CONSENSYS

Diligence

Reentrancy redux & best practices

DEVCON4
NOVEMBER 2, 2018

JOSEPH CHOW

CONSENSYS
Diligence

One line of code spurred a series of momentous events in blockchain history

Flashback - June 12 2016

Protect against recursive withdrawRewardFor attack #242

Merged **CJentzsch** merged 1 commit into `slockit:master` from `LefterisJP:withdraw_reward_for_recursive_attack` on Jun 13

Conversation 5 Commits 1 Files changed 1

Showing changes from **all commits** **1 changed file** +2 −1

3 DAO.sol **Show notes**

```
@@ -744,9 +744,10 @@ contract DAO is DAOInterface, Token, TokenCreation {
744 744
745 745          reward = rewardAccount.balance < reward ? rewardAccount.balance : reward;
746 746
    747 +        paidOut[_account] += reward;
747 748          if (!rewardAccount.payOut(_account, reward))
748 749              throw;
749     -        paidOut[_account] += reward;
    750 +
750 751          return true;
751 752      }
752 753
```

Outline

1. Reentrancy
2. What makes reentrancy possible?
3. Reentrancy in October 2018
4. Best practices
5. Hack some token contracts

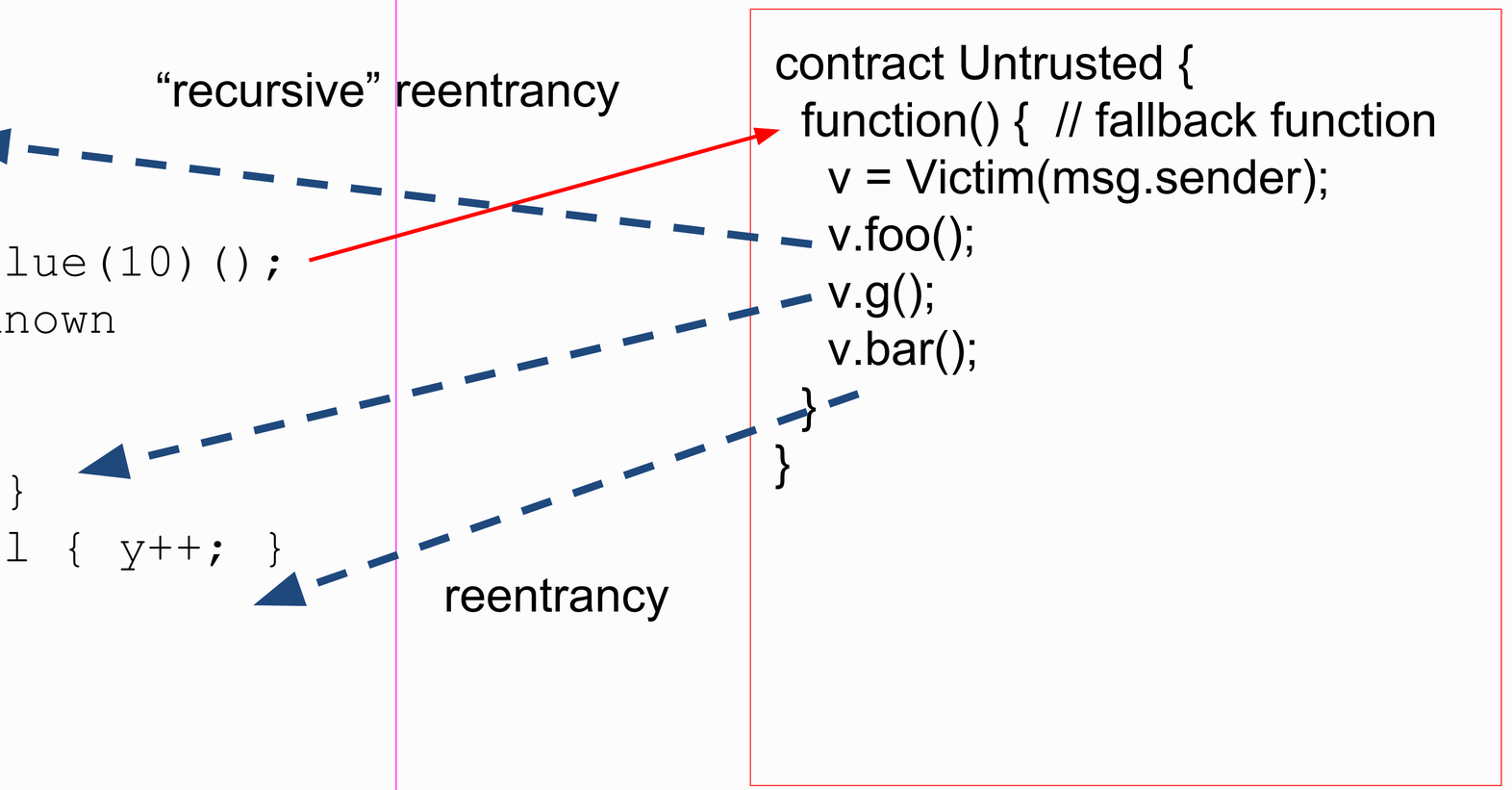
Reentrancy

```
contract Victim {
  // state
  int x = 2;
  uint private y = 1;

  function foo() {
    x--;
    msg.sender.call.value(10)();
    // x, y is now unknown
  }

  function g() { x++; }
  function h() internal { y++; }
  function bar() {
    if (x%2 == 0) h();
  }

}
```



