In Math，We Trust！

攻击角度：Move语言安全性分析

分享人：Amos

日期：2023-01-12

十种常见攻击/漏洞

SharkTeam

1. 重入攻击：语言层面
2. 权限漏洞：语言+业务层面
3. 逻辑校验漏洞：语言+业务层面
4. 函数恶意初始化：语言层面
5. 回退攻击：语言层面
6. 提案攻击：业务层面
7. 合约升级漏洞：语言+区块链层面
8. 操纵价格预言机：业务层面
9. 三明治攻击：区块链层面
10. 重放攻击：区块链层面

1. 重入攻击

SharkTeam

重入攻击

（1）动态的外部调用call+回调函数fallback

（2）先外部调用，后修改状态，绕过某些检查

（3）破坏“外部调用+修改状态”的原子性

```
contract ERCToken{
    function withdraw(uint amount) public returns(bool) {
        if (credit[msg.sender]>= amount) {      //检查
            msg.sender.call.value(amount)();    //交互
            credit[msg.sender]-=amount;         //生效
            emit Withdraw(msg.sender, amount);
            return true;
        }
        return false;
    }
}
```

预防措施

（1）使用“检查-生效-交互”模式

（2）添加重入锁，不允许函数重入

（3）避免直接使用call函数， 比如：

转账时使用transfer()代替call.value()

动态调用 & 静态调用

（1）动态调用：在执行之前的合约编译阶段，并不会加载和编译外部合约的代码，在执行到调用的外部函数时会跳转到被调用函数所在上下文中进行执行。

（2）静态调用：在编译阶段，将调用的函数加载到当前环境，与合约交互时，根据合约按顺序进行执行，不会发生跳转和动态加载。

因此，动态调用是重入发生的前提，静态调用的过程不可能发生重入。

Move是静态语言，没有重入的条件。

1. 重入攻击

SharkTeam

- 破坏原子性操作(Move)

```
struct Coin has store {
    value: u64
}
struct Balance has key {
    coin: Coin
}
/// Transfers `amount` of tokens from `from` to `to`.
public fun transfer(from: &signer, to: address, amount: u64) acquires Balance {
    let check = withdraw(signer::address_of(from), amount);
    deposit(to, check);
}
fun withdraw(addr: address, amount: u64) : Coin acquires Balance {
    let balance = balance_of(addr);
    // balance must be greater than the withdraw amount
    assert!(balance >= amount, EINSUFFICIENT_BALANCE);
    let balance_ref = &mut borrow_global_mut<Balance>(addr).coin.value;
    // *balance_ref = balance - amount;
    Coin { value: amount }
}
fun deposit(addr: address, check: Coin) acquires Balance{
    let balance = balance_of(addr);
    let balance_ref = &mut borrow_global_mut<Balance>(addr).coin.value;
    let Coin { value } = check;
    *balance_ref = balance + value;
}
```

2. 权限漏洞

SharkTeam

权限漏洞

权限漏洞指在检查授权时存在纰漏，使得攻击者在获得低权限用户账户后，利用一些方式绕过权限检查，访问或者操作其他用户或者窃取了更高的权限。

智能合约中的权限漏洞和关键逻辑有关，例如代币的铸造、提取和销毁，更改所有权等。

权限漏洞类型

（1）函数默认可见性

Solidity：public、external、internal、private，默认是 public

Move：public(script)/entry、public、public(friend)、private，默认是 private

```
/// Transfers `amount` of tokens from `from` to `to`.
public fun transfer(from: &signer, to: address, amount: u64) acquires Balance {
    let check = withdraw(signer::address_of(from), amount);
    deposit(to, check);
}
public fun withdraw(addr: address, amount: u64) : Coin acquires Balance {
    let balance = balance_of(addr);
    // balance must be greater than the withdraw amount
    assert!(balance >= amount, EINSUFFICIENT_BALANCE);
    let balance_ref = &mut borrow_global_mut<Balance>(addr).coin.value;
    *balance_ref = balance - amount;
    Coin { value: amount }
}
public fun deposit(addr: address, check: Coin) acquires Balance{
    let balance = balance_of(addr);
    let balance_ref = &mut borrow_global_mut<Balance>(addr).coin.value;
    let Coin { value } = check;
    *balance_ref = balance + value;
}
```

