OWASP 中国
The Open Web Application Security Project

区块链安全 Top 10 2019

V1.0

OWASP 中国

2019 年 10 月

关于 OWASP 中国

“开源 Web 应用安全项目”（Open Web Application Security Project，简称“OWASP”）是一个开放的社区，致力于帮助各企业组织开发、购买和维护可信任的应用程序。OWASP 中国是 OWASP 安全组织在中国的分部，是全球近 250 个区域分部之一。

在 OWASP 中国，您可以找到以下免费和开源的信息：

- 应用安全工具和参考标准；
- 关于应用安全测试、安全代码开发和安全代码审查方面的完整书籍；
- 演示文稿和视频；
- 关于常见风险的 Cheat Sheets；
- 标准的安全控制和安全库；
- 全国 20 个区域性分会；
- 尖端技术研究；
- 专业的全球会议；
- 邮件列表。

更多中文信息，请访问：http://www.owasp.org.cn；

更多全球信息，请访问：https://www.owasp.org。

所有的 OWASP 工具、文档、论坛和全球各地分会都是开放的，并对所有致力于改进应用程序安全的人士开放。

我们主张将应用程序安全问题看作是人、过程和技术的问题，因为提高应用程序安全最有效的方法是在这些方面提升。

OWASP 全球和 OWASP 中国没有商业压力，使得我们能够提供无偏见、实用、低成本的应用安全信息。尽管 OWASP 全球和 OWASP 中国支持合