Smart Contract Security Tips

Ethereum.devcon2
Sep 20 2016 - Joseph Chow

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

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
```

Community resource: for the community, by the community

ethereum

https://github.com/ConsenSys/smart-contract-best-practices

https://github.com/ethereum/wiki/wiki/Safety

Feel free to edit the wiki or submit a pull request, for anything:

- Fix a typo, or example
- Add a link to a community blog post (even
- your own), or other related security info
- Write a new section

General Philosophy

- Prepare for failure
  - This is not defeat, but admitting unknown unknowns
- Roll out carefully
  - A production system needs baking time in production
  - Testnets, beta on mainnet, then production mainnet
- Keep contracts simple
- Stay up to date
  - Bibliography at https://github.com/ConsenSys/smart-contract-best-practices and https://github.com/ethereum/wiki/wiki/Safety
  - Includes community bloggers, Twitter, Reddit...
- Be aware of blockchain properties

Prepare for failure example (from SingularDTV)


```
uint fundBalance;
 function checkInvariants() constant internal {
      if (this.balance < fundBalance) throw;
 }
 function emergencyCall() external noEther {
      if (this.balance < fundBalance) {
           if (this.balance > 0 && workshop.send(this.balance)) {
                throw;
           }
      }
}
```

External Calls

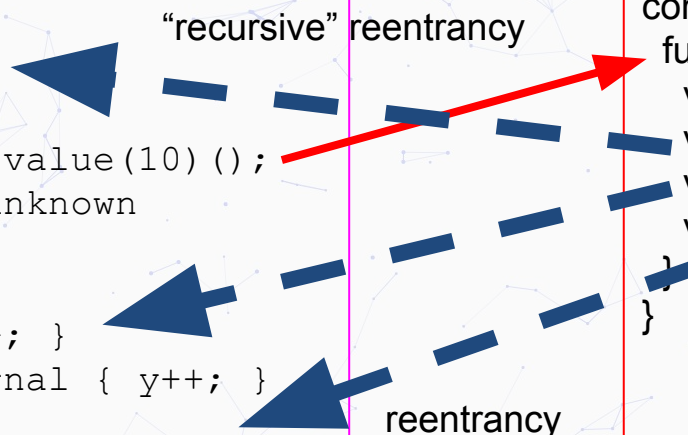
- Avoid calls to untrusted contracts as much as you can
  - Untrusted basically means a contract you've not written

- Assume untrusted contracts are malicious
- Avoid untrustedContract.doSomething()
- Avoid address.call()
  - Avoid address.delegatecall(), address.callcode()
- **After any untrusted call, assume that the state of your contract has been manipulated**

External Calls - Example

ethereum

```
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();
  }

}
```




```
contract Untrusted {
  function() {  // fallback function
    v = Victim(msg.sender);
    v.foo();
    v.g();
    v.bar();
  }
}
```

Use send(), avoid call.value()()

- // good

  if(!someAddress.send(100)) { ... // Some failure code }

- // bad

  if(!someAddress.call.value(100)()) { ... // Some failure code }

- send() is safe because attacker only gets 2,300 gas: only enough to log an event
- call.value()() passes along virtually all gas to the attacker's fallback function

Handle errors in raw calls

- Raw calls do not progagate exceptions
  - address.send(), address.call(), (delegatecall and callcode) return false if they fail
- Unlike ExternalContract(address).doSomething() which will throw if doSomething() throws
- // good

  if(!someAddress.send(100)) { ... // Some failure code }
- // bad

  someAddress.send(100); // an “unchecked send”

Keep fallback functions simple

- Receiving Ether from a .send(), fallback function only gets 2,300 gas: can only log an event
  - function() { LogDepositReceived(msg.sender); }
- Use a proper function if more gas is required
  - function deposit() external { balances[msg.sender] += msg.value; }

- // bad, uses more than 2,300 gas. Breaks senders that use send() instead of call.value()()

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

Call Depth Attack

- **Any** call (even a fully trusted and correct one) can be made to fail
- The EVM “CALL (and CREATE) stack” has a maximum depth of 1024
- Attacker can make recursive calls to depth 1023, then call your function and all of its subcalls will fail
- // INSECURE