2. 权限漏洞

SharkTeam

权限漏洞类型

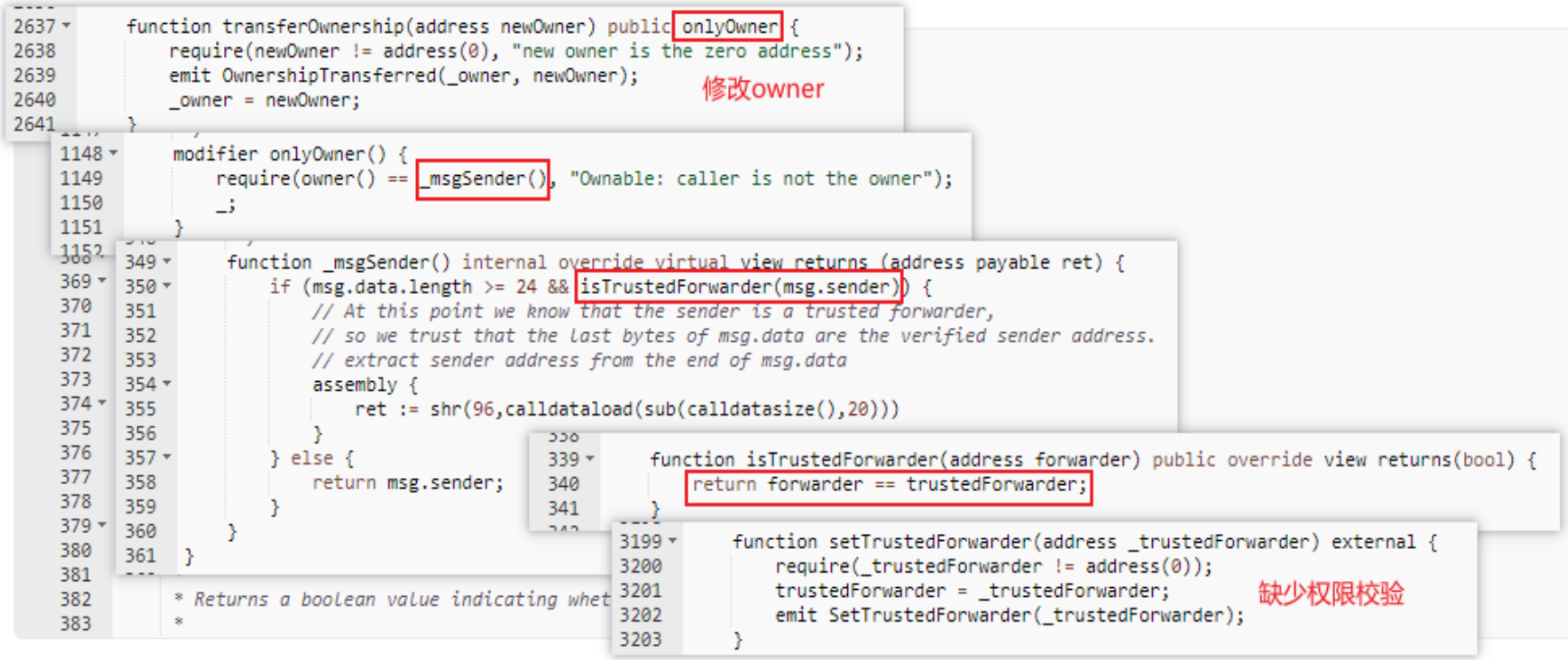
（2）缺少modifier验证或验证存在错误或漏洞

Solidity：modifier、require、assert

Move：spec、acquires、assert

权限漏洞攻击事件

2022年1月18日，BSC上的Crosswise项目因权限漏洞遭受到黑客攻击。黑客仅用 1 个 CRSS token 便获取 Crosswise MasterChef 池中价值 87.9 万美元的 692K 个 CRSS.



2. 权限漏洞

SharkTeam

权限漏洞类型

（3）tx.origin身份验证

Solidity：tx.origin作为身份验证凭据，容易被攻击者通过社会工程学诱骗owner签署攻击交易，从而绕过owner身份验证；安全考虑，建议减少使用tx.origin，尽量使用 msg.sender

Move：静态调用，没有上下文调用的概念，没有这个漏洞

（4）初始化函数问题

Solidity：自定义初始化函数，msg.sender 和 initializer 校验（调用者和调用次数）

Move：自定义初始化函数，signer 校验以及资源的唯一性（初始化函数创建资源，如Token、Balance等）

（5）调用自毁(selfdestruct)

