Clone 与漏洞

1. Re-Entrancy（DAO）初步判断：可以

```
contract EtherStore {

    uint256 public withdrawalLimit = 1 ether;
    mapping(address => uint256) public lastWithdrawTime;
    mapping(address => uint256) public balances;

    function depositFunds() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdrawFunds (uint256 _weiToWithdraw) public {
        require(balances[msg.sender] >= _weiToWithdraw);
        // limit the withdrawal
        require(_weiToWithdraw <= withdrawalLimit);
        // limit the time allowed to withdraw
        require(now >= lastWithdrawTime[msg.sender] + 1 weeks);
        require(msg.sender.call.value(_weiToWithdraw)());
        balances[msg.sender] -= _weiToWithdraw;
        lastWithdrawTime[msg.sender] = now;
    }
}
```

2. Arithmetic Over/Under Flows 初步判断：可以

```
contract TimeLock {

    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;

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

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

    function withdraw() public {
        require(balances[msg.sender] > 0);
        require(now > lockTime[msg.sender]);
        uint transferValue = balances[msg.sender];
        balances[msg.sender] = 0;
        msg.sender.transfer(transferValue);
    }
}
```

3. Unexpected Ether（不可以）

```
contract EtherGame {

    uint public payoutMileStone1 = 3 ether;
    uint public mileStone1Reward = 2 ether;
    uint public payoutMileStone2 = 5 ether;
    uint public mileStone2Reward = 3 ether;
    uint public finalMileStone = 10 ether;
    uint public finalReward = 5 ether;

    mapping(address => uint) redeemableEther;
    // users pay 0.5 ether. At specific milestones, credit their accounts
    function play() public payable {
        require(msg.value == 0.5 ether); // each play is 0.5 ether
        uint currentBalance = this.balance + msg.value;
        // ensure no players after the game as finished
        require(currentBalance <= finalMileStone);
        // if at a milestone credit the players account
        if (currentBalance == payoutMileStone1) {
            redeemableEther[msg.sender] += mileStone1Reward;
        }
        else if (currentBalance == payoutMileStone2) {
            redeemableEther[msg.sender] += mileStone2Reward;
        }
        else if (currentBalance == finalMileStone ) {
            redeemableEther[msg.sender] += finalReward;
        }
        return;
    }

```

4. Delegatecall

```
contract FibonacciBalance {

    address public fibonacciLibrary;
    // the current fibonacci number to withdraw
    uint public calculatedFibNumber;
    // the starting fibonacci sequence number
    uint public start = 3;
    uint public withdrawalCounter;
    // the fibonancci function selector
    bytes4 constant fibSig = bytes4(sha3("setFibonacci(uint256)"));

    // constructor - loads the contract with ether
    constructor(address _fibonacciLibrary) public payable {
        fibonacciLibrary = _fibonacciLibrary;
    }

    function withdraw() {
        withdrawalCounter += 1;
        // calculate the fibonacci number for the current withdrawal user
        // this sets calculatedFibNumber
        require(fibonacciLibrary.delegatecall(fibSig, withdrawalCounter));
        msg.sender.transfer(calculatedFibNumber * 1 ether);
    }

    // allow users to call fibonacci library functions
    function() public {
        require(fibonacciLibrary.delegatecall(msg.data));
    }
}

```

5. Default Visibilities（初步判断：可以）

```
contract HashForEther {

    function withdrawWinnings() {
        // Winner if the last 8 hex characters of the address are 0.
        require(uint32(msg.sender) == 0);
        _sendWinnings();
     }

     function _sendWinnings() {
         msg.sender.transfer(this.balance);
     }
}
```