What makes reentrancy possible?

#1: Calling an untrusted contract

This can be generalized to...

#1: Making an external call to a contract (you haven't written)

Even if you trust a contract someone has written, they might have bug where they call a malicious contract: that's enough for your contract to get attacked and exploited.

Re-entrancy in 2016

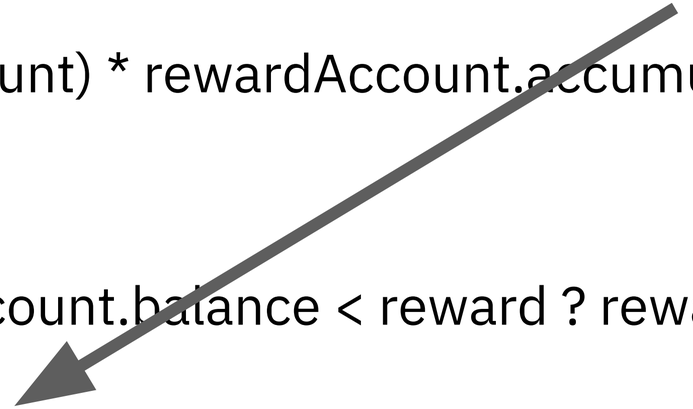
```
function withdrawRewardFor(address _account) noEther internal returns (bool _success) {
    if ((balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply <
paidOut[_account])
        throw;

    uint reward =
        (balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply -
paidOut[_account];

    reward = rewardAccount.balance < reward ? rewardAccount.balance : reward;

    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}
```

UNTRUSTED CONTRACT EXTERNAL CALL



#2: Changing state after an external call

After an external call, the state of the contract can be unknown.

Thus, making a state change or sending ETH after this unknown state, can be fatal.

Re-entrancy in 2016

```
function withdrawRewardFor(address _account) noEther internal returns (bool _success) {
    if ((balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply <
paidOut[_account])
        throw;

    uint reward =
        (balanceOf(_account) * rewardAccount.accumulatedInput()) / totalSupply -
paidOut[_account];

    reward = rewardAccount.balance < reward ? rewardAccount.balance : reward;

    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}
```

UNTRUSTED CONTRACT EXTERNAL CALL

STATE CHANGE AFTER EXTERNAL CALL


We learned from TheDAO,
but still need to be very
careful

https://etherscan.io/address/0xf91546835f756da0c10cfa0cda95b15577b84aa7#code

Reentrancy in 2018 October

```
function LCOpenTimeout(bytes32 _lcID) public {
    require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
    require(now > Channels[_lcID].LCopenTimeout);

    if(Channels[_lcID].initialDeposit[0] != 0) {
      Channels[_lcID].partyAddresses[0].transfer(Channels[_lcID].ethBalances[0]);
    }
    if(Channels[_lcID].initialDeposit[1] != 0) {
      require(Channels[_lcID].token.transfer(Channels[_lcID].partyAddresses[0],
Channels[_lcID].erc20Balances[0]),"CreateChannel: token transfer failure");
    }
    emit DidLCClose(_lcID, 0, Channels[_lcID].ethBalances[0], Channels[_lcID].erc20Balances[0], 0, 0);

    // only safe to delete since no action was taken on this channel
    delete Channels[_lcID];
  }
```

