Resource-Oriented Programming
in Libra Move and Flow Cadence

Hsuan Lee  李玄
Co-founder & CEO, portto
Who am I?

1. Co-founder & CEO @ portto
2. VP of Engineering @ Cobinhood & DEXON
3. Software Engineer @ 17 Media, Agoda, Yahoo
Outline of this Speech

1. The history of programming languages
2. Pain-points in smart contracts
3. Resource-oriented programming
4. Some examples
History of programming languages

1950s
“High-Level”
Fortran, C, Lisp

1970s
Relational DB
SQL

1970s
Object-Oriented
C++, Java, Objective-C

1990s
Internet
JavaScript, Flash
Programming on blockchain

- Introduced by Ethereum
- General purpose programming
- Suitable for
  - Transfer scarce assets
  - Control access
  - Provide auditable execution
  - Provide traceable proof??
What’s wrong with current model?

- Centralized ledger
- Reduce chance of parallelism
- Data structure does not reflect ownership
- Huge attack surface
- Difficult to audit & analyze
Common attacks

1. Reentrance
   a. DAO hack:
      https://quantstamp.com/blog/what-is-a-re-entrancy-attack
   b. ERC777 + Uniswap / Lendf.me:

2. Abuse authorization
   a. Parity wallets got locked:
      https://github.com/openethereum/openethereum/issues/6995
   b. Centralized ERC20:
      https://etherscan.io/address/0xc12d1c73ee7dc3615ba4e37e4abfbd7d8a38907e
Existing

Access Control
Who you are (list)

Scarce Assets
Data structure

Security
Contract level

ROP

Access Control
What you have

Scarce Assets
Resources

Security
VM level
Resource lifecycle

Create: Carefully controlled
Transfer: Uncopyable, Must be assigned once
Delete: Deliberate, Never by accident
ROP accounts

Account 0
- Public
  - Ref #1
  - Res #1
- Private
  - Ref #2
  - Res #2
  - Storage

Account 1
- Public
  - Ref #2
- Private
  - Storage
Ethereum fungible token

```solidity
contract ERC20 {
    mapping (address => uint256) private _balances;

    function _transfer(address sender, address recipient, uint256 amount) {
        // ensure the sender has a valid balance
        require(_balances[sender] >= amount);

        // subtract the amount from the senders ledger balance
        _balances[sender] = _balances[sender] - amount;

        // add the amount to the recipient’s ledger balance
        _balances[recipient] = _balances[recipient] + amount
    }
}
```
rop fungible token

```rust
pub resource Vault: Provider, Receiver {
    pub var balance: UFix64

    init(balance: UFix64) {
        self.balance = balance
    }

    pub fun withdraw(amount: UFix64): @Vault {
        self.balance = self.balance - amount
        return <-create Vault(balance: amount)
    }

    pub fun deposit(from: @Vault) {
        self.balance = self.balance + from.balance
        destroy from
    }
}
```
ROP fungible token

Account 0
- Public
- Private
  - Ref mint
- Storage
  - Vault
  - Minter
- Code
  - Token
  - Contract

Account 1
- Public
- Private
  - Ref balance deposit
  - Ref withdraw
- Storage
  - Vault
  - 30

Account 2
- Public
- Private
  - Ref balance deposit
  - Ref withdraw
- Storage
  - Vault
  - 0
ROP fungible token demo

https://play.onflow.org/26b79fc4-bde4-4783-85ca-b5bdbfdbc543
ROP Advantages

- Built-in security
- Less human error
- Better parallelism
- State rent made possible
- Resource hierarchy
Resource-oriented resources

- Getting Started With Move
- Cadence Language Reference
- Cadence Fungible Tokens
Good Stuff

Flow 台灣開發者社群  
Blocto 開發者 Discord  
幣安台灣徵才中
Questions?