Near Rust合约安全

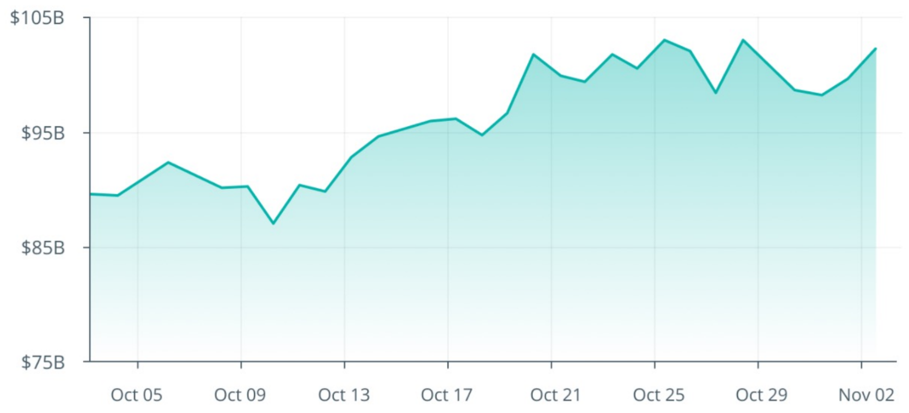
2021年11月24日

Muhui JIANG, PhD @ The HK PolyU
BlockSec Rust合约安全负责人

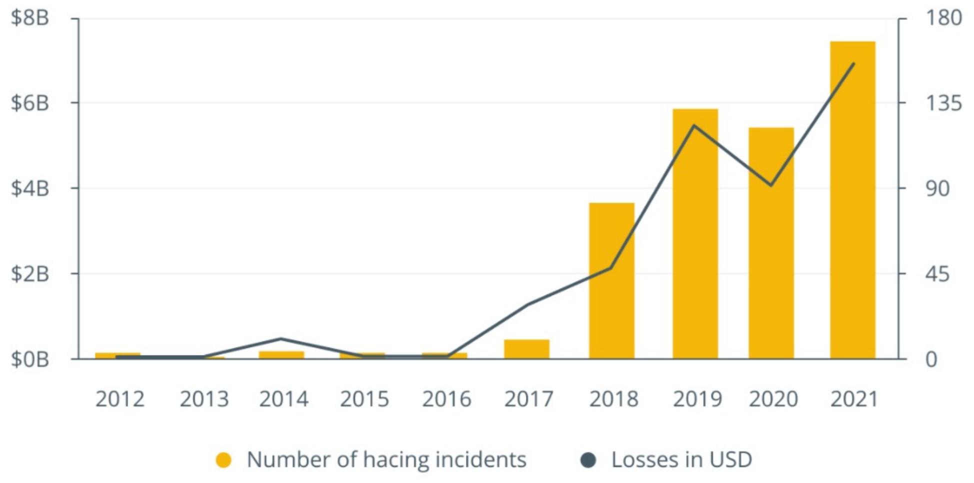
课程大纲

- 引言：合约安全的重要性
- 常见安全问题与攻击类型
  - 权限控制
  - 重入攻击
  - 价格操控
  - 整数溢出
  - 计算精度
  - 拒绝服务攻击
- 攻击防护与安全提升

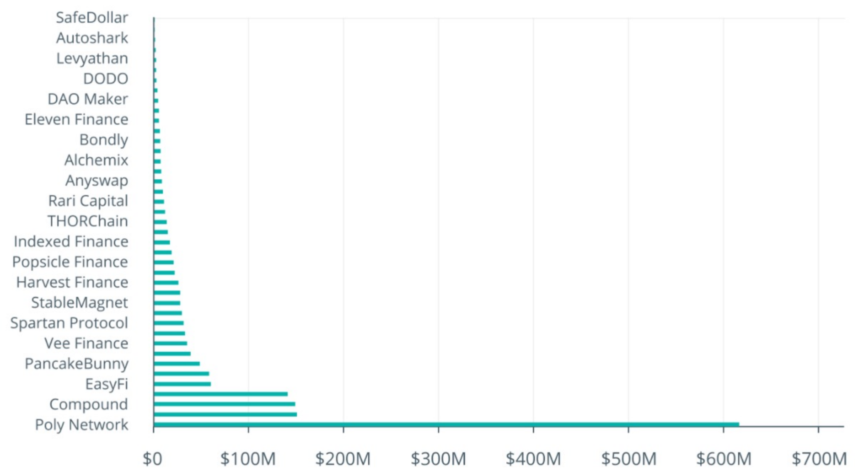
合约安全的重要性



$105B
$95B
$85B
$75B
Oct 05
Oct 09
Oct 13
Oct 17
Oct 21
Oct 25
Oct 29
Nov 02



$8B
$6B
$4B
$2B
$0B
180
135
90
45
0
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
● Number of hacing incidents
● Losses in USD



SafeDollar
Autoshark
Levyathan
DODO
DAO Maker
Eleven Finance
Bondly
Alchemix
Anyswap
Rari Capital
THORChain
Indexed Finance
Popsicle Finance
Harvest Finance
StableMagnet
Spartan Protocol
Vee Finance
PancakeBunny
EasyFi
Compound
Poly Network
$0
$100M
$200M
$300M
$400M
$500M
$600M
$700M

COINTELEGRAPH

BlockSec NEAR The Rust Programming Language

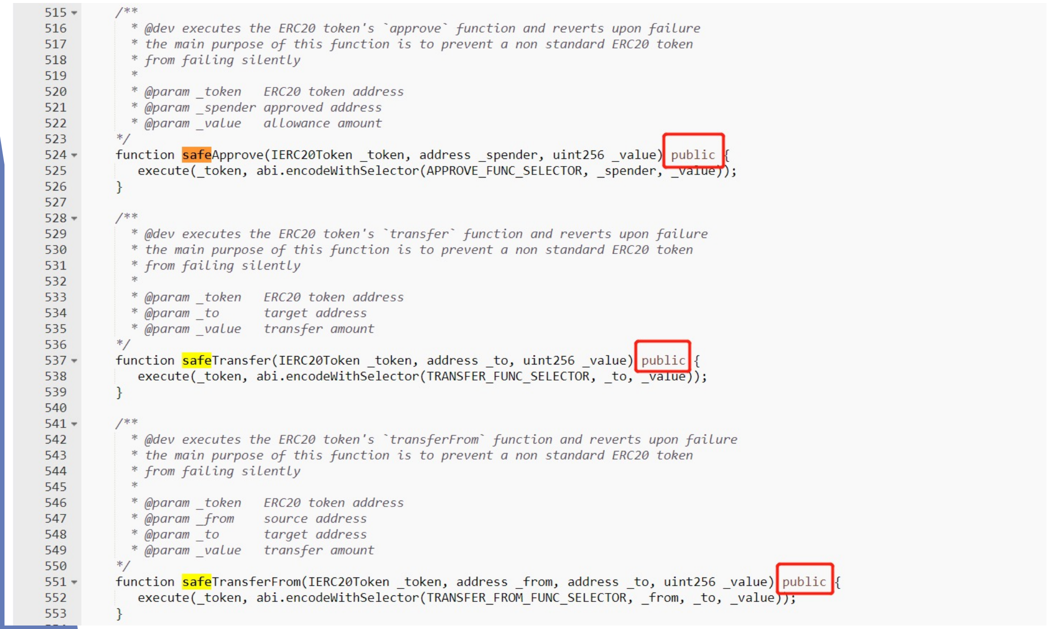
课程大纲

- 引言：合约安全的重要性
- 常见安全问题与攻击类型
    - 权限控制
    - 重入攻击
    - 价格操控
    - 整数溢出
    - 计算精度
    - 拒绝服务攻击
- 攻击防护与安全提升

