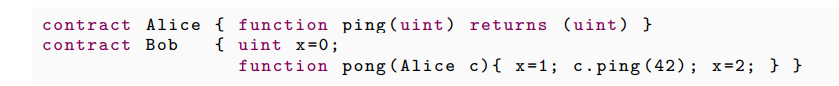
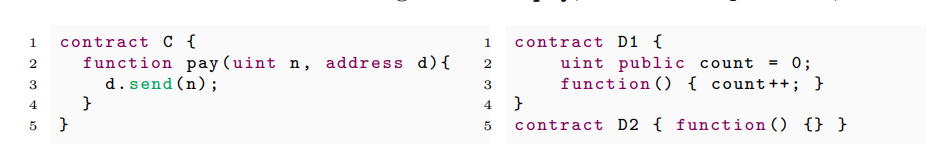
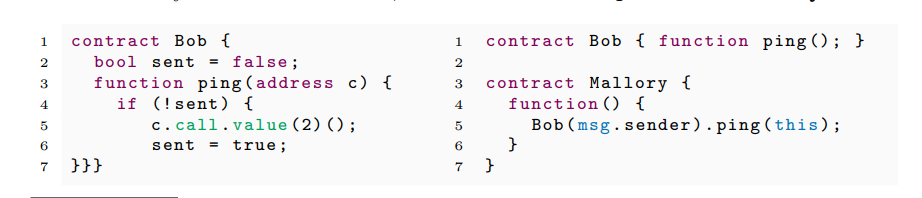
1. **A Survey of Attacks on Ethereum Smart Contracts (SoK) 2017**
   1. 主要观点
      1. 列举漏洞种类和漏洞案例分析
   2. 漏洞类型
      1. **Call to the unknown**
         1. 合约名.函数名（参数）：直接调用（direct call），如果函数的参数错误（int传给uint），则被调用合约的回退函数被执行
      2. **Exception disorder**
         1. 对比
            1. 外部用户调用Bob的pong，alice的ping抛出异常，执行停止，并且交易回退，x=0
            2. pong函数直接调用ping，只有一边被revert，pong函数继续执行，x=2
         2. 异常的类型：gas耗尽，堆栈溢出，throw命令被执行
         3. 嵌套调用链中出现异常之后的两种处理方式：
            1. 如果链中每个元素都是直接调用，则执行停止，所有执行包括转账都恢复，但gas都被消耗掉了
            2. 如果链中有一个call（call ,delegatecall或send），那么异常将沿着链传播，直到遇到上一个call停止，返回false，并从这个call开始继续执行
         4. 处理异常的方式不规范会导致安全问题
            1. 不检测转账函数是否执行成功——ether留在原账户
            2. 检测但没有正确处理——恶意合约故意抛出异常，导致dos攻击
      3. **gasless send**



* + - 1. 三种情况
         1. n不等于0，d等于D1，fail with out-of-gas exception
         2. n不等于0，d等于D2，succeed
         3. n等于0，小于0.4.0 fail，大于等于0.4.0，same as previous two
      2. send发送ether成功的情况：接收者回退函数不需要太多gas；接收者是一个用户。
    1. **Type casts**
       1. 编译器不会检查地址是否是符合声明合约
       2. 编译器不会检查对外部合约函数的调用是否合法
       3. 如果调用的合约地址c无效，返回并不执行任何代码
       4. 如果调用的合约函数不存在，调用该合约的回退函数
       5. 如果调用的合约函数的签名不合法，则调用该合约的回退函数
    2. **Reentrancy**
       1. 转账语句触发回调函数，回调函数可以再次调用转账函数，导致重入
       2. 每次回调都会触发异常，但是调用 call 不传播异常，所以只有最后一次转账call因为gas耗尽而产生异常，被回退，之前的转账都是有效的。
    3. **Keeping secrets**
       1. 私有参数虽然可以设置不可见，但是区块链是公开的，如果给私有参数赋值，其他人可以监测到交易中包含的参数赋值内容
    4. **Immutable bugs（EVM）**
       1. 合约一旦发布不可更改，但是如果合约有漏洞被攻击了，那么程序员必须变更或终止合同。
       2. THE DAO攻击就是利用了不变性，最终导致的硬分叉至今还存在争议
    5. **Ether lost in transfer（EVM）**
       1. 有些地址是孤立的，一旦将以太币发送过去就永远丢失，所以程序员必须确保接收者地址的正确性
    6. **Stack size limit（EVM）**
       1. 每次合约调用另一个合约，或者调用自身，与交易相关的堆栈就会增加一帧，调用堆栈被限制为1024帧，如果超出这个限制会抛出异常。
    7. **Unpredictable state（Blockchain）**
       1. 长分支取代短分支导致短分支的交易无效
       2. 调用动态更新的合约——攻击者伪造被调用合约
    8. **Generating randomness（Blockchain）**
       1. 生成随机数的方法
          1. 选择某个块的时间戳为种子——矿工可以产生一定的影响
          2. 定时承诺协议：泄露秘密就会丢失押金——不合理
    9. **Time constraints（Blockchain）**
       1. 程序会使用时间约束，但是矿工可以控制时间戳

1. **Smart Contract Vulnerabilities: Does Anyone Care?**
   1. 主要观点
      1. 总结现有漏洞检测工具的结果，通过基于Datalog的公式，对EVM执行轨迹进行分析，对漏洞进行检测和验证，结果发现，现有的漏洞研究夸大了危险性。
   2. 漏洞类型
      1. **RE (re-entrancy)**
      2. **UE (Unhandled exceptions)**
      3. **LE (Locked Ether)**
         1. 具有存款功能的合约，由于某些原因不能取款
      4. **TO (Transaction Order Dependency)**
         1. 在正常交易未被确认之前，攻击者发起影响正常交易金额的交易，并设置高gas使得矿工先确认自己的交易，完成对正常交易的攻击。
      5. **IO (Integer overflow)**
         1. 整数溢出可能导致合约被冻结
2. **Defining Smart Contract Defects on Ethereum**
   1. 主要观点
      1. 从安全性、可用性、性能、可维护性和可重用性，五个方面定义智能合约的漏洞，20种漏洞类型，帮助开发人员更好的理解缺陷。
   2. 漏洞类型
      1. **Security Defects 安全性 9种**
         1. **Unchecked External Calls**
            1. 建议：用transfer函数替代send和call.value函数
