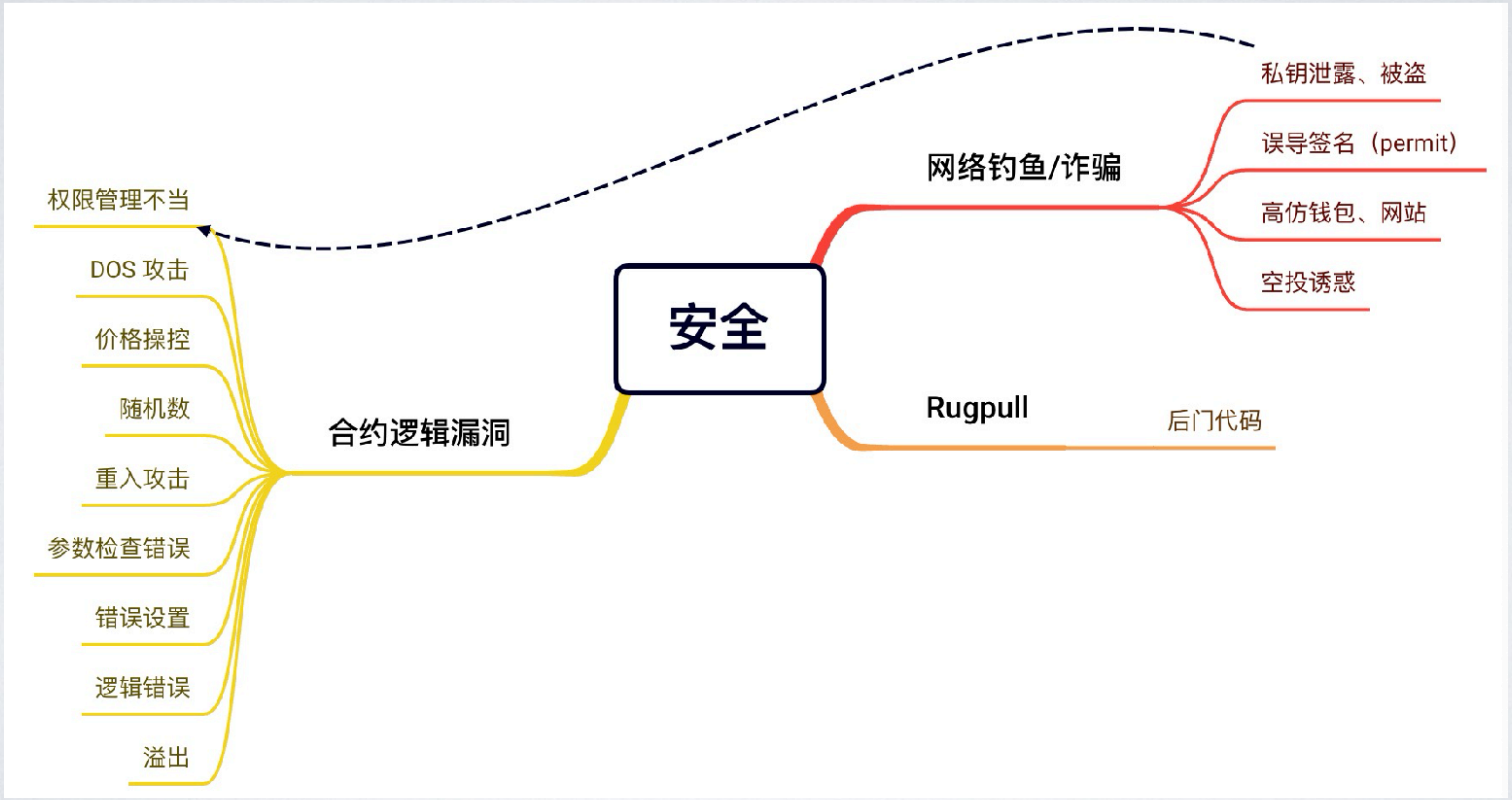