Reentrancy in 2018

```
function LCOpenTimeout(bytes32 _lcID) public {
    require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
    require(now > Channels[_lcID].LCopenTimeout);

    if(Channels[_lcID].initialDeposit[0] != 0) {
        Channels[_lcID].partyAddresses[0].transfer(Channels[_lcID].ethBalances[0]);
    }
    if(Channels[_lcID].initialDeposit[1] != 0) {
        require(Channels[_lcID].token.transfer(Channels[_lcID].partyAddresses[0],
Channels[_lcID].erc20Balances[0]),"CreateChannel: token transfer failure");
    }
    emit DidLCClose(_lcID, 0, Channels[_lcID].ethBalances[0], Channels[_lcID].erc20Balances[0], 0, 0);

    // only safe to delete since no action was taken on this channel
    delete Channels[_lcID];
}
```

TRANSFER ETH

TRANSFER TOKENS

CLEAN-UP

Reentrancy in 2018

```
function LCOpenTimeout(bytes32 _lcID) public {
    require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
    require(now > Channels[_lcID].LCopenTimeout);

    if(Channels[_lcID].initialDeposit[0] != 0) {
      Channels[_lcID].partyAddresses[0].transfer(Channels[_lcID].ethBalances[0]);
    }
    if(Channels[_lcID].initialDeposit[1] != 0) {
      require(Channels[_lcID].token.transfer(Channels[_lcID].partyAddresses[0],
Channels[_lcID].erc20Balances[0]),"CreateChannel: token transfer failure");
    }
    emit DidLCClose(_lcID, 0, Channels[_lcID].ethBalances[0], Channels[_lcID].erc20Balances[0], 0, 0);

    // only safe to delete since no action was taken on this channel
    delete Channels[_lcID];
  }
```

Reentrancy in 2018

```
function LCOpenTimeout(bytes32 _lcID) public {
    require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
    require(now > Channels[_lcID].LCopenTimeout);

    if(Channels[_lcID].initialDeposit[0] != 0) {
        Channels[_lcID].partyAddresses[0].transfer(Channels[_lcID].ethBalances[0]);
    }
    if(Channels[_lcID].initialDeposit[1] != 0) {
        require(Channels[_lcID].token.transfer(Channels[_lcID].partyAddresses[0],
Channels[_lcID].erc20Balances[0]),"CreateChannel: token transfer failure");
    }
    emit DidLCClose(_lcID, 0, Channels[_lcID].ethBalances[0], Channels[_lcID].erc20Balances[0], 0, 0);

    // only safe to delete since no action was taken on this channel
    delete Channels[_lcID];
}
```

TRANSFER ETH

UNTRUSTED CALL

STATE CHANGE

This was hacked: ETH stolen

```
  function LCOpenTimeout(bytes32 _lcID) public {
      require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
      require(now > Channels[_lcID].LCopenTimeout);

      if(Channels[_lcID].initialDeposit[0] != 0) {
          Channels[_lcID].partyAddresses[0].transfer(Channels[_lcID].ethBalances[0]);
      }
      if(Channels[_lcID].initialDeposit[1] != 0) {
          require(Channels[_lcID].token.transfer(Channels[_lcID].partyAddresses[0],
Channels[_lcID].erc20Balances[0]),"CreateChannel: token transfer failure");
      }
      emit DidLCClose(_lcID, 0, Channels[_lcID].ethBalances[0], Channels[_lcID].erc20Balances[0], 0, 0);

      // only safe to delete since no action was taken on this channel
      delete Channels[_lcID];
  }
```

How could it have been prevented?