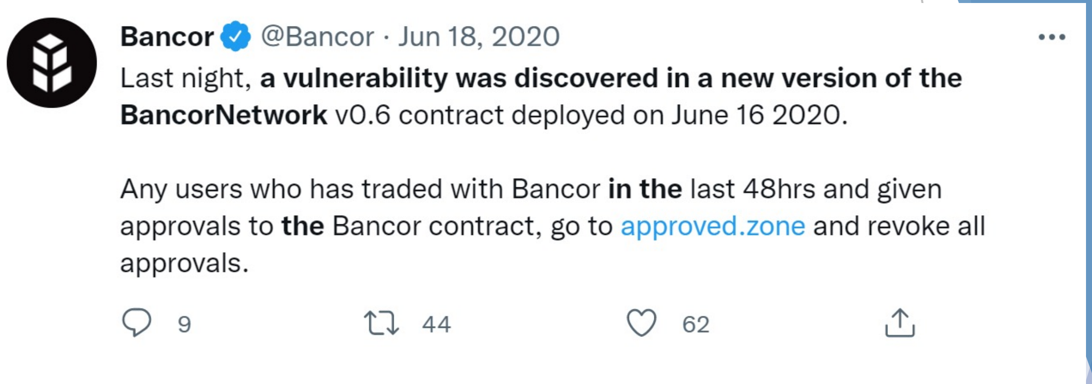
BlockSec NEAR The Rust Programming Language

权限控制

- 函数可见性
  - Public 函数： 可以被任何人调用
  - Private 函数： 只可以在内部调用



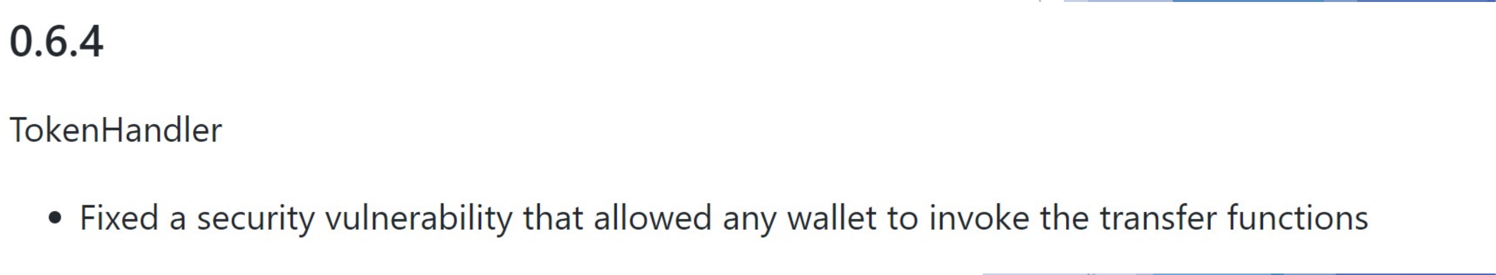
```
/**
  * @dev executes the ERC20 token's `approve` function and reverts upon failure
  * the main purpose of this function is to prevent a non standard ERC20 token
  * from failing silently
  *
  * @param _token   ERC20 token address
  * @param _spender approved address
  * @param _value   allowance amount
*/
function safeApprove(IERC20Token _token, address _spender, uint256 _value) public {
    execute(_token, abi.encodeWithSelector(APPROVE_FUNC_SELECTOR, _spender, _value));
}

/**
  * @dev executes the ERC20 token's `transfer` function and reverts upon failure
  * the main purpose of this function is to prevent a non standard ERC20 token
  * from failing silently
  *
  * @param _token   ERC20 token address
  * @param _to      target address
  * @param _value   transfer amount
*/
function safeTransfer(IERC20Token _token, address _to, uint256 _value) public {
    execute(_token, abi.encodeWithSelector(TRANSFER_FUNC_SELECTOR, _to, _value));
}

/**
  * @dev executes the ERC20 token's `transferFrom` function and reverts upon failure
  * the main purpose of this function is to prevent a non standard ERC20 token
  * from failing silently
  *
  * @param _token   ERC20 token address
  * @param _from    source address
  * @param _to      target address
  * @param _value   transfer amount
*/
function safeTransferFrom(IERC20Token _token, address _from, address _to, uint256 _value) public {
    execute(_token, abi.encodeWithSelector(TRANSFER_FROM_FUNC_SELECTOR, _from, _to, _value));
}
```



Bancor @Bancor · Jun 18, 2020

Last night, **a vulnerability was discovered in a new version of the BancorNetwork** v0.6 contract deployed on June 16 2020.

Any users who has traded with Bancor **in the** last 48hrs and given approvals to **the** Bancor contract, go to approved.zone and revoke all approvals.



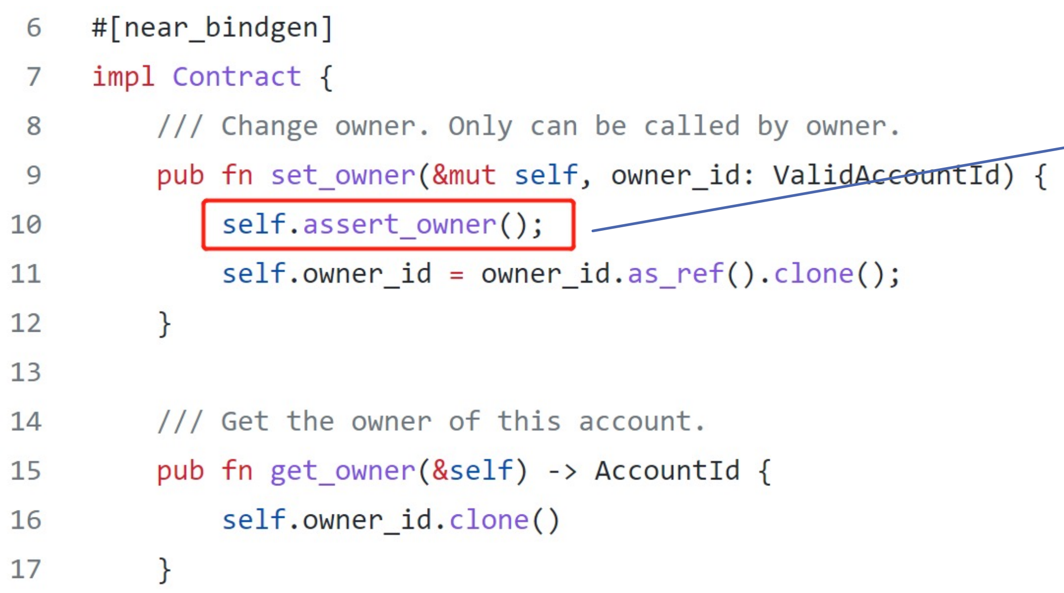
0.6.4

TokenHandler

- Fixed a security vulnerability that allowed any wallet to invoke the transfer functions