Solidity：selfdestruct(address payable addr) 函数可以销毁当前合约，并且把当前合约的ETH余额发送给指定地址addr，需要权限限制，即msg.sender校验

Move：不存在自毁函数，没有这个漏洞

3. 逻辑校验漏洞

SharkTeam

逻辑校验漏洞

智能合约开发的业务相关逻辑设计复杂，涉及的经济学计算和参数较多，不同项目和协议之间可组合性极其丰富，很难预测，非常容易出现安全漏洞。

逻辑校验漏洞类型

（1）未校验返回值

Solidity：call函数返回值(bool success, bytes memory returndata)校验，

A. 校验success，require(success, “”)

B. 校验returndata，业务层面需求

```
// solium-disable-next-line security/no-call-value
(bool success, bytes memory returnData)
    = target.call{value: value}(callData);
require(success, "Transaction execution reverted.");
```

Move：发生意外则报错回退；业务层面需求

（2）未校验相关计算数据

Solidity & Move ： 业务层面，复杂的业务逻辑、数据结构、经济模型、计算公式等

A. 业务逻辑存在漏洞

B. 代码实现存在漏洞

3. 逻辑校验漏洞

SharkTeam

逻辑校验漏洞

2021年7月2日，火币生态链（Heco）上DeFi项目XDXSwap受到闪电贷攻击，损失约400万美金。

File 1 of 8 : UniswapV2Pair.sol

```
198
199     // this low-level function should be called from a contract which performs important safety checks
200     function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
201         require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
202         (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
203         require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');
204
205         uint balance0;
206         uint balance1;
207         { // scope for _token{0,1}, avoids stack too deep errors
208         address _token0 = token0;
209         address _token1 = token1;
210         require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
211         if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
212         if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
213         if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender, amount0Out, amount1Out, data);
214         balance0 = IERC20Uniswap(_token0).balanceOf(address(this));
215         balance1 = IERC20Uniswap(_token1).balanceOf(address(this));
216         }
217         uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
218         uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
219         require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
220         { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
221         // uint balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(25)); // 3-->2
222         // uint balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(25));// 3-->2
223         // require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(10000**2), 'UniswapV2: K');
224         }
225
226         _update(balance0, balance1, _reserve0, _reserve1);
227         emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
228     }
229
```

保证token0和token1中至少有1个是有输入的，但不检查输入多少

检查 输入金额≥借款金额+兑换费用

3. 逻辑校验漏洞

SharkTeam

逻辑校验漏洞类型

（3）未校验函数参数

Solidity：数组长度校验，地址（类型、白名单等）校验，msg校验，call参数校验（包括函数签名），业务层面范围校验等

Move：signer校验；业务层面参数校验

```
// solium-disable-next-line security/no-call-value
(bool success, bytes memory returnData)
    = target.call{value: value}(callData);
require(success, "Transaction execution reverted.");
```

（4）未规范使用require（未校验存在性/支持性逻辑）

Solidity & Move：

A. 转账：校验余额是否足够支持转账；Token、Balance等资源是否存在(Move)

B. 借贷：抵押物是否存在；抵押物价值是否支持借款数量

C. 闪电贷：矿池余额是否支持闪电贷，交易结束前是否已经还款，还款金额是否正确

4. 函数恶意初始化

SharkTeam

函数恶意初始化

实现合约的初始化有两种方式：

（1）自动执行方式，如Solidity中的构造函数，SUI Move中的init函数

（2）自定义方式，如Solidity代理升级模式，Aptos Move

Solidity函数恶意初始化

Punk Protocol 安全事件：

2021年8月10日， Punk Protocol被攻击，损失890万美元，后来又找回了495万美元；

原因：合约中的自定义初始化函数可以被多次调用

预防措施

（1）限制调用者身份，即：Solidity校验msg.sender，Move校验signer

（2）限制调用次数（仅可以调用一次），如 Openzeppelin 中的 Initializer，Move资源的唯一性

安全事件

2021年8月10日，Punk Protocol 遭遇攻击，造成 890 万美元损失，后来团队又找回了 495 万美元。

攻击原因在于投资策略中找到了一个关键漏洞：CompoundModel 代码中缺少初始化函数的修饰符的问题，可以被重复初始化。

```
23      function initialize(
24          address forge_,
25          address token_,
26          address cToken_,
27          address comp_,
28          address comptroller_,
29          address uRouterV2_ ) public {
30
31              addToken( token_ );
32              setForge( forge_ );
33              _cToken          = cToken_;
34              _comp            = comp_;
35              _comptroller     = comptroller_;
36              _uRouterV2       = uRouterV2_;
37
38      }
```