Vulnerable

1. Transfer ETH
2. Transfer token (external call, untrusted)
3. Delete channel (state change)

Safer

1. Transfer ETH
2. Delete channel (state change)
3. Transfer token (external call, untrusted)

Idea of fix

```
function LCOpenTimeout(bytes32 _lcID) public {
    require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
    require(now > Channels[_lcID].LCopenTimeout);

    delete Channels[_lcID];  // put this as high up as possible.  Need to save references for below.

    if(Channels[_lcID].initialDeposit[0] != 0) {
        Channels[_lcID].partyAddresses[0].transfer(Channels[_lcID].ethBalances[0]);
    }
    if(Channels[_lcID].initialDeposit[1] != 0) {
        require(Channels[_lcID].token.transfer(Channels[_lcID].partyAddresses[0],
Channels[_lcID].erc20Balances[0]),"CreateChannel: token transfer failure");
    }

    emit DidLCClose(_lcID, 0, Channels[_lcID].ethBalances[0], Channels[_lcID].erc20Balances[0], 0, 0);
}
```

```
function LCOpenTimeout(bytes32 _lcID) public {  // Sample fix
    require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
    require(now > Channels[_lcID].LCopenTimeout);
    uint256 memory initialDeposit = Channels[_lcID].initialDeposit;
    addresses[2] memory partyAddresses = Channels[_lcID].partyAddresses;
    uint256[4] memory ethBalances = Channels[_lcID].ethBalances;
    uint256[4] memory erc20Balances = Channels[_lcID].erc20Balances;
    HumanStandardToken memory untrustedToken = Channels[_lcID].token;

    delete Channels[_lcID];  // put this as high up as possible

    if(initialDeposit[0] != 0) {
       partyAddresses[0].transfer(ethBalances[0]);
    }
    if(initialDeposit[1] != 0) {
       require(untrustedToken.transfer(partyAddresses[0], erc20Balances[0]),"CreateChannel: token
transfer failure");
    }
    emit DidLCClose(_lcID, 0, ethBalances[0], erc20Balances[0], 0, 0); }
```

```
function LCOpenTimeout(bytes32 _lcID) public {  // Sample fix
    require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
    require(now > Channels[_lcID].LCopenTimeout);
    uint256 memory initialDeposit = Channels[_lcID].initialDeposit;
    addresses[2] memory partyAddresses = Channels[_lcID].partyAddresses;
    uint256[4] memory ethBalances = Channels[_lcID].ethBalances;
    uint256[4] memory erc20Balances = Channels[_lcID].erc20Balances;
    HumanStandardToken memory untrustedToken = Channels[_lcID].token;

    delete Channels[_lcID];  // put this as high up as possible

    if(initialDeposit[0] != 0) {
       partyAddresses[0].transfer(ethBalances[0]);
    }
    if(initialDeposit[1] != 0) {
       require(untrustedToken.transfer(partyAddresses[0], erc20Balances[0]),"CreateChannel: token
transfer failure");
    }
    emit DidLCClose(_lcID, 0, ethBalances[0], erc20Balances[0], 0, 0); }
```

Best practices do help make contracts safer & more secure

Smart contract best practices

https://consensys.github.io/smart-contract-best-practices/recommendations

Top 3 recommendations

since 2016

Use caution when making external calls

Mark untrusted contracts

Avoid state changes after external calls

Use caution when making external calls

Calls to untrusted contracts can introduce several unexpected risks or errors. External calls may execute malicious code in that contract or any other contract that it depends upon. As such, every external call should be treated as a potential security risk. When it is not possible, or undesirable to remove external calls, use the recommendations in the rest of this section to minimize the danger.

Mark untrusted contracts

When interacting with external contracts, name your variables, methods, and contract interfaces in a way that makes it clear that interacting with them is potentially unsafe. This applies to your own functions that call external contracts.

Avoid state changes after external calls

Whether using raw calls (of the form someAddress.call()) or contract calls (of the form ExternalContract.someMethod()), assume that malicious code might execute. Even if ExternalContract is not malicious, malicious code can be executed by any contracts it calls.

One particular danger is malicious code may hijack the control flow, leading to race conditions. (See Race Conditions for a fuller discussion of this problem).

If you are making a call to an untrusted external contract, avoid state changes after the call. This pattern is also sometimes known as the checks-effects-interactions pattern.