权限控制

- 白名单机制
  - 对于public函数，要有完整的验证机制。例如，只限定给owner，或者特定权限的人调用。因此，需要通过白名单校验



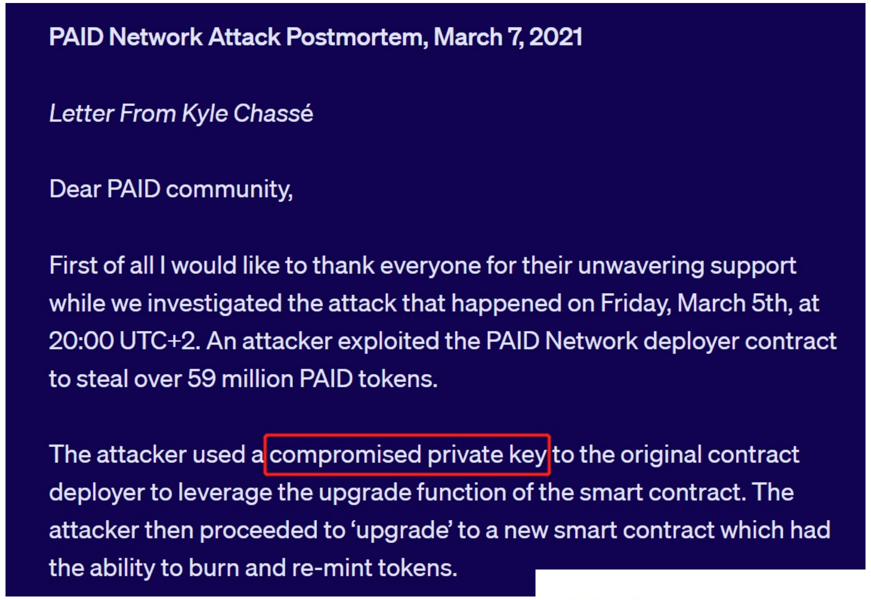
```
#[near_bindgen]
impl Contract {
    /// Change owner. Only can be called by owner.
    pub fn set_owner(&mut self, owner_id: ValidAccountId) {
        self.assert_owner();
        self.owner_id = owner_id.as_ref().clone();
    }

    /// Get the owner of this account.
    pub fn get_owner(&self) -> AccountId {
        self.owner_id.clone()
    }
```

只有owner有权限更改合约的owner

BlockSec NEAR The Rust Programming Language

权限控制

- 多签机制
  - 去中心化
  - 防止某一有权限人员的私钥丢失

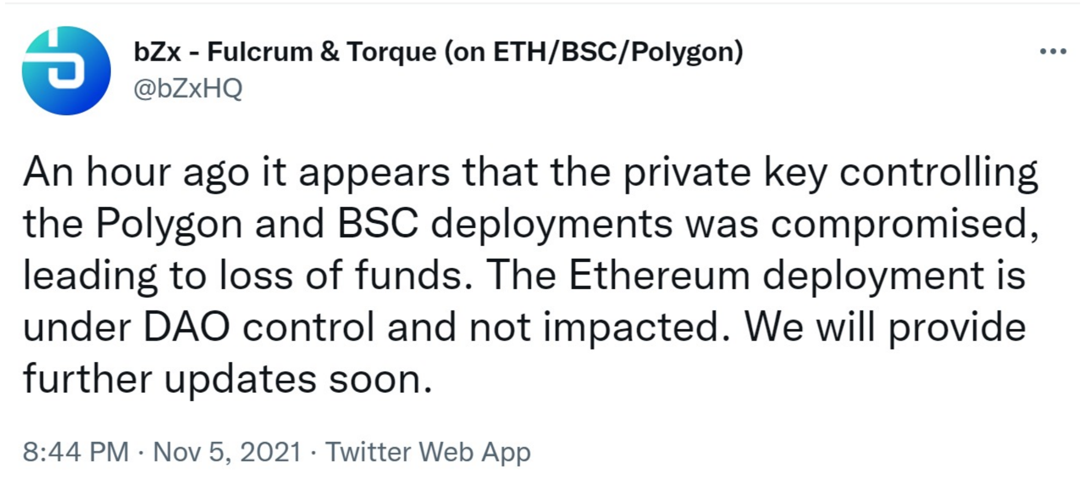

PAID Network Attack Postmortem, March 7, 2021

Letter From Kyle Chassé

Dear PAID community,

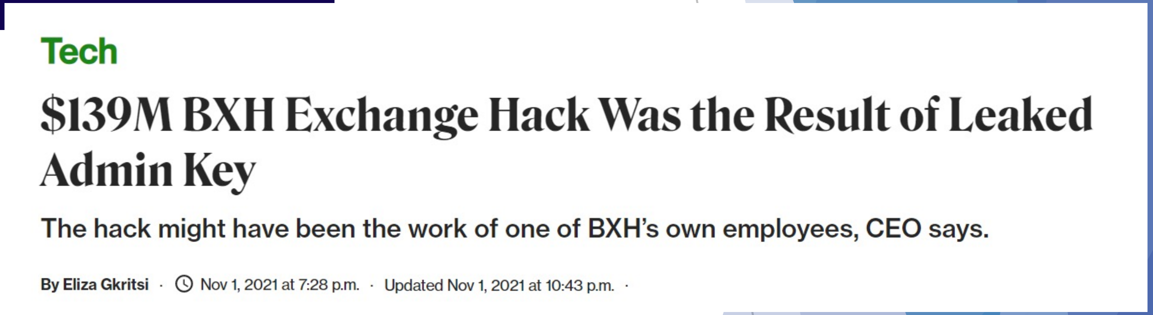
First of all I would like to thank everyone for their unwavering support while we investigated the attack that happened on Friday, March 5th, at 20:00 UTC+2. An attacker exploited the PAID Network deployer contract to steal over 59 million PAID tokens.

The attacker used a compromised private key to the original contract deployer to leverage the upgrade function of the smart contract. The attacker then proceeded to 'upgrade' to a new smart contract which had the ability to burn and re-mint tokens.


bZx - Fulcrum & Torque (on ETH/BSC/Polygon)
@bZxHQ

An hour ago it appears that the private key controlling the Polygon and BSC deployments was compromised, leading to loss of funds. The Ethereum deployment is under DAO control and not impacted. We will provide further updates soon.