         2. **Dos Under External Influence**
            1. 建议：防止攻击者故意触发异常，使用send函数返回布尔值，而不是transfer函数
         3. **Strict Balance Equality**
            1. 建议：使用范围而不是“==”
         4. **Unmatched Type Assignment**
            1. 在循环中，变量的类型如果定义为uint8（0-255），而判断条件变量为uint256等较大类型数值，那么变量永远不会累加至大于该条件变量，出现无限循环
            2. 建议：如果不确定最大循环迭代次数，使用uint或uint256
         5. **Transaction State Dependency**
            1. 通过检查tx.origin来作为转账条件，可以被攻击
            2. 建议：使用msg.sender
         6. **Block Info Dependency**
            1. 通过区块的信息产生随机数，可能会被矿工控制
            2. 建议：用户获得随机数并在本地产生哈希，所有用户提交哈希之后要求所有用户提交原始随机数，检查随机数是否可以生成相同的哈希，如果用户的随机数都是正确的，则用这个随机数生成随机数
         7. **re-entrancy**
            1. 建议：使用send 或transfer转移以太
         8. **Nested Call**
            1. 循环太多次call会超出气体限制，导致错误
            2. 建议：开发人员应估计合同可以支持的最大循环迭代次数，并限制这些循环迭代次数。
         9. **Misleading Data Location**
            1. storage不会动态分配，如果函数内部创建了storage变量，将默认指向slot 0（这个位置有其他变量），会引发不可预测的错误
            2. 建议：澄清函数内部的参数的存储结构
      2. **Availability Defects 可用性 4种 （may not be utilized by attacker but are bad designs）**
         1. **Unmatched ERC-20 Standard**
            1. 编写不符合规范的合约，导致出错
            2. 建议：检查合约严格符合ERC20标准
         2. **Missing Reminder**
            1. ABI缺少提示，导致用户多次转账失败，浪费gas
            2. 建议：与外部交互的函数，添加必要的提示
         3. **Missing Return Statement**
            1. 函数声明了返回值，但函数实现并没有返回任何值，编译器会设置默认值，如果调用者不知道源代码，将函数的返回值作为一个判断条件，那么会出现未知错误。
            2. 建议：每个声明了返回值的函数，都在函数体内添加返回语句。
         4. **Greedy Contract**
            1. 合约的回退函数有payable，那么此合约可以接收转账，但是如果该合约没有实现转出的功能，那么将没有人可以得到合约中的Ethers
            2. 建议：如果合约可以接收以太币，那么一定要实现取款方式
      3. **Performance Defects 性能 3种（increase their gas cost）**
         1. **Unused Statement**
            1. 无用的函数参数和局部变量会影响代码的可读性（编译器将源代码编译为二进制代码的过程种会删掉这些无用参数）
            2. 删除无用语句
         2. **High Gas Consumption Function Type**
            1. Public函数消耗昂贵的内存，而external函数可以直接从便宜的calldata调用，如果函数只能被外部调用，那么将其定义为public，实属浪费。
            2. 建议：只能被外部调用的函数，声明为external函数
         3. **High Gas Consumption Data Type** 
            1. 字节数组bytes和byte[]相比，bytes消耗更少的gas
            2. 建议：使用bytes而不是byte[]
      4. **Maintainability Defects 可维护性 2种 （shorten the lifecycle of the contract）**
         1. **Hard Coded Address**
            1. 使用硬编码地址，如果地址出错，则合约永远无法正常运行，如果地址被销毁，那么发送给地址的Ether将永远丢失
            2. 建议：使用函数参数输入地址，而不是硬编码地址
         2. **Missing Interrupt**
            1. 如果合约没有自毁方式，那么当攻击发生时，就无法及时止损
            2. 建议：如果合约拥有大量的Ethers，添加自毁函数，使得合约可以在遭受攻击的时候，及时转移余额。
      5. **Reusability Defects 可重用性 2种 (increase the difficulty of code reuse)**
         1. **Deprecated APIs**
            1. Solidity在不断更新，一些API会被弃用或更新，如果开发者使用编译器提示将来会被弃用的API，那么之后就需要重构以实现代码重用，这造成了资源浪费。
            2. 建议：遵循最新的Solidity文档，并使用最新的API
         2. **Unspecified Compiler Version**
            1. 如果合约没有指定编译器版本，开发人员在将来的代码重用中可能会因为版本差异而遇到编译错误。
            2. 建议：修复编译器版本