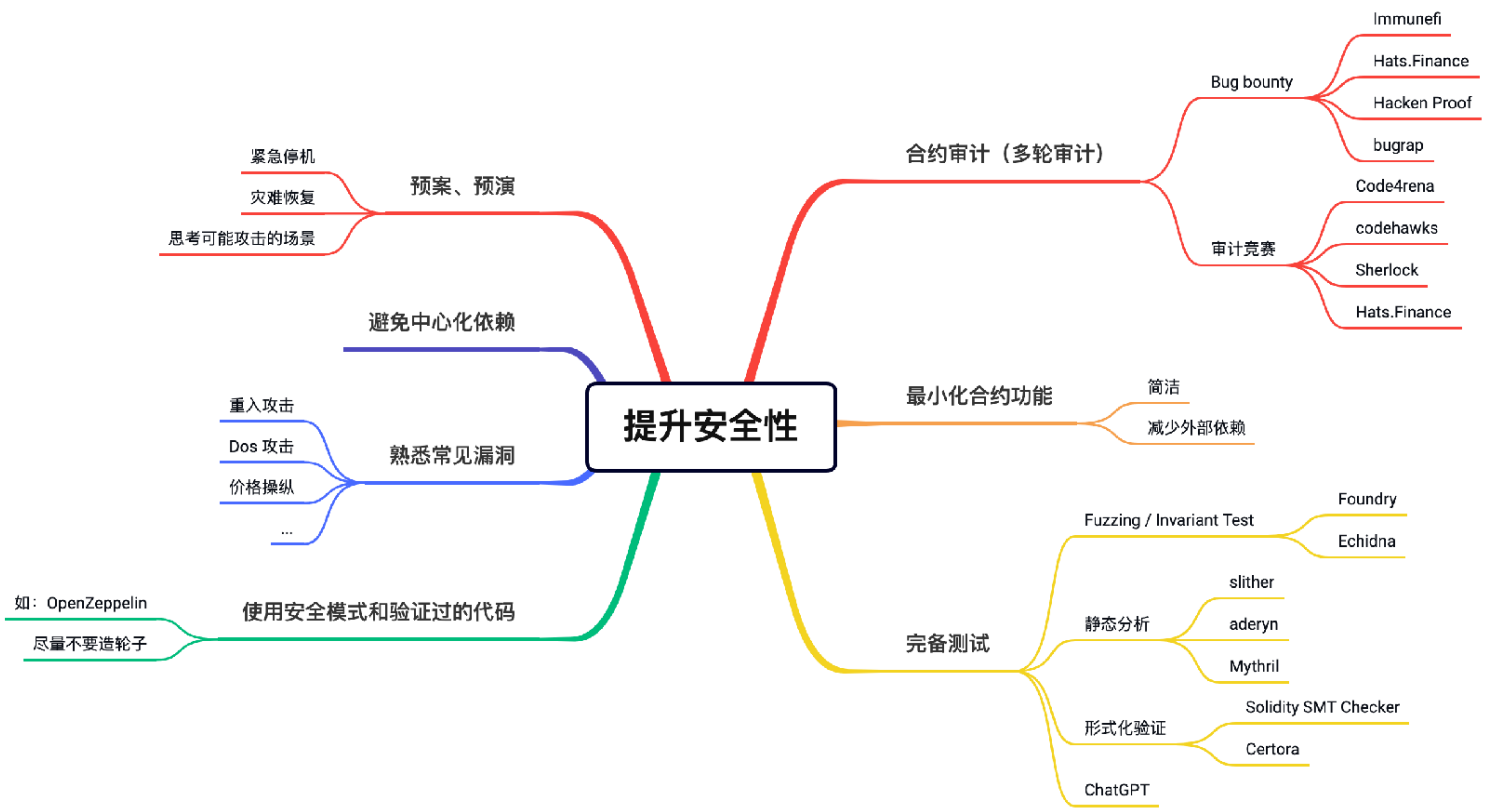
Solidity 安全