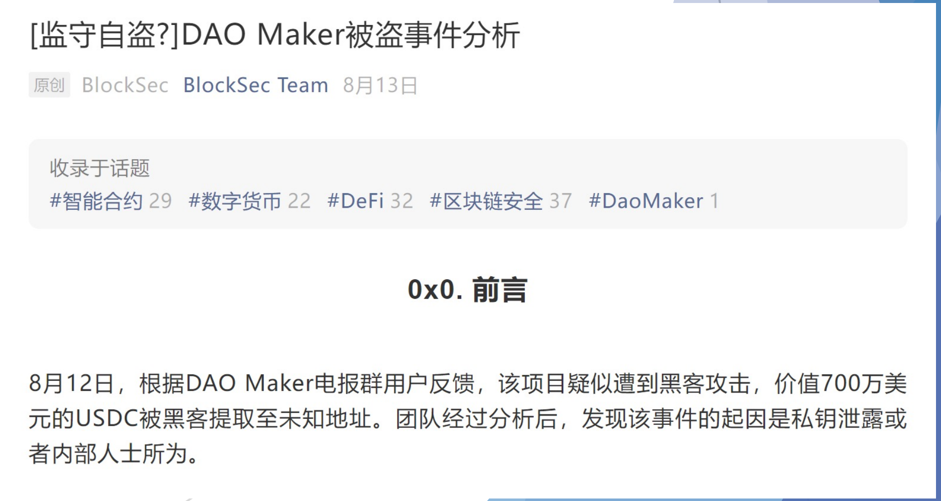
8:44 PM · Nov 5, 2021 · Twitter Web App


Tech

$139M BXH Exchange Hack Was the Result of Leaked Admin Key

The hack might have been the work of one of BXH's own employees, CEO says.

By Eliza Gkritsi · Nov 1, 2021 at 7:28 p.m. · Updated Nov 1, 2021 at 10:43 p.m. ·


[监守自盗?]DAO Maker被盗事件分析

原创 BlockSec BlockSec Team 8月13日

收录于话题
#智能合约 29 #数字货币 22 #DeFi 32 #区块链安全 37 #DaoMaker 1

0x0. 前言

8月12日，根据DAO Maker电报群用户反馈，该项目疑似遭到黑客攻击，价值700万美元的USDC被黑客提取至未知地址。团队经过分析后，发现该事件的起因是私钥泄露或者内部人士所为。

课程大纲

- 引言：合约安全的重要性
- 常见安全问题与攻击类型
  - 权限控制
  - 重入攻击
  - 价格操控
  - 整数溢出
  - 计算精度
  - 拒绝服务攻击
- 攻击防护与安全提升

BlockSec NEAR The Rust Programming Language

重入攻击

攻击者

职员A

职员B

取款操作

- 记账 account.balance = account.balance - amount
- 转钱 transfer (account, amount, _)

谁先谁后？

重入攻击

攻击者

职员A

职员B

初始状态：攻击者账户：100元

步骤一： 向职员A请求取款60元

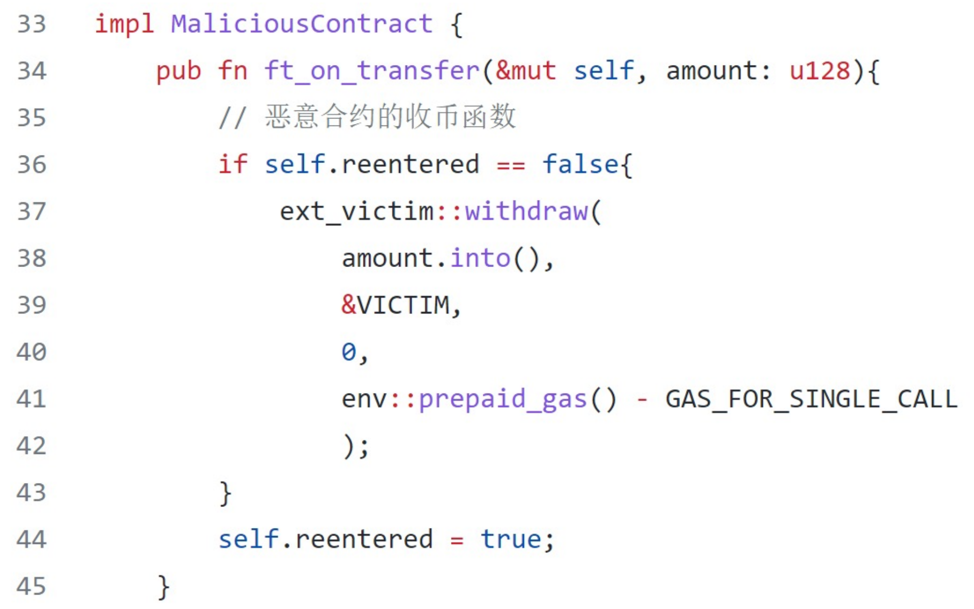
- 步骤二：职员A拿出60元给攻击者
- 步骤三：向职员B请求取款60元，此时职员B并不知道攻击者已经从职员A处取款60元
- 步骤四：职员B拿出60元给攻击者
- 步骤五：职员B更改攻击者银行账户余额。 攻击者账户：100 - 60 = 40

- 步骤六：职员A更改攻击者银行账户余额。攻击者账户：40 – 60 = -20

攻击者攻击成功：用100元的余额，套利120元

NEAR上的重入攻击Demo

- 合约A：Attacker



```rust
impl MaliciousContract {
    pub fn ft_on_transfer(&mut self, amount: u128){
        // 恶意合约的收币函数
        if self.reentered == false{
            ext_victim::withdraw(
                amount.into(),
                &VICTIM,
                0,
                env::prepaid_gas() - GAS_FOR_SINGLE_CALL
                );
        }
        self.reentered = true;
    }
```

NEAR上的重入攻击Demo

- 合约B：Victim（可以是一个DEX项目）
  - Attacker账户再Victim中拥有100Token，Victim总计持有200个Token

```rust
impl Default for VictimContract {
    fn default() -> Self {
        Self {
            attacker_balance: 100,
            other_balance:100
        }
    }
}
```

```rust
#[near_bindgen]
impl VictimContract {
    pub fn withdraw(&mut self,amount: u128) -> Promise{
        assert!(self.attacker_balance>= amount);
        // Call Attacker的收币函数
        ext_ft_token::ft_transfer_call(
            amount.into(),
            &FT_TOKEN,
            0,
            env::prepaid_gas() - GAS_FOR_SINGLE_CALL * 2
            )
            .then(ext_self::ft_resolve_transfer(
                amount.into(),
                &env::current_account_id(),
                0,
                GAS_FOR_SINGLE_CALL,
            ))
    }