```
mapping(address => uint) refunds;
function withdrawRefund(address recipient) {
  uint refund = refunds[recipient];
  refunds[recipient] = 0;
  recipient.send(refund); // this line is vulnerable to a call depth attack
}
```

- A solution is for msg.sender to “pull” their refund instead of a contract “push” to the recipient

More information

ethereum

"Pull" over "push" for external calls (and payments)

Denial of Service against contracts

Reentrancy and race conditions, and many more

https://github.com/ConsenSys/smart-contract-best-practices

https://github.com/ethereum/wiki/wiki/Safety

Feel free to edit the wiki or submit a pull request, for anything:

- Fix a typo, or example
- Add a link to a community blog post (even your own), or other related security info
- Write a new section

Conclusion

ethereum

Prepare for failure

Roll out carefully

Keep contracts simple

Calling untrusted code is always dangerous

A security resource of the community, by the community, for the community

https://github.com/ConsenSys/smart-contract-best-practices

https://github.com/ethereum/wiki/wiki/Safety

Appendix

Denial of Service

- Unexpected throw; the block gas limit; unbounded arrays; misunderstanding gas refunds.
- // INSECURE

```
contract Auction {
    address currentLeader;
    uint highestBid;
    function bid() {
        if (msg.value <= highestBid) { throw; }
        if (!currentLeader.send(highestBid)) { throw; } // Refund the old leader, and throw if it fails
        currentLeader = msg.sender;
        highestBid = msg.value;
    }
}
```

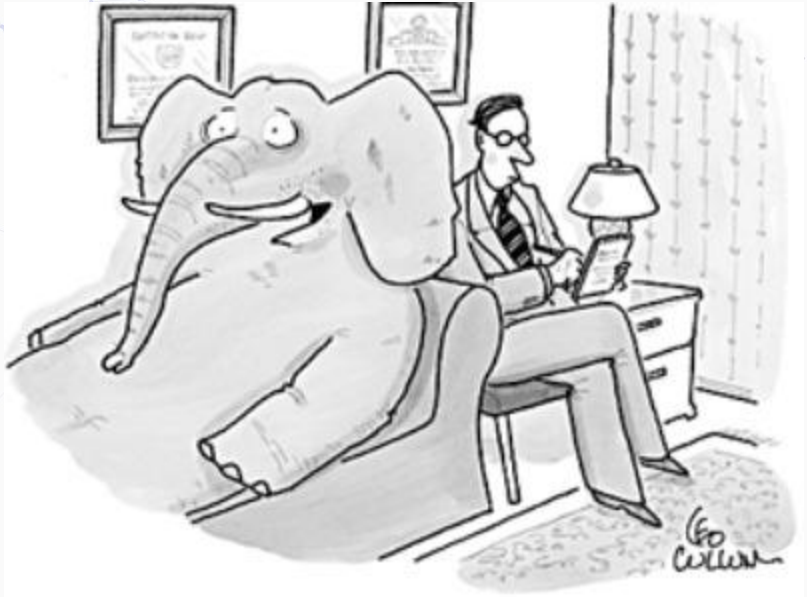
- A currentLeader that refuses payment will permanently be the leader.
- Throw can't be removed otherwise Call Depth Attack. Solution: favor "pull" over "push"

Favor “pull” over “push” for external calls

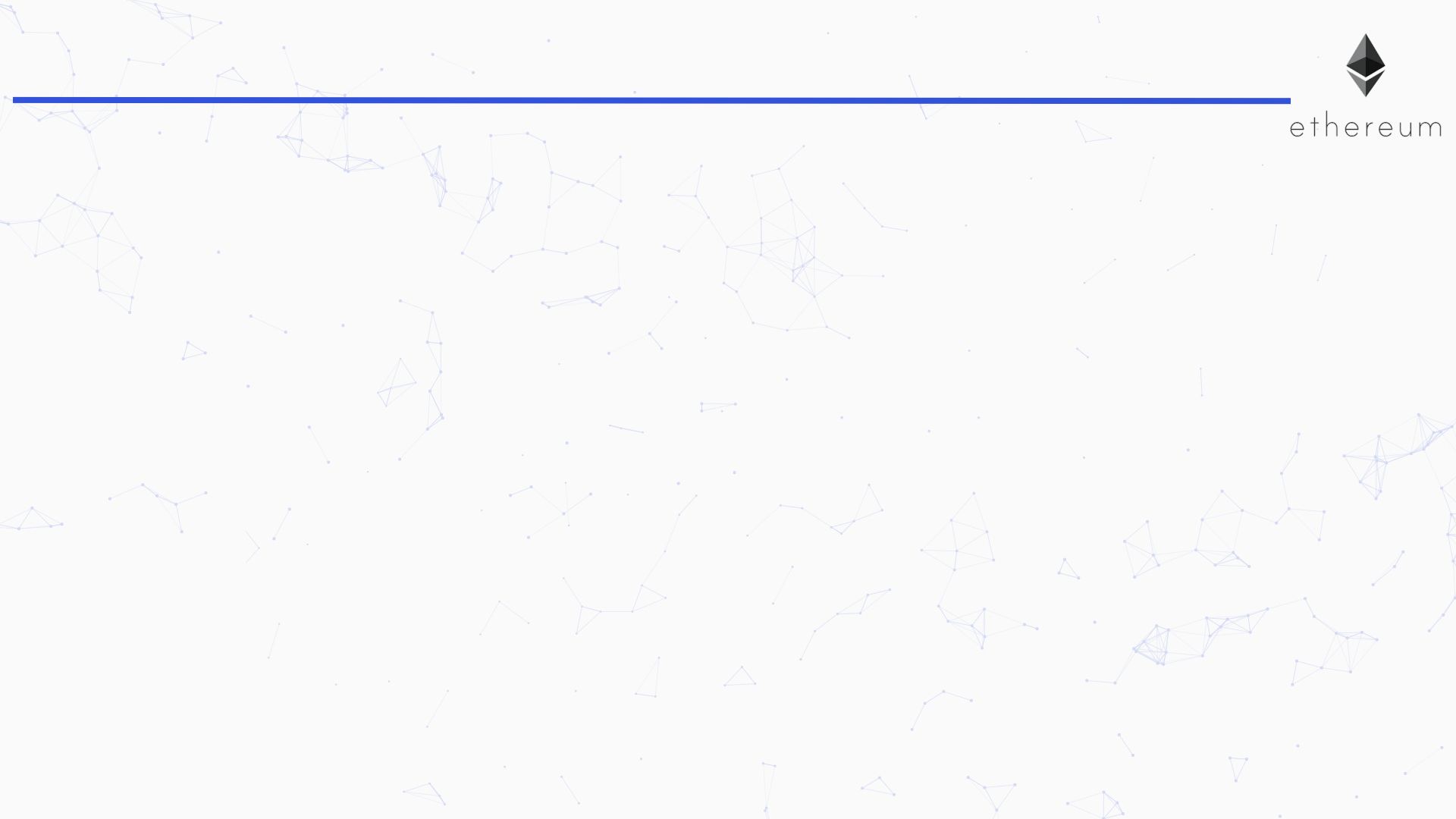

```
// good
contract auction {
    address highestBidder;
    uint highestBid;
    mapping(address => uint) refunds;

    function bid() external {
        if (msg.value < highestBid) throw;

        if (highestBidder != 0) {
            refunds[highestBidder] += highestBid; // record
the refund that this user can claim
        }

