Basic Pentesting on Ethereum Blockchain

Suen Chun Hui

https://www.linkedin.com/in/chunhuisuen/
Understanding the stack
Blockchain as a stack

Wallet layer
Client layer
Application layer | Peering layer
Logic layer | Storage layer
Scalability layer | Permissionless Privacy layer
Base layer
P2P networking layer

© Dr Suen Chun Hui 2019
P2P Networking Layer - attacks

- networking and connectivity layer, similar to P2P overlay networks
  - node discovery (dynamic list of nodes to connect to)
  - secure connection between nodes

- Can be attacked by DoS
  - eclipse attack can be serious, if consensus(base layer) inherently assumes sufficient randomness of peer list or peer connections

© Dr Suen Chun Hui 2019
Base layer attack

- Base consensus mechanism (eg. PoW, PoS)

- **51% attack**, attack on consensus protocol
  - means the majority collude to attack other participants
  - affects everyone in network, *including decentralized exchanges, autonomous smart contracts, etc*

- ↑network size, ↓risk of 51% attack
Scalability layer - challenges

- improve overall scalability of chain (throughput, may have side-effect on latency or commit time)
- using a 'divide and conquer' approach to split base consensus in 2(or more) layers

- Techniques: DPoS, Sharding
  - Challenges – maintaining atomicity across layers and shards
Privacy layer

- Known privacy techniques
  - Mixing / Ring signatures
  - Secure Multi-party computation
  - Zero Knowledge proofs
- Weakness of ZKP techniques
  - Snark – “Toxic waste” issue
  - Other parameters: proof size, proof/verify speed
Logic layer - smart contract security

- non-turing complete language
  - lesser features
  - ↓ risk of security bug
- turing complete
  - more features
  - ↑ risk fo security bug (eg. infinite loop)
  - need more security checking tools
Client layer

- full vs light node
  - **full node keeps all data**
  - light node only keep hash of all blocks and not content of block
- light node
  - **pulls data on-demand from full nodes**
  - light node is able to verify TX if data provided by full node
Smart Contract – pentesting:
Notable security holes
Why Smart Contract pentesting?

- Bytecode (optionally contract mode) is public
- Code execution (by miners) is remote, decentralized and anonymous
- Hackers are remote and anonymous
- Security flaw has big loss (direct financial loss) and no recourse (no centralized authority to address loss, eg. police, bank, court)
Some concepts

- GAS – transaction fee paid per transaction. Calculated based on computation and storage opcodes
- Fallback function – allows a smart contract to ‘accept’ eth payment like a normal wallet address and act upon it.

```python
function () public external { ... }
```
Reentrancy Attacks

- Early ethereum startup, bug in DAO (decentralized autonomous organization) smart contract
- Caused 150M USD loss in ether
- Deployed a hard fork to roll-back the attack
Reentrancy Attacks

Dangers of calling external contracts – can take over control flow.

```solidity
mapping (address => uint) public balances;
function withdraw() public {
    bool success;
    bytes memory data;
    //send ether back to sender address/callback fn.
    (success, data) =
        msg.sender.call.value(balances[msg.sender])("");
    if (!success) {
        revert("Withdraw failed");
    }
    //vulnerable-balance update is behind transfer call
    balances[msg.sender] = 0;
}
```
Reentrancy Attacks

Dangers of calling external contracts – can take over control flow.

```solidity
mapping (address => uint) private balances;
function withdraw() public {
    bool success;
    bytes memory data;
    // send ether back to sender address/callback fn.
    (success, data) =
        msg.sender.call.value(balances[msg.sender])("");
    if (!success) {
        revert("Withdraw failed");
    }
    // vulnerable-balance update is behind transfer call
    balances[msg.sender] = 0;
}

// ----------- attacker.sol
function () public external {
    msg.sender.withdraw();
}
```

© Dr Suen Chun Hui 2019
Integer overflow/underflow

Dangers of calling external contracts – can take over control flow.

```solidity
mapping (address => uint256) public balanceOf;
function transfer(address _to, uint256 _value) {
    require(balanceOf[msg.sender] >= _value);
    balanceOf[msg.sender] -= _value; //can overflow
    balanceOf[_to] += _value;        //can overflow
}
```
Parity Bug – poor deployment

- Parent contract owner was unitialized
  - Allowed for random user to re-init
    
    ```solidity
    modifier only_uninitialized {
      if (m_numOwners > 0) throw; _;
    }
    
    function initWallet(address[] _owners, uint _required, uint _daylimit) {
      only_uninitialized {
        initDaylimit(_daylimit);
        initMultiowned(_owners, _required);
      }
    }
    
    selfdestruct() was accidentally called
    ```

The $280M Ethereum’s Parity bug.

A critical security vulnerability in Parity multi-sig wallet got triggered on 6th November—paralyzing wallets created after the 20th July.

© Dr Suen Chun Hui 2019
Other know attacks, tools

• Other attacks
  • https://consensys.github.io/smart-contract-best-practices/known_attacks/
  • Other reentrancy attacks
  • Front-running (loss of market information)
  • DoS attacks (network layer)

• Security tools:
  • https://consensys.github.io/smart-contract-best-practices/security_tools/
  • Code analyzers: mythril, oyente, etc
  • Code coverage, linting
Smart Contract – pentesting: Some help & tips
Check-effect-interaction rule

- Do conditional \texttt{checks} first (eg. \texttt{require()})
- \textbf{Effect} changes to your variables \& data
- \textbf{Interact} with external contracts
- General rule for preventing re-entrancy attack
- Do not rely on gas depletion to prevent re-entrancy
openzeppelin

- battle-tested library of reusable smart contracts
- install using npm
  `npm install openzeppelin-solidity`

Can be integrated easily with truffle

```solidity
import "openzeppelin-solidity/contracts/token/ERC20/ERC20Mintable.sol";

contract SMUToken is ERC20Mintable {
  string public constant name = "SMU Token";
  string public constant symbol = "SMU";
  uint8 public constant decimals = 18;
}
```
openzeppelin

- Modules:
  - Token (ERC20, ERC721, ERC777)
    - Crowdsale
  - Payment, escrow
  - Math (prevent integer over/underflow)
  - Introspection (ERC165, ERC1820)
  - Cryptographic primitives
  - etc
Published Code is not 100%

- External ABI of bytecode is not verified on etherscan
Smart Contract – pentesting: Hands-on
Re-entrancy hands-on

- pentest_target.sol

- Pentest_attack_template.sol
Solidity hints

Call function:
<contr var>..<mtd name>..value(<eth val>)(<mtd params>);

Call function with payable eth:
<contr var>..<mtd name>(<mtd params>);

Get eth balance:
address(<contr var>).balance

Sender(tx caller) address:
msg.sender

Sender(tx caller) payable value:
msg.value
Re-entrancy hands-on (solution)

- pentest_attacker.sol