(!destroyed);
    msg.sender.send(1000 ether);
    destroyed = true;
}
```
External Contract - Mitigation

- Ensure external contract returns expected value
- Perform sensitive operations before calling external contract
- Never assume external contract functionality
**Proof of Weak Hands Coin - 866 Eth**

**BatchOverflow**


<table>
<thead>
<tr>
<th>Age</th>
<th>From</th>
<th>To</th>
<th>Value</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 hrs 8 mins ago</td>
<td>0x3f2dd0cb25bbf89...</td>
<td>0x521c526d5b50de...</td>
<td>57,896,044,618,658,100,000,000,000...</td>
<td>Erc20</td>
</tr>
<tr>
<td>2 hrs 8 mins ago</td>
<td>0x3f2dd0cb25bbf89...</td>
<td>0x4473c6396eba3d...</td>
<td>57,896,044,618,658,100,000,000,000...</td>
<td>Erc20</td>
</tr>
<tr>
<td>2 hrs 54 mins ago</td>
<td>0x3f2dd0cb25bbf89...</td>
<td>0x66f471fd1c471bb...</td>
<td>57,896,044,618,658,100,000,000,000...</td>
<td>Erc20</td>
</tr>
<tr>
<td>2 hrs 54 mins ago</td>
<td>0x3f2dd0cb25bbf89...</td>
<td>0x4473c6396eba3d...</td>
<td>57,896,044,618,658,100,000,000,000...</td>
<td>Erc20</td>
</tr>
</tbody>
</table>
● Integer only
● Conversions (Unsigned -> Signed)

More overflow cases:
https://github.com/ethereum/solidity/issues/796#issuecomment-253578925
• Perform operations within boundaries
• SafeMath library

```solidity
function add(uint256 _a, uint256 _b) internal pure returns (uint256) {
    uint256 c = _a + _b;
    require(c >= _a);
    return c;
}
```

• Blockchain CSRF?
• tx.origin vs msg.sender

```solidity
function withdraw(uint _amount) {
  require(tx.origin == owner);
  msg.sender.transfer(1000 ether);
}
```
<table>
<thead>
<tr>
<th>call()</th>
<th>For Vulnerable Contract</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx.origin</td>
<td>User</td>
</tr>
<tr>
<td>msg.sender</td>
<td>User</td>
</tr>
</tbody>
</table>
### Call Function Analysis

<table>
<thead>
<tr>
<th>call()</th>
<th>For Malicious Contract</th>
<th>For Vulnerable Contract</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx.origin</td>
<td>User</td>
<td>User</td>
</tr>
<tr>
<td>msg.sender</td>
<td>User</td>
<td>Malicious Contract</td>
</tr>
</tbody>
</table>
- call vs delegatecall
- Caution when using tx.origin
- Looping over unknown arrays
- Leads to denial of service

```solidity
function distribute() public {
    for(uint i = 0; i < investors.length; i++) {
        // Important administrative tasks
        // ...
    }
}
```
● Avoid looping over arrays of unknown length
● Set an upper limit for the array length
● Control the loop by checking gasleft()

```javascript
function distribute() public {
    while(i < investors.length && gasleft() > 20000) {
        // Important administrative tasks
        // ...
    }
}
```
Things we didn’t cover

- Randomness (is hard)
- Visibility (Function, secret)
- Unexpected ether (contract-suicide)
- Delegatecall (3rd party libraries)
- Storage
Questions?