        highestBidder = msg.sender;
        highestBid = msg.value;
    }

    function withdrawRefund() external {
        uint refund = refunds[msg.sender];
        refunds[msg.sender] = 0;
        if (!msg.sender.send(refund)) {
            refunds[msg.sender] = refund; // reverting state
because send failed
        }
    }
}
```

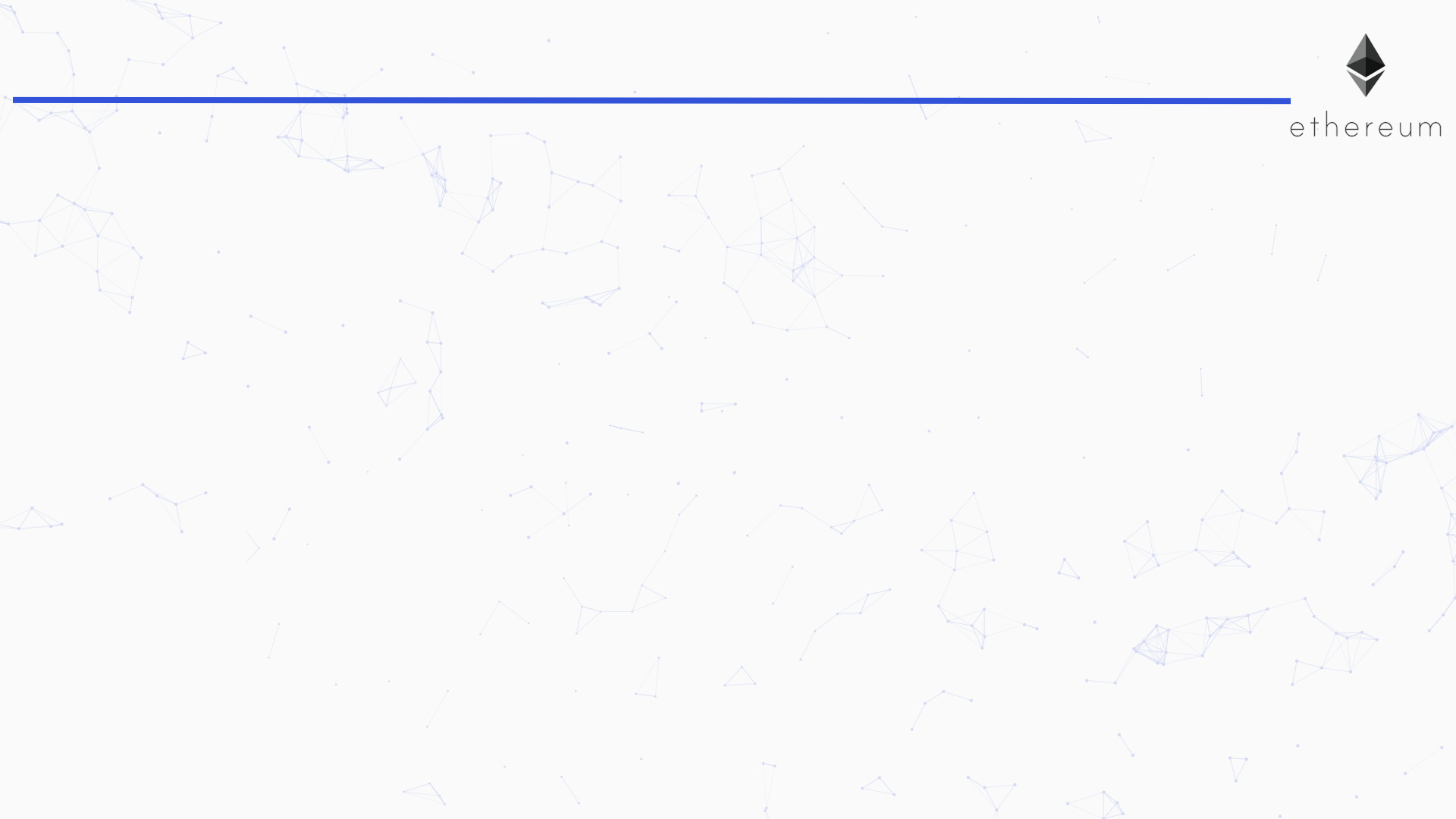
Smart Contract Security before TheDAO

ethereum



"I'm right there in the room, and no one even acknowledges me."


ethereum


ethereum