```

NEAR上的重入攻击Demo

- 合约C：Token合约（NEP141）

```rust
#[near_bindgen]
impl FungibleToken {
    pub fn ft_transfer_call(&mut self,amount: u128)-> PromiseOrValue<U128>{
        // 相当于 internal_ft_transfer
        self.attacker_balance += amount;
        self.victim_balance   -= amount;

        // Call Attacker的收币函数
        ext_fungible_token_receiver::ft_on_transfer(
            amount.into(),
            &ATTACKER,
            0,
            env::prepaid_gas() - GAS_FOR_SINGLE_CALL
            ).into()
    }
```

NEAR上的重入攻击Demo

- 攻击流程：
  - Attacker合约通过malicious_call函数，调用Victim合约（合约B）中的withdraw函数，希望提现60个Token
  - 合约B判断Attacker账户是否有足够的余额，此时余额100>60，账户有足够余额
  - 合约B执行withdraw函数，withdraw函数将调用合约C中的ft_transfer_call函数
  - 合约C中的ft_transfer_call函数，将更新attacker账户的余额，以及Victim合约账户余额，并且调用合约A的ft_on_transfer，通知合约A收币
  - 合约A被Attacker所控制，再次执行合约B的withdraw函数，达到**重入**效果
  - Victim合约中的attacker的balance还未更新，暂时余额还是100，断言依然通过
  - Victim合约再次调用ft_transfer_call对Attacker转账，
  - 函数逐级返回，攻击者获利

测试链上部署

- 相关代码
  - https://github.com/blocksecteam/near_demo/tree/main/ReentrancyDemo
- 攻击交易
  - https://explorer.testnet.near.org/accounts/blocksec.testnet

重入的防护技术

- 先记账，再转钱
- 引入互斥锁
- 设置gas limit

课程大纲

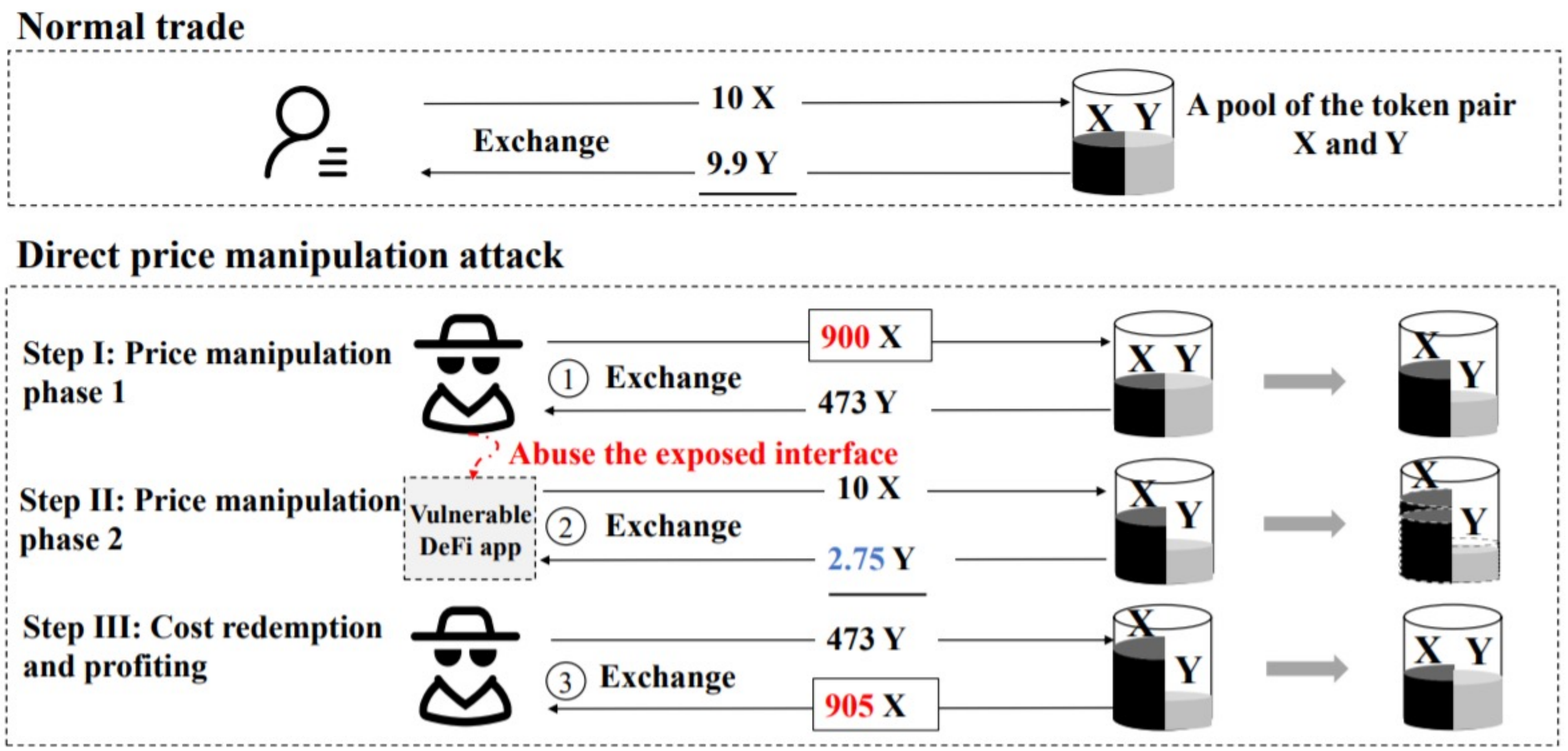
- 引言：合约安全的重要性
- 常见安全问题与攻击类型
  - 权限控制
  - 重入攻击
  - 价格操控
  - 整数溢出
  - 计算精度
  - 拒绝服务攻击
- 攻击防护与安全提升

