Blockchain Ponzi Scheme

Freddie, Patrick, Arzu, Terry, Hope

Background: Ponzi Schemes

https://www.sciencedirect.com/science/article/pii/S0167739X18301407

- “Ponzi schemes are financial frauds which lure users under the promise of high profits. Actually, users are repaid only with the investments of new users joining the scheme...”

- “A recent study [6] estimates that Ponzi schemes operated through Bitcoin have gathered more than millions *USD* in the period from September 2013 to September 2014.2”

R1 the contract distributes money among investors, according to some logic.

R2 the contract receives money *only* from investors.

R3 each investor makes a profit *if* enough investors invest enough money in the contract afterwards.

R4 the later an investor joins the contract, the greater the risk of losing his investment.

Background: Smart Contract Ponzi Schemes

Smart Contracts create a ponzi opportunity with several attractive factors

1. Initiator may remain anonymous
2. Smart Contracts are unmodifiable and unstoppable, with no central authority to intervene
3. There is a "...false sense of trustworthiness from the fact that the code of smart contracts is public and immutable, and their execution is automatically enforced."

Types of Smart Contract Ponzi Schemes

1. Chain: The schemes in this category usually multiply the investment by a predefined constant factor, which is equal for all users. The scheme starts paying back users, one at a time, in order of arrival, and in full
2. Waterfall: Each new investment is poured along the chain of investors, so that each can take their share. The Waterfall creates a variable profit, whereas the Chain creates a fixed profit.
3. Handover: A type of chain-shaped scheme, where the entry toll is determined by the contract, and it increased each time a new investor joins the scheme.

Goal

- Problem: The lure of this type of scheme creates a false sense of security, where the currency itself is secure, but the influx of new currency is not guaranteed.
- Goal: Create a waterfall smart contract ponzi scheme utilizing Ganache & Solidity.