Naming untrustedToken

```
  function LCOpenTimeout(bytes32 _lcID) public {
      require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
      require(now > Channels[_lcID].LCopenTimeout);

      if(Channels[_lcID].initialDeposit[0] != 0) {
          Channels[_lcID].partyAddresses[0].transfer(Channels[_lcID].ethBalances[0]);
      }
      if(Channels[_lcID].initialDeposit[1] != 0) {
          require(Channels[_lcID].untrustedToken.transfer(Channels[_lcID].partyAddresses[0],
Channels[_lcID].erc20Balances[0]),"CreateChannel: token transfer failure");
      }
      emit DidLCClose(_lcID, 0, Channels[_lcID].ethBalances[0], Channels[_lcID].erc20Balances[0], 0, 0);

      // only safe to delete since no action was taken on this channel
      delete Channels[_lcID];
  }
```

New best practice? comment (ETH) transfers explicitly

```
function LCOpenTimeout(bytes32 _lcID) public {
    require(msg.sender == Channels[_lcID].partyAddresses[0] && Channels[_lcID].isOpen == false);
    require(now > Channels[_lcID].LCopenTimeout);

    if(Channels[_lcID].initialDeposit[0] != 0) {
      Channels[_lcID].partyAddresses[0].transfer(Channels[_lcID].ethBalances[0]);  // ETH transfer
    }
    if(Channels[_lcID].initialDeposit[1] != 0) {
      require(Channels[_lcID].untrustedToken.transfer(Channels[_lcID].partyAddresses[0],
Channels[_lcID].erc20Balances[0]),"CreateChannel: token transfer failure");
    }
    emit DidLCClose(_lcID, 0, Channels[_lcID].ethBalances[0], Channels[_lcID].erc20Balances[0], 0, 0);

    // only safe to delete since no action was taken on this channel
    delete Channels[_lcID];
  }
```

Expose transfers and state changes

1. Transfer ETH
2. Transfer token (external call, untrusted)
3. Delete channel (state change)

Vulnerable

1. Transfer ETH
2. Transfer token (external call, untrusted)
3. Delete channel (state change)

Safer

1. Transfer ETH
2. Delete channel (state change)
3. Transfer token (external call, untrusted)

Conclusion

Vulnerable

1. Transfer something
2. External call (untrusted)
3. State change (including clean up)

Safer

1. Transfer something
2. State change (including clean up)
3. External call (untrusted)

When an external call is required, all state changes should be done before the external call

Vulnerable

1. (Transfer something)
2. External call (untrusted)
3. State change (including clean up)

Safer

1. (Transfer something)
2. State change (including clean up)
3. External call (untrusted)

1. address.call.value()()
2. State change (including clean up)

TheDAO combined first two steps by using a powerful, but less safe method of transferring ETH.

Instead of address.call.value()(), safer is address.transfer()

Uses of address.call.value()() should be examined very carefully.

Top 3 recommendations

since 2016

Use caution when making external calls

Mark untrusted contracts

Avoid state changes after external calls

CONSENSYS
Diligence is hiring

Clients inc 0x, Aragon, Interactive ICO, Gnosis, uPort, Virtue Poker...