安全的重要性

- 合约不可修改 + 直接处理资金
- 每年因安全问题，导致数十亿的美元的资金损失
- 让区块链安全比 Web2 安全成为一个更突出的问题

常见安全问题


安全
合约逻辑漏洞
权限管理不当
DOS 攻击
价格操控
随机数
重入攻击
参数检查错误
错误设置
逻辑错误
溢出
网络钓鱼/诈骗
私钥泄露、被盗
误导签名（permit）
高仿钱包、网站
空投诱惑
Rugpull
后门代码


提升安全性
预案、预演
紧急停机
灾难恢复
思考可能攻击的场景
避免中心化依赖
熟悉常见漏洞
重入攻击
Dos 攻击
价格操纵
...
使用安全模式和验证过的代码
如：OpenZeppelin
尽量不要造轮子
合约审计（多轮审计）
Bug bounty
Immunefi
Hats.Finance
Hacken Proof
bugrap
审计竞赛
Code4rena
codehawks
Sherlock
Hats.Finance
最小化合约功能
简洁
减少外部依赖
完备测试
Fuzzing / Invariant Test
Foundry
Echidna
静态分析
slither
aderyn
Mythril
形式化验证
Solidity SMT Checker
Certora
ChatGPT

常见漏洞

- 重入攻击
- Dos 拒绝服务
- 签名重用
- 溢出（Solidity < 0.8）、 精度损失
- 合约账户控制

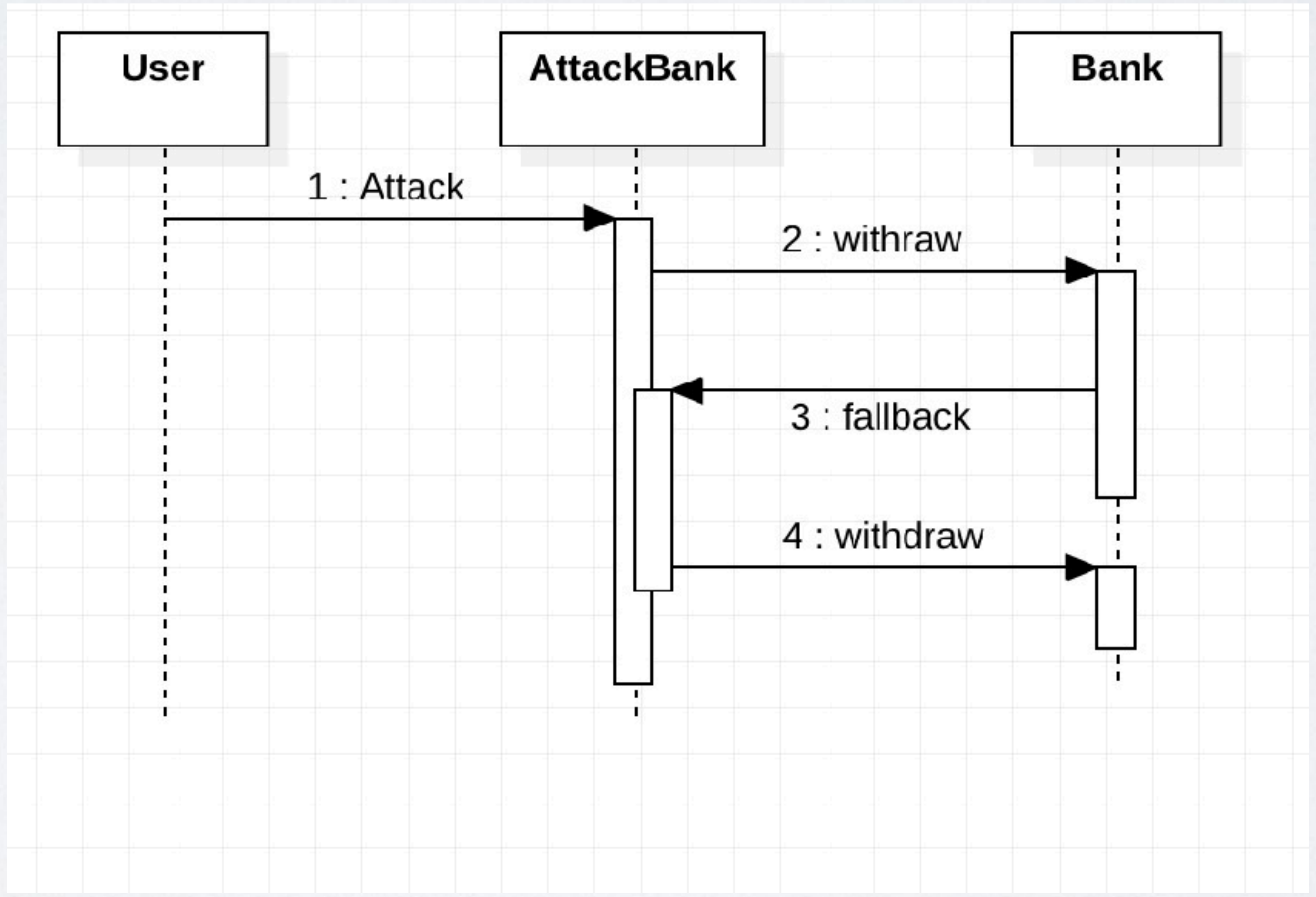
找问题

```
function withdraw() public {
    (bool success, ) = msg.sender.call{value: deposits[msg.sender]}("");
    deposits[msg.sender] = 0;

    require(success, "Failed to send Ether");
}
```