7. External Contract Referencing（可以）

```
import "Rot13Encryption.sol";

// encrypt your top secret info
contract EncryptionContract {
    // library for encryption
    Rot13Encryption encryptionLibrary;

    // constructor - initialise the library
    constructor(Rot13Encryption _encryptionLibrary) {
        encryptionLibrary = _encryptionLibrary;
    }

    function encryptPrivateData(string privateInfo) {
        // potentially do some operations here
        encryptionLibrary.rot13Encrypt(privateInfo);
     }
 }
```

```
contract Rot13Encryption {

    event Result(string convertedString);

    //rot13 encrypt a string
    function rot13Encrypt (string text) public {
        uint256 length = bytes(text).length;
        for (var i = 0; i < length; i++) {
            byte char = bytes(text)[i];
            //inline assembly to modify the string
            assembly {
                char := byte(0,char) // get the first byte
                if and(gt(char,0x6D), lt(char,0x7B)) // if the character is
                { char:= sub(0x60, sub(0x7A,char)) } // subtract from the as
                if iszero(eq(char, 0x20)) // ignore spaces
                {mstore8(add(add(text,0x20), mul(i,1)), add(char,13))} // ad
            }
        }
        emit Result(text);
    }

    // rot13 decrypt a string
    function rot13Decrypt (string text) public {
        uint256 length = bytes(text).length;
        for (var i = 0; i < length; i++) {
            byte char = bytes(text)[i];
            assembly {
                char := byte(0,char)
                if and(gt(char,0x60), lt(char,0x6E))
                { char:= add(0x7B, sub(char,0x61)) }
                if iszero(eq(char, 0x20))
```

9. Unchecked CALL Return Values（可以）

```
contract Lotto {

    bool public payedOut = false;
    address public winner;
    uint public winAmount;

    // ... extra functionality here

    function sendToWinner() public {
        require(!payedOut);
        winner.send(winAmount);
        payedOut = true;
    }

    function withdrawLeftOver() public {
        require(payedOut);
        msg.sender.send(this.balance);
    }
}
```

10. Race Conditions / Front Running（不可以）

```
contract FindThisHash {
    bytes32 constant public hash = 0xb5b5b97fafd9855eec9b41f74dfb6c38f595114

    constructor() public payable {} // load with ether

    function solve(string solution) public {
        // If you can find the pre image of the hash, receive 1000 ether
        require(hash == sha3(solution));
        msg.sender.transfer(1000 ether);
    }
}
```

11. Denial Of Service (DOS)

```
contract TrickleWallet {

    address public partner; // withdrawal partner - pay the gas, split the w
    address public constant owner = 0xA9E;
    uint timeLastWithdrawn;
    mapping(address => uint) withdrawPartnerBalances; // keep track of partn

    function setWithdrawPartner(address _partner) public {
        require(partner == '0x0' || msg.sender == partner);
        partner = _partner;
    }

    // withdraw 1% to recipient and 1% to owner
    function withdraw() public {
        uint amountToSend = address(this).balance/100;
        // perform a call without checking return
        // the recipient can revert, the owner will still get their share
        partner.call.value(amountToSend)();
        owner.transfer(amountToSend);
        // keep track of last withdrawal time
        timeLastWithdrawn = now;
        withdrawPartnerBalances[partner] += amountToSend;
    }

    // allow deposit of funds
    function() payable {}

    // convenience function
    function contractBalance() view returns (uint) {
        return address(this).balance;
```

12. Block Timestamp Manipulation（不可以）

```
contract Roulette {
    uint public pastBlockTime; // Forces one bet per block

    constructor() public payable {} // initially fund contract

    // fallback function used to make a bet
    function () public payable {
        require(msg.value == 10 ether); // must send 10 ether to play
        require(now != pastBlockTime); // only 1 transaction per block
        pastBlockTime = now;
        if(now % 15 == 0) { // winner
            msg.sender.transfer(this.balance);
        }
    }
}
```

13. Constructors with Care

```
contract OwnerWallet {
    address public owner;

    //constructor
    function ownerWallet(address _owner) public {
        owner = _owner;
    }

    // fallback. Collect ether.
    function () payable {}

    function withdraw() public {
        require(msg.sender == owner);
        msg.sender.transfer(this.balance);
    }
}
```

15. Floating Points and Precision（可以）

```
contract FunWithNumbers {
    uint constant public tokensPerEth = 10;
    uint constant public weiPerEth = 1e18;
    mapping(address => uint) public balances;

    function buyTokens() public payable {
        uint tokens = msg.value/weiPerEth*tokensPerEth; // convert wei to et
        balances[msg.sender] += tokens;
    }

    function sellTokens(uint tokens) public {
        require(balances[msg.sender] >= tokens);
        uint eth = tokens/tokensPerEth;
        balances[msg.sender] -= tokens;
        msg.sender.transfer(eth*weiPerEth); //
    }
}
```

16. Tx.Origin Authentication（可以）

```
contract Phishable {
    address public owner;

    constructor (address _owner) {
        owner = _owner;
    }

    function () public payable {} // collect ether

    function withdrawAll(address _recipient) public {
        require(tx.origin == owner);
        _recipient.transfer(this.balance);
    }
}
```