Step by Step Towards a Safe Contract: Insights from an Undergraduate Ethereum Lab

Elaine Shi, Andrew Miller
University of Maryland
• Students worked in groups of 4.

• Each group is assigned one graduate student advisor
  – My Ph.D. students
Ethereum Lab Setup

• **Phase 1: Proposer phase**
  – Students develop of choice on Ethereum

• **Phase 2: Amendment/critique phase**
  – Instructors and graduate TAs give feedback
  – Students critique each other’s designs
  – Students amend their designs
The good news:

• An inspiring experience where the students, my Ph.D. students, and I learned together

• Some students said that they really enjoy learning about crypto-currency. Crypto-currency is awesome.

• All students did an impressive job!
The bad news:

• Some students did not like the experience due to the in-development nature of the Serpent language -- despite the fact that they all did an impressive job!!
Games (where people play for money)
  - Rock paper scissors, Russian Roulette, and many others

Escrow service

Auctions
  - blind auctions, silent auction

Parking meter

Stock market app
Lessons Learned

• In Phase 1, we noticed that students created many “insecure” contracts.

• Conclusion:
  – **Security is difficult.**
  – Programming smart contracts: **you can mess up in new ways** in comparison with traditional programming.
Step by Step Towards a Safe Contract
The Simplified Contract Programming Model

Contract

- Store messages
- Store $$$
- Program logic

Users exchange messages and store funds in the contract.
The Simplified Contract Programming Model

Contract
- Store messages
- Store $$$
- Program logic

Data stored on block chain
Program executed by all miners
The Simplified Contract Programming Model

Contract

- Store messages
- Store $$$
- Program logic

“A trusted 3rd party with no privacy”
def add_player()
/* Player 1 and 2 enter the game by sending money.
   Contract records their identities. */

def input()
/* Player 1 and 2 sends their input to contract
   Contract records their inputs */

def winner()
/* Decide the winner and sends balance on the contract
to the winner*/
**Typical mistake 1:**

```python
def add_player():
    if not self.storage['player1'] and msg.value > 1000:
        self.storage['player1'] = msg.sender
        self.storage['WINNINGS'] = self.storage['WINNINGS'] + msg.value
        return(1)
    elif not self.storage['player2'] and msg.value > 1000:
        self.storage['player2'] = msg.sender
        self.storage['WINNINGS'] = self.storage['WINNINGS'] + msg.value
        return(2)
    else:
        return(0)
```
Typical mistake 1: corner cases in state machine

```python
def add_player():
    if not self.storage["player1"] and msg.value > 1000:
        self.storage["player1"] = msg.sender
        self.storage["WINNINGS"] = self.storage["WINNINGS"] + msg.value
        return(1)
    elif not self.storage["player2"] and msg.value > 1000:
        self.storage["player2"] = msg.sender
        self.storage["WINNINGS"] = self.storage["WINNINGS"] + msg.value
        return(2)
    else:
        return(0)
```

If 3rd player enters, or player sends < 1000 ethers, money is leaked

Similar mistakes arise in other applications.
def input(choice):
    if self.storage["player1"] == msg.sender:
        self.storage["p1value"] = choice
        return(1)
    elif self.storage["player2"] == msg.sender:
        self.storage["p2value"] = choice
        return(2)
    else:
        return(0)
Typical mistake 2: failure to use cryptography

Players’ choices sent and stored in cleartext.
Typical mistake 2: failure to use cryptography

```python
def input(choice):
    if self.storage["player1"] == msg.sender:
        self.storage["p1value"] = choice
        return(1)
    elif self.storage["player2"] == msg.sender:
        self.storage["p2value"] = choice
        return(2)
    else:
        return(0)
```

Solution: use cryptographic commitment
Typical mistake 3:

```python
def opencommit(choice, r):
    if self.storage["player1"] == msg.sender and "(choice, r)" is a valid opening of self.storage["p1value"]:
        self.storage["p1value"] = choice
        self.storage["opened1"] = 1
    elif self.storage["player2"] == msg.sender and "(choice, r)" is a valid opening of self.storage["p2value"]:
        self.storage["p2value"] = choice
        self.storage["opened2"] = 1
```
Typical mistake 3: incentive incompatible contracts

```python
def opencommit(choice, r):
    if self.storage["player1"] == msg.sender and
        "(choice, r)" is a valid opening of self.storage["p1value"]:
        self.storage["p1value"] = choice
        self.storage["opened1"] = 1
    elif self.storage["player2"] == msg.sender and
        "(choice, r)" is a valid opening of self.storage["p2value"]:
        self.storage["p2value"] = choice
        self.storage["opened2"] = 1
```

Player has no incentive to open commitment when he sees that he is losing.
Typical mistake 3: incentive incompatible contracts

```python
def opencommit(choice, r):
    if self.storage["player1"] == msg.sender and
        "(choice, r)" is a valid opening of self.storage["p1value"]:
        self.storage["p1value"] = choice
        self.storage["opened1"] = 1
    elif self.storage["player2"] == msg.sender and
        "(choice, r)" is a valid opening of self.storage["p2value"]:
        self.storage["p2value"] = choice
        self.storage["opened2"] = 1
```

Solution:
Require deposit to play. Player loses deposit if commitment is not opened after a time-out.
Coming Up Soon

- Online course materials for programming smart contracts
- Lab instructions and accompanying virtual machines
  – For instructors teaching cryptocurrency
# crowd funding contract

def campaign_ended():
    ...
    if campaign_deadline and goal_not_reached:
        # Refund all the donors
        for i in range(n donors):
            send(donor[i], value[i])

    ...
More Subtle Bugs

# crowdfunding contract

def campaign_ended():
    ...
    if campaign_deadline and goal_not_reached:
        # Refund all the donors
        for i in range(n_donors):
            send(donor[i], value[i])
    ...

Callstack can be at most 1024. If campaign_ended() is called at depth 1023, then send fails, no one gets their refund.
Thank you!

elaine@cs.umd.edu

http://www.cs.umd.edu/~elaine