5. 回退攻击

SharkTeam

回退(revert)攻击

回退攻击指根据操作结果来决定交易是否执行，如果操作结果没有达到攻击者的预期，攻击者会让交易回退。

回退攻击的条件

（1）随机性，即结果具有随机性，不同的结果获利不同。

（2）有利可图，即至少有一种结果能让攻击者获利。

（3）可编程性（合约调用合约），可以通过自定义合约函数来发起回退攻击。

预防措施

（1）限制调用者身份，不允许合约或脚本(Move）调用

Solidity：require(msg.sender==tx.orgin, “not EOA”);

Move：私有的入口函数 entry

（2）不在合约中使用随机性

6. 提案攻击

提案攻击

提案攻击针对的是去中心化自治组织 （DAO）。

若某用户通过某些途径获取大量的治理代币，从而获得DAO的绝对控制权（即不需要其他人投票既可以通过提案），那么该用户就具备了发起提案攻击的条件。

提案攻击发生在DAO中，应用了DAO的所有项目都有可能发生提案攻击，跟开发语言无关。因此，在Move生态中，使用了DAO的项目同样需要提防提案攻击。

提案攻击的安全事件

（1）2022年4月17日，以太坊上面的算法稳定币项目Beanstalk Farms遭受到了提案攻击。攻击者通过闪电贷获得了大量的治理代币，窃取了DAO的绝对控制权。

（2）2022年5月9日，币安智能链的Fortress Loans遭到提案攻击。DAO的治理代币价格极低，攻击者仅仅使用9 ETH就兑换到了超过DAO投票阈值（400000）的治理代币，获取了DAO的绝对控制权。

6. 提案攻击

SharkTeam

预防措施

（1）降低攻击的价值，限制提案治理的范围

（2）增加获得投票权的成本，间接降低治理代币的流动性，项目可以提供激励措施

（3）增加执行攻击的成本，设计人员需要某种用户身份验证才能参与投票

（4）技术层面的安全保护：避免闪电贷，请第三方合约审计团队对合约进行审计

安全的DAO

（1）不支持通过闪电贷在一笔交易内获得的临时的治理代币用于投票

（2）治理代币价格要高，避免以很低的价格获得大量的票数

（3）治理代币不能集中在极少数人手中

（4）治理的投票阈值要合理，而且要随着治理代币的总量不断更新

7. 合约升级漏洞

SharkTeam

Solidity代理升级模式

在以太坊中，智能合约有多重升级模式，其中最主要的就是代理升级模式。使用代理合约和逻辑合约来实现存储数据与业务逻辑的分离模式。通过代理合约调用逻辑合约中的函数，访问的是代理合约的存储数据。