重入攻击

- 调用外部函数时，要时刻注意重入问题：
  - 常见的有： 带 hook 的转账： ERC777、 ERC3613
- 重入



testReplay.sol

防范重入攻击问题

- 先检查 - 再修改 - 最后交互（checks-effect-interaction）
- 重入锁控制

testReplay.sol

瞬时存储

- 2024/03 引入的新操作码，引入了一个新存储空间（瞬时存储），读写更便宜

- 对该存储的修改仅在一个交易内有效

```
bool transient locked;

modifier nonReentrant {
    require(!locked, "Reentrancy attempt");
    locked = true;
    _;
    // 函数退出后，可以再次调用它，即使是在同一交易中。
    locked = false;
}
```

找问题

```
function enter() public {
    // Check for duplicate entrants
    for (uint256 i; i < entrants.length; i++) {
        if (entrants[i] == msg.sender) {
            revert("You've already entered!");
        }
    }
    entrants.push(msg.sender);
}
```

找问题

没有 receive
fallback

```
    uint256 public totalDeposits;
    mapping(address => uint256) public deposits;

    function deposit() external payable {
        deposits[msg.sender] += msg.value;
        totalDeposits += msg.value;
    }

    function withdraw() external {
        assert(address(this).balance == totalDeposits);

        uint256 amount = deposits[msg.sender];
        totalDeposits -= amount;
        deposits[msg.sender] = 0;

        payable(msg.sender).transfer(amount);
   }
```

找问题

- selfdestruct(recipient)
- 验证者 fee_recipient

https://prysm.offchainlabs.com/docs/configure-prysm/fee-recipient/

找问题

```
bytes32 public constant TYPEHASH = keccak256(“withdrawBySig(uint256 amount)”);


function withdrawBySig(uint8 v, bytes32 r, bytes32 s, uint256 amount) external payable {
    bytes32 structHash = keccak256(abi.encode(TYPEHASH, amount));
    bytes32 hash = _hashTypedDataV4(structHash);
    address signer = ECDSA.recover(hash, v, r, s);
    require(inWhitelist[signer],  "error signer");
    _withdraw(signer, amount);
}

function _withdraw(address user, uint256 amount) internal {
    uint256 currentBalance = balances[user];
    if (currentBalance < amount) {
        revert SignatureReplay__InsufficientBalance(currentBalance, amount);
    }
    balances[user] = currentBalance - amount;
    payable(msg.sender).transfer(amount);
}
```

找问题

```
uint256 public moneyToSplitUp = 225;
uint256 public users = 4;
uint count;

function shareMoney() public view returns (uint256) {
    return moneyToSplitUp / users;
}

function decrement() public {
    unchecked {
        count--;
    }
}
```

找问题

```
function isContract(address account) public view returns (bool) {
    uint size;
    assembly {
        size := extcodesize(account)
    }
    return size > 0;
}

// 确保仅有 EOA 能调用
function protected() external {
    require(!isContract(msg.sender), "no contract allowed”);
    ………

}
```

CTF

- https://ethernaut.openzeppelin.com/
- https://capturetheether.com/
- https://www.damnvulnerabledefi.xyz/
- https://ciphershastra.com/Maya.html

最佳实践

- https://consensys.github.io/smart-contract-best-practices/

作业

- 尝试盗取Vault 中的资金
- https://decert.me/quests/b5368265-89b3-4058-8a57-a41bde625f5b