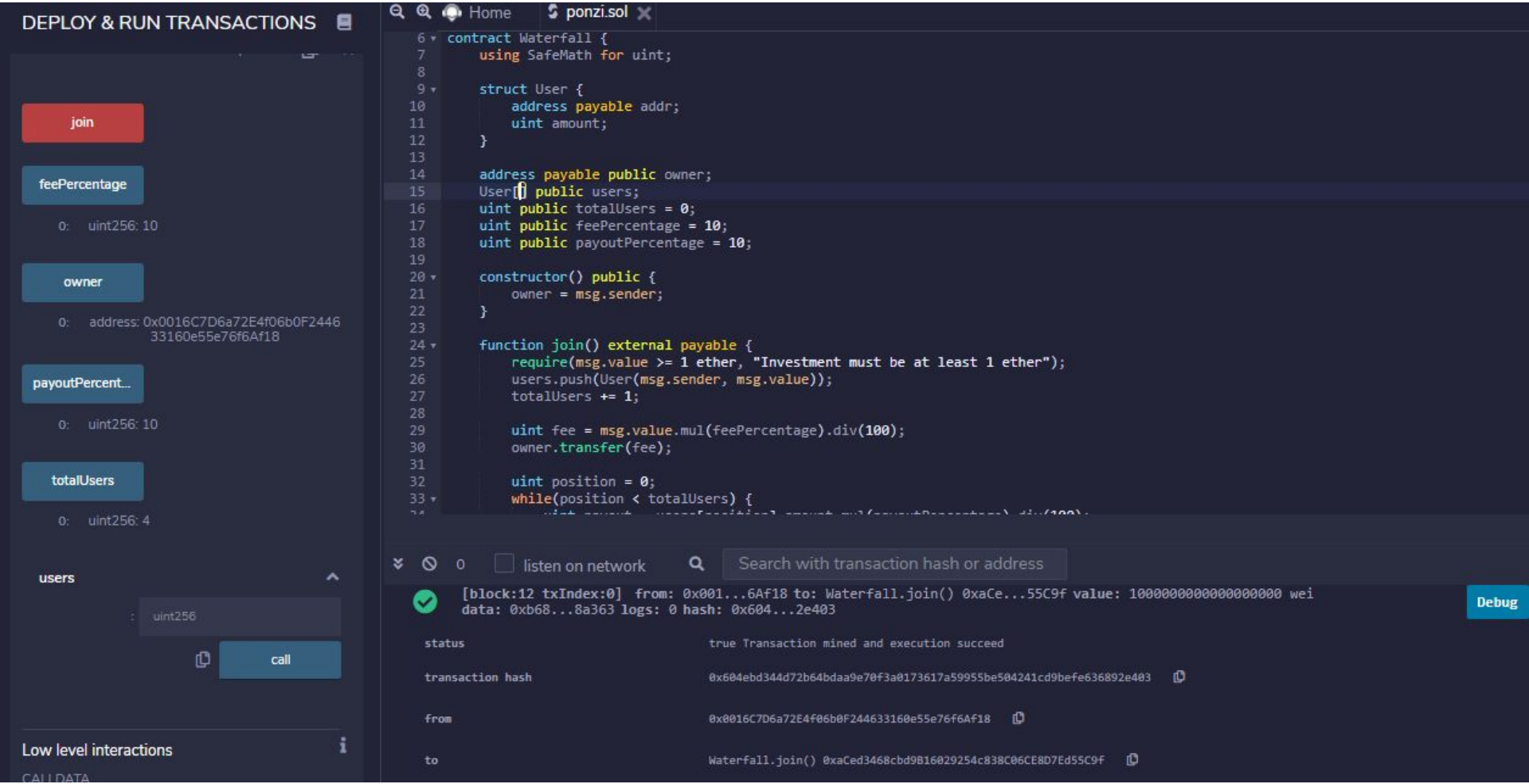
Considerations for Avoidance

- Review the advertisements. Alluring too-good-to-be-true ads are often scams
- The Solidity code should be checked for bugs and intentional “errors” that benefit the initiator. Simple code is more transparent, code with 300+ lines of code can start looking fishy and produce more errors.
- Review Transaction Logs:
    a. “...only a few users have a ratio greater than 1: the most numerous classes are those of users who never received any money back, or have a ratio between 0 and 1;
    b. most Ponzi schemes have a relatively short lifespan, consisting in a peak of intense activity followed by a period of stagnation;
    c. the Gini coefficients of the *payouts* of Ponzi schemes tend to be high, meaning a strong inequality in the distribution of money.”

Application

Home | ponzi.sol

```
pragma solidity ^0.5.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.0/contracts/math/SafeMath.sol";


contract Waterfall {
    using SafeMath for uint;

    struct User {
        address payable addr;
        uint amount;
    }

    address payable public owner;
    User[] public users;
    uint public totalUsers = 0;
    uint public feePercentage = 10;
    uint public payoutPercentage = 10;

    constructor() public {
        owner = msg.sender;
    }

    function join() external payable {
        require(msg.value >= 1 ether, "Investment must be at least 1 ether");
        users.push(User(msg.sender, msg.value));
        totalUsers += 1;

        uint fee = msg.value.mul(feePercentage).div(100);
        owner.transfer(fee);

        uint position = 0;
        while(position < totalUsers) {
            uint payout = users[position].amount.mul(payoutPercentage).div(100);
            if(payout > address(this).balance){
                break;
            }
            users[position].addr.transfer(payout);
            position += 1;
        }
    }
}
```


DEPLOY & RUN TRANSACTIONS
join
feePercentage
0: uint256: 10
owner
0: address: 0x0016C7D6a72E4f06b0F2446 33160e55e76f6Af18
payoutPercent...
0: uint256: 10
totalUsers
0: uint256: 4
users
uint256
call
Low level interactions
CALLDATA
Home
ponzi.sol
contract Waterfall {
using SafeMath for uint;
struct User {
address payable addr;
uint amount;
}
address payable public owner;
User[] public users;
uint public totalUsers = 0;
uint public feePercentage = 10;
uint public payoutPercentage = 10;
constructor() public {
owner = msg.sender;
}
function join() external payable {
require(msg.value >= 1 ether, "Investment must be at least 1 ether");
users.push(User(msg.sender, msg.value));
totalUsers += 1;
uint fee = msg.value.mul(feePercentage).div(100);
owner.transfer(fee);
uint position = 0;
while(position < totalUsers) {
listen on network
Search with transaction hash or address
[block:12 txIndex:0] from: 0x001...6Af18 to: Waterfall.join() 0xaCe...55C9f value: 1000000000000000000 wei
data: 0xb68...8a363 logs: 0 hash: 0x604...2e403
Debug
status
true Transaction mined and execution succeed
transaction hash
0x604ebd344d72b64bdaa9e70f3a0173617a59955be504241cd9befe636892e403
from
0x0016C7D6a72E4f06b0F244633160e55e76f6Af18
to
Waterfall.join() 0xaCed3468cbd9B16029254c838C06CE8D7Ed55C9f