代理升级模式兼容性漏洞：兼容性

（1）存储的兼容性，逻辑合约中定义的状态变量与代理合约中的状态变量的冲突。

（2）升级的兼容性，合约升级后，新合约的数据处理逻辑与原来的数据不兼容，比如精度等。

Move合约升级

（1）Starcoin和Aptos支持合约升级，SUI不支持合约升级

（2）选择合适的升级方案和升级策略

（3）考虑模块、资源、公共的API的兼容性

（4）如果升级方案选择DAO模式，还要考虑DAO的安全性。

8. 操纵价格预言机

SharkTeam

（1）预言机(Oracle)是链接链上智能合约与链下现实环境的桥梁。

价格预言机则是链上智能合约获取代币链上或链下价格的桥梁工具。

（2）价格预言机类型：

A. 链上价格预言机

B. 链下价格预言机

C. 中心化价格预言机

D. 去中心化价格预言机

（3）操纵价格预言机是最常见的一种经济攻击手段，尤其是对于链上价格预言机。

（4）操纵价格预言机有两种方式：

A. 闪电贷操纵链上的AMM价格

B. 预言机故障导致随机攻击

（5）**预防措施**

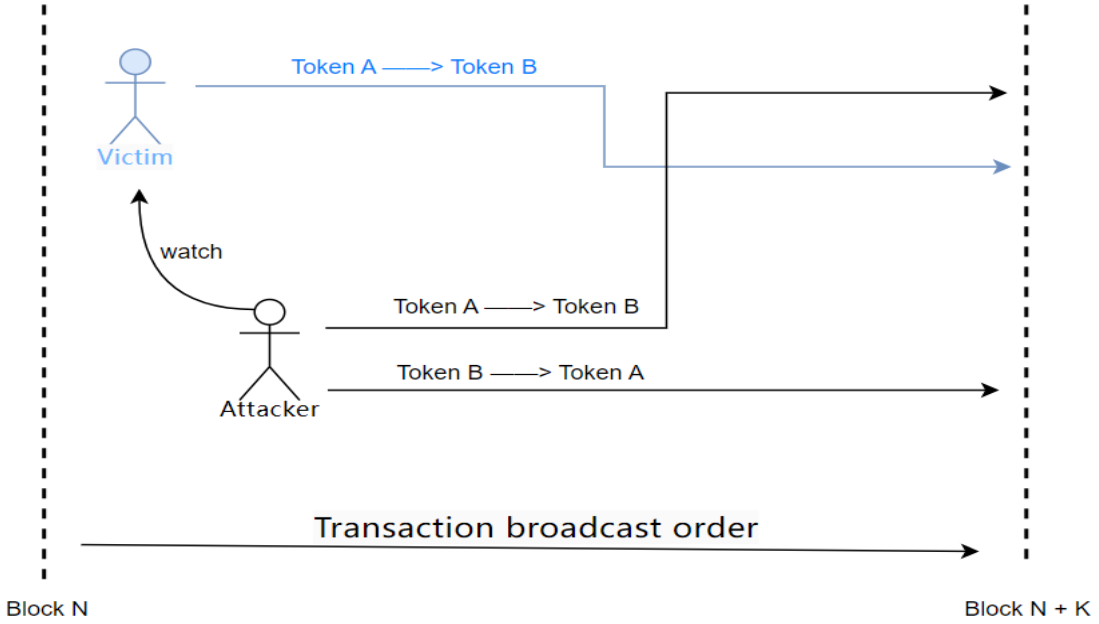
A. 使用时间加权平均价格

B. 设置减速带（时间延迟/时间锁定）

C. 使用链下的去中心化价格预言机

9. 三明治攻击

三明治攻击（Sandwich Attack）是一种抢先交易。在三明治攻击中，恶意交易者在他们选择的网络上寻找待处理交易，三明治攻击的发生是在交易前和交易后各下一个订单。从本质上讲，攻击者会同时进行前置(front-run)和后置(back-run) 交易，而原来的待处理交易则夹在中间。



1. 攻击者提前洞察了受害者的交易意图，认为有利可图，提前通过利用10个TokenA兑换了100个TokenB，同时抬高了TokenB的价格
2. 受害者在不知情的情况下，通过意外价格滑点，以5个TokenA兑换了40个TokenB，原本5个TokenA可以兑换50个TokenB
3. 攻击者再将100Token B兑换成12.5个TokenA，获利2.5个TokenA

预防措施：

（1）Gas限制

（2）避免低流动性池

（3）小额交易

10. 重放攻击

SharkTeam

重放攻击 (Replay Attacks)是一种传统网络攻击手段，又称重播攻击、回放攻击，指攻击者发送一个目的主机已接收过的包，来达到欺骗系统的目的，主要用于身份认证过程，破坏认证的正确性。

在区块链领域中，重放攻击是重复利用签名的攻击手段，我们将重放攻击分为**交易重放**和**签名重放。**

（1）**交易重放**指重复利用交易签名，将原链上的交易一成不变放到目标链上，重放过后交易在目标链上可以正常执行并完成交易验证。

（2）**签名重放**指重复利用交易数据中的私钥签名进行重放，重放过程中无需像交易重放那样去重放整个交易，而是重放相应的签名信息。

在实施EIP-155后，交易签名带有chainId，即链与分叉链之间的标识符。由于chainId不同，交易重放无法完成，签名重放可以间接完成。

预防措施：

- 可以在签名消息中加入chainid和nonce两个参数值，chainid用于识别链ID的标识符，nonce是交易次数计数值；
- 记录签名是否使用过，比如利用mapping进行签名中对应参数映射为bool值，这样做可以防止签名多次使用；
- 项目上线前，需联系第三方专业审计团队进行审计。

In Math，We Trust！

Thanks

- www.sharkteam.org
- https://t.me/sharkteamorg
- https://twitter.com/sharkteamorg