价格操控

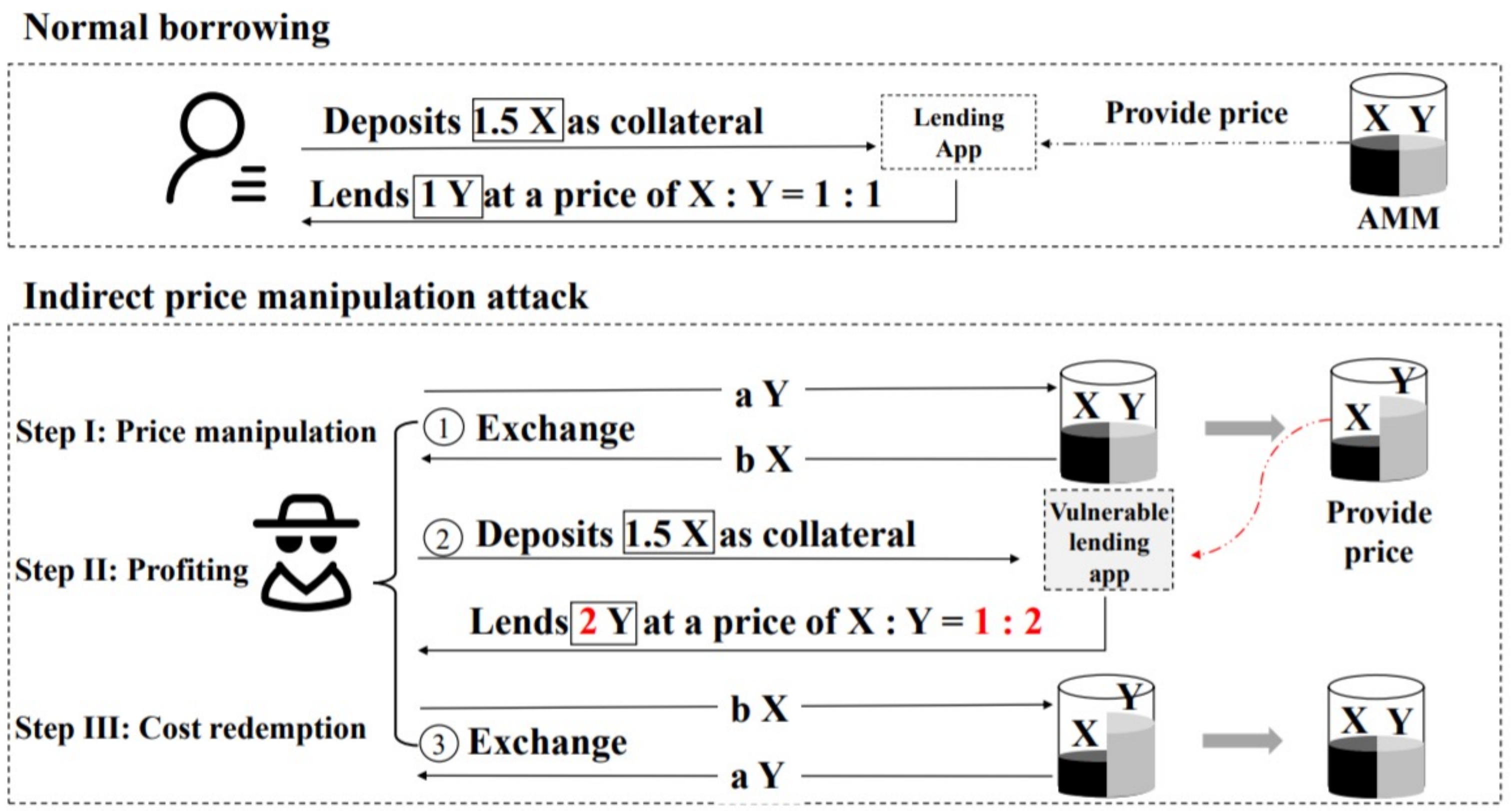
- Uniswap的价格计算公式

$$F(x) = \frac{997x * R_Y}{997x + 1000R_X}$$

价格操控 之 直接操控


Normal trade
10 X
Exchange
9.9 Y
X Y
A pool of the token pair X and Y
Direct price manipulation attack
Step I: Price manipulation phase 1
900 X
① Exchange
473 Y
X Y
X Y
Abuse the exposed interface
Step II: Price manipulation phase 2
Vulnerable DeFi app
10 X
② Exchange
2.75 Y
X Y
X Y
Step III: Cost redemption and profiting
473 Y
③ Exchange
905 X
X Y
X Y

价格操控 之 间接操控


Normal borrowing
Deposits 1.5 X as collateral
Lending App
Provide price
X Y
Lends 1 Y at a price of X : Y = 1 : 1
AMM
Indirect price manipulation attack
Step I: Price manipulation
① Exchange
a Y
b X
X Y
Y
X
② Deposits 1.5 X as collateral
Step II: Profiting
Vulnerable lending app
Provide price
Lends 2 Y at a price of X : Y = 1 : 2
Step III: Cost redemption
③ Exchange
b X
a Y
Y
X
X Y

价格操控 之 防护

- 面对直接攻击
  - 滑点校验
- 面对间接攻击
  - TWAP oracle
  - VWAP oracle

BlockSec NEAR The Rust Programming Language

课程大纲

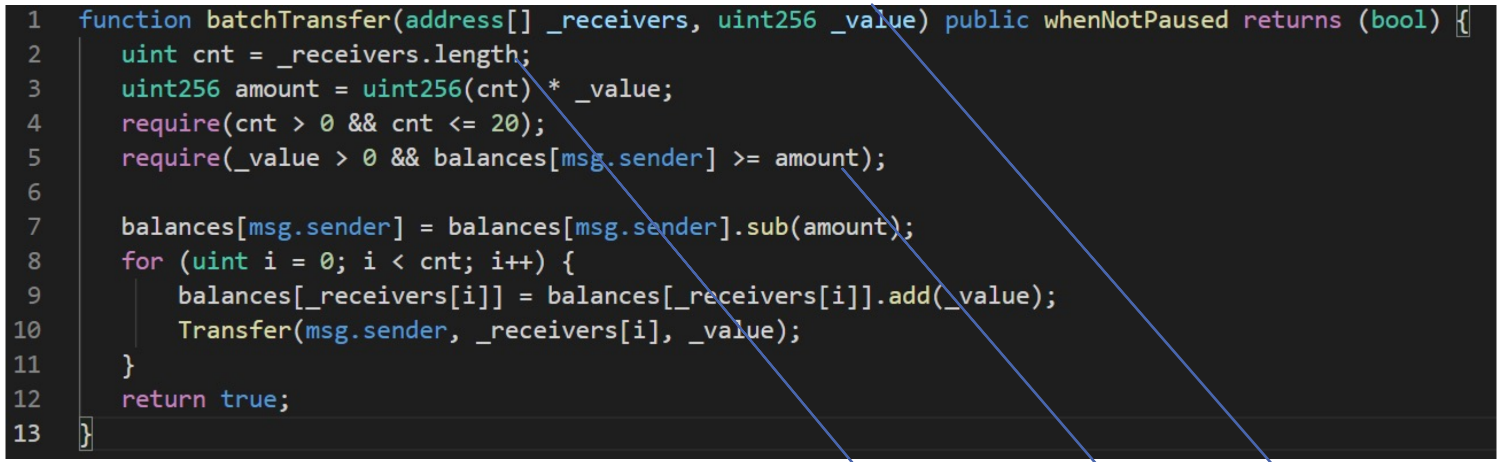
- 引言：合约安全的重要性
- 常见安全问题与攻击类型
  - 权限控制
  - 重入攻击
  - 价格操控
  - 整数溢出
  - 计算精度
  - 拒绝服务攻击
- 攻击防护与安全提升

整数溢出

- 整数上溢
  - 0xffffffff + 0x00000001 = 0x00000000
- 整数下溢
  - 0x00000000 – 0x00000001 = 0xffffffff

整数溢出实例

- BeautyChain 团队于 2018年4月22日 遭受攻击，损失10^58 个BEC。
  - https://etherscan.io/tx/0xad89ff16fd1ebe3a0a7cf4ed282302c06626c1af33221ebe0d3a470aba4a660f



```
function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
    uint cnt = _receivers.length;
    uint256 amount = uint256(cnt) * _value;
    require(cnt > 0 && cnt <= 20);
    require(_value > 0 && balances[msg.sender] >= amount);

    balances[msg.sender] = balances[msg.sender].sub(amount);
    for (uint i = 0; i < cnt; i++) {
        balances[_receivers[i]] = balances[_receivers[i]].add(_value);
        Transfer(msg.sender, _receivers[i], _value);
    }
    return true;
}
```

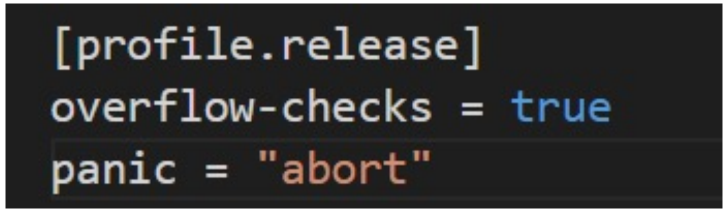
2^255

2

Amount = 0, 检查通过

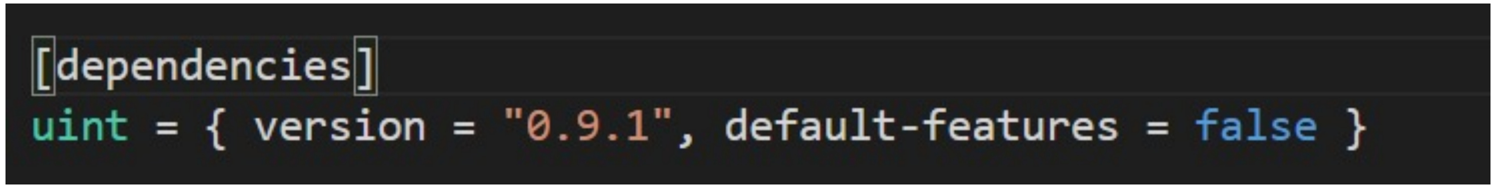
NEAR中的整数溢出防护

- 配置Cargo.toml,再release模式下检查整数溢出



```
[profile.release]
overflow-checks = true
panic = "abort"
```

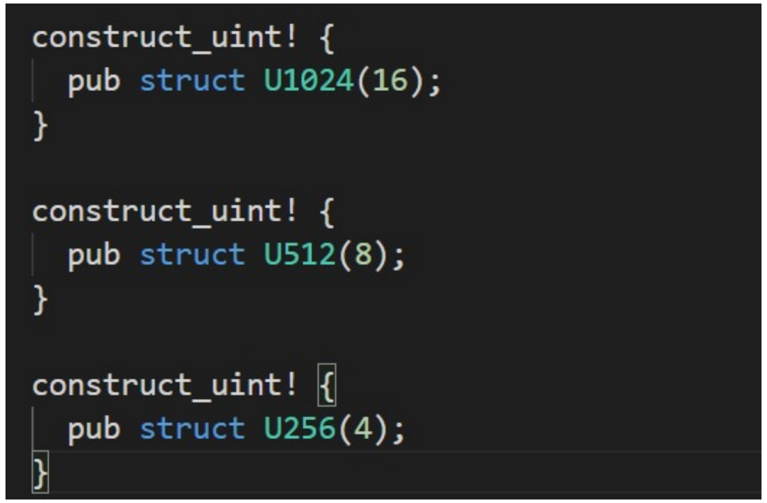
- 使用Rust Crate uint支持更大整数
  - 目前Rust标准库所能提供最大整数类型仅为u128
  - Rust Crate uint可提供大无符号整数类型 可在Cargo.toml中添加依赖



```
[dependencies]
uint = { version = "0.9.1", default-features = false }
```

NEAR中的整数溢出防护

- 使用Rust Crate uint支持更大整数
  - 在Rust程序中导入该crate
    - use uint::construct_uint;
  - 构造自己想要的无符号整数类型



```
construct_uint! {
  pub struct U1024(16);
}

construct_uint! {
  pub struct U512(8);
}

construct_uint! {
  pub struct U256(4);
}
```

BlockSec NEAR The Rust Programming Language

NEAR中的整数溢出防护

- 使用uint类型转化函数检测整数上溢

```
#[test]
fn test_overflow(){
    // u128所能表示的最大值, 即 2^128 -1
    let amounts: u128 = 340282366920938463463374607431768211455;

    // U256能够正常表示(2^128 -1)*(2^128 -1)的运算结果, 并不会发生溢出。
    let amount_u256 = U256::from(amounts) * U256::from(amounts);
    println!("{:?}",amount_u256);

    // 此处(2^128 -1) + 1 = 2^128
    let amount_u256 = U256::from(amounts) + 1;
    println!("{:?}",amount_u256);

    // 将溢出u128无符号整数所能表示的范围0至2^128 -1, 因此会触发Panic.
    let amount_u128 = amount_u256.as_u128();
    println!("{:?}",amount_u128);
}
```

NEAR中的整数溢出防护

- 使用Safe Math检查整数上溢或者下溢
  - checked_sub检查下溢

```
#[test]
fn test_underflow(){
    let amounts= U256::from(0);
    let amount_u256 = amounts.checked_sub(U256::from(1));
    println!("{:?}",amount_u256);
}
```

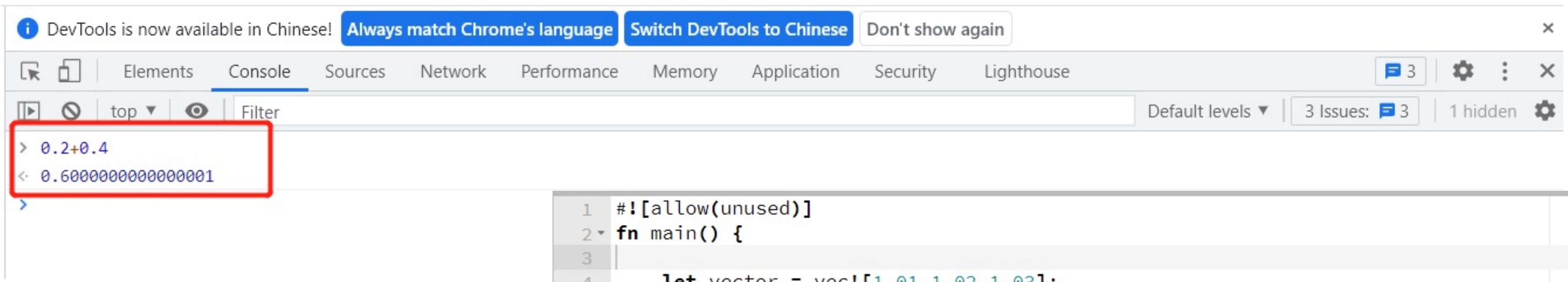
```
 #[test]
 fn test_underflow(){
     let amounts= U256::from(0);
-    let amount_u256 = amounts.checked_sub(U256::from(1));
+    let amount_u256 = amounts.checked_sub(U256::from(1)).expect("ERR_SUB_INS
     println!("{:?}",amount_u256);
 }
```

课程大纲

- 引言：合约安全的重要性
- 常见安全问题与攻击类型
  - 权限控制
  - 重入攻击
  - 价格操控
  - 整数溢出
  - 计算精度
  - 拒绝服务攻击
- 攻击防护与安全提升

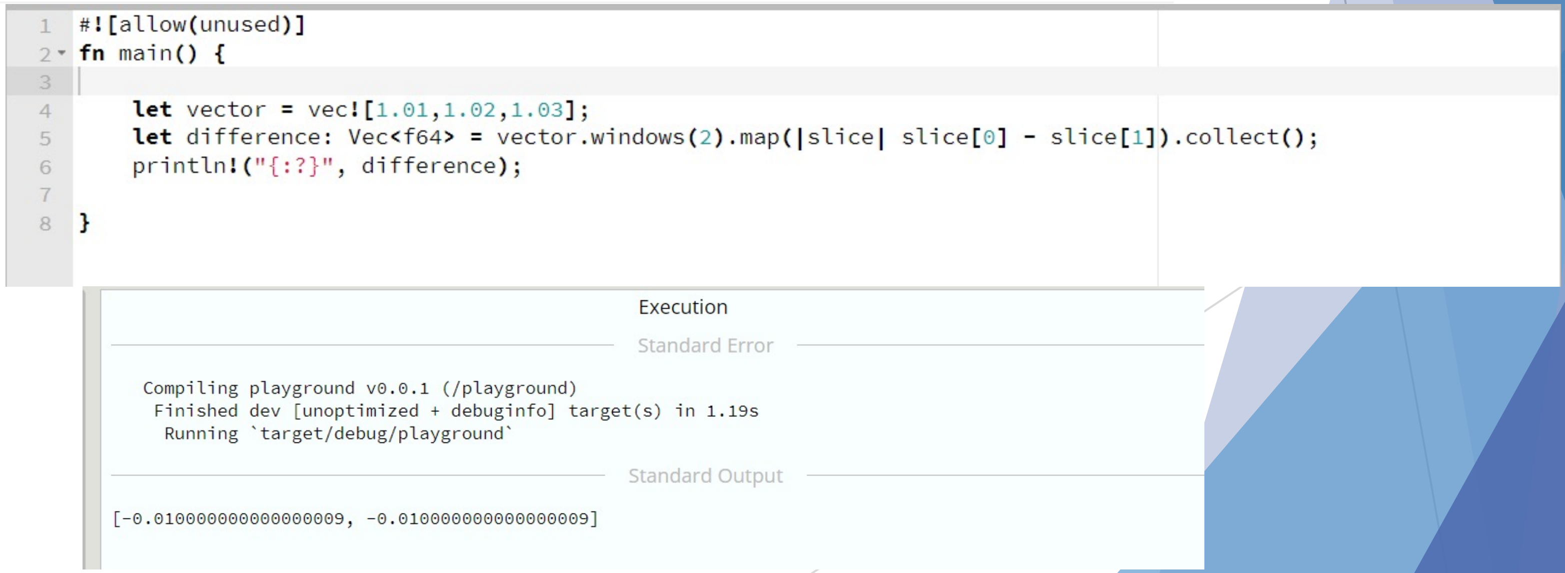
计算精度

- 在你的chrome或者firefox里按下F12
- 进入控制台



```
> 0.2+0.4
< 0.6000000000000001
```

- 在Rust中
  - 浮点计算并不精确



```rust
#![allow(unused)]
fn main() {

    let vector = vec![1.01,1.02,1.03];
    let difference: Vec<f64> = vector.windows(2).map(|slice| slice[0] - slice[1]).collect();
    println!("{:?}", difference);

}
```

Execution

Standard Error

```
   Compiling playground v0.0.1 (/playground)
    Finished dev [unoptimized + debuginfo] target(s) in 1.19s
     Running `target/debug/playground`
```

Standard Output

```
[-0.010000000000000009, -0.010000000000000009]
```

计算精度

- 用整数表示浮点数
- 乘法在除法之前
  - (10/4)*2 = 4
  - (10*2)/4 = 5
- 分子较大
  - 2/3 + 2/3 = 0
  - 2*1000/3 +2*1000/3 = 666 + 666 = 1332
    - 1332/1000 = 1
- 为了防止上溢，可以将比如u128转换为u256，经过计算后，再转回u128

课程大纲

- 引言：合约安全的重要性
- 常见安全问题与攻击类型
  - 权限控制
  - 重入攻击
  - 价格操控
  - 整数溢出
  - 计算精度
  - 拒绝服务攻击
- 攻击防护与安全提升

拒绝服务攻击

- 合约逻辑存在缺陷，未考虑计算复杂度，导致使用gas超出上限
- 合约依赖外部合约的执行结果，导致阻塞
- Owner丢失私钥，无法对系统状态做及时更新

计算复杂度导致gas超出限制

- 合约逻辑存在缺陷，未考虑计算复杂度，导致使用gas超出上限。 NEAR主网配置 max_total_prepaid_gas = 300TGas
  - https://github.com/near/nearcore/blob/master/nearcore/res/mainnet_genesis.json#L192
- 循环遍历一个可被外部调用更改的数据结构
- 一个给用户分红的智能合约
  - https://github.com/blocksecteam/near_demo/blob/main/DoSDemo/unlimit_iter_contract/src/lib.rs

合约依赖外部合约的执行结果，导致阻塞

- 一个竞价的智能合约
  - https://github.com/blocksecteam/near_demo/blob/main/DoSDemo/invalid_refund/src/lib.rs

课程大纲

- 引言：合约安全的重要性
- 常见安全问题与攻击类型
  - 权限控制
  - 重入攻击
  - 价格操控
  - 整数溢出
  - 计算精度
  - 拒绝服务攻击
- 攻击防护与安全提升

合约攻击防护与安全提升

- 开发者应当提高安全意识，尽可能书写规范，安全的代码
  - 在合约中使用safemath，避免整数溢出
  - 转账操作先记账，后转钱，避免重入
  - 特定函数使用多签
  - More ...
- 尽量不要从0开始写代码，学会复用已有的，安全的代码项目
  - Solidity：OpenZeppelin
  - Near：需要我们共同的努力
- 智能合约审计服务
  - BlockSec https://www.blocksecteam.com/
  - Twitter：https://twitter.com/BlockSecTeam
  - Email：contact@blocksecteam.com

Q & A