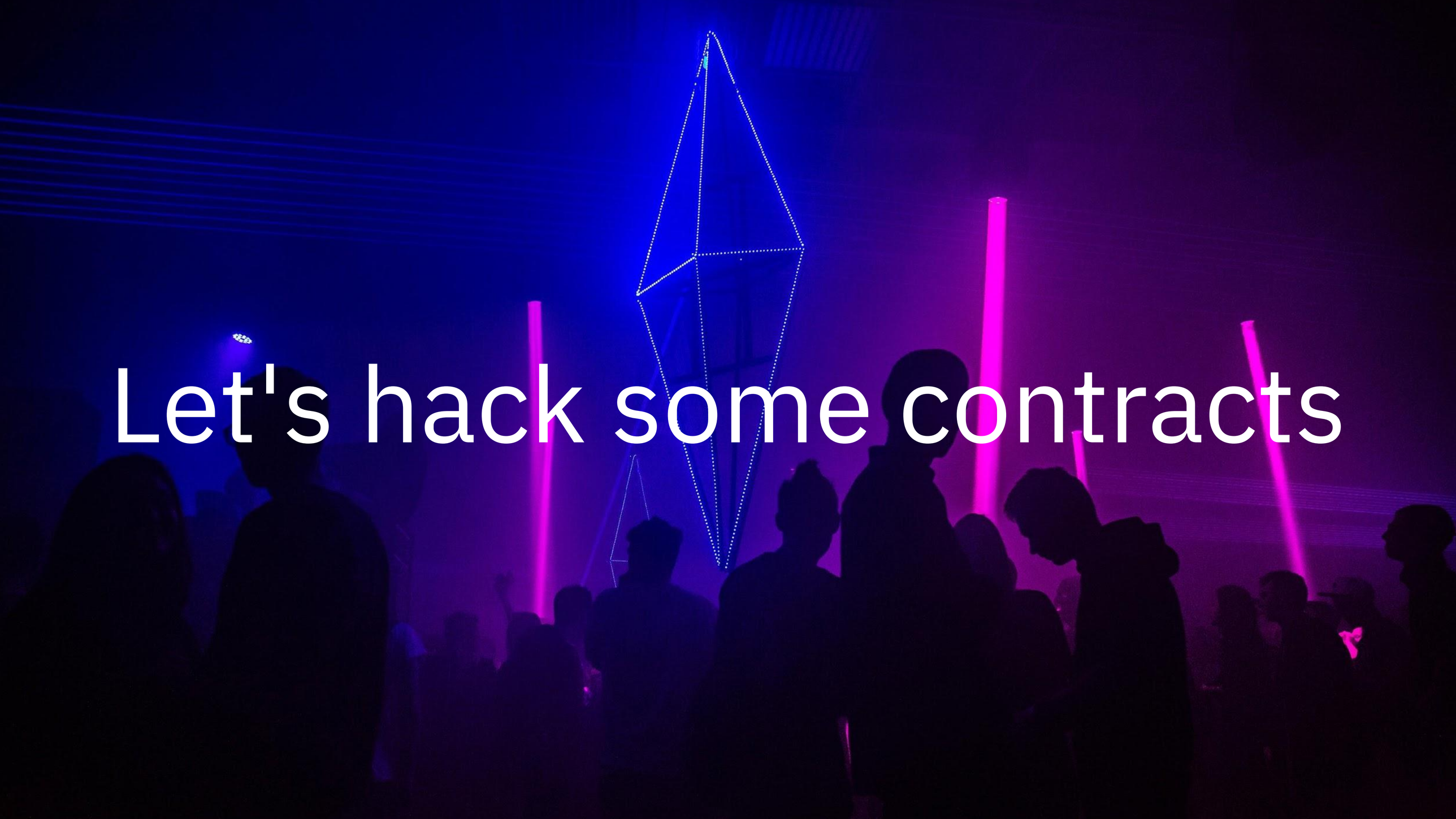
- Solidity and EVM experts interested in security
- Security engineers

https://media.consensys.net/mesh-spotlight-gon%C3%A7alo-sa-consensys-diligence-c2b7921d88a7

join-diligence@consensys.net

ConsenSys is also hiring scalability, sharding, systems experts

https://consensys.net/open-roles/1202993


Let's hack some contracts

https://capturetheether.com/challenges/math/token-sale

CONSENSYS
Diligence

https://capturetheether.com/challenges/math/token-whale

CONSENSYS
Diligence

https://capturetheether.com/challenges/miscellaneous/token-bank

CONSENSYS
Diligence

I want you to use and share this information as much as you want, modify it as you desire, thus it is public domain

CONSENSYS
Diligence

CC0 1.0 Universal (CC0 1.0)
Public Domain Dedication
No Copyright
The person who associated a work with this deed has dedicated the work to the public domain by waiving all of his or her rights to the work worldwide under copyright law, including all related and neighboring rights, to the extent allowed by law.

You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission. See Other Information below.

This license is acceptable for Free Cultural Works.
Other Information
In no way are the patent or trademark rights of any person affected by CC0, nor are the rights that other persons may have in the work or in how the work is used, such as publicity or privacy rights.
Unless expressly stated otherwise, the person who associated a work with this deed makes no warranties about the work, and disclaims liability for all uses of the work, to the fullest extent permitted by applicable law.
When using or citing the work, you should not imply endorsement by the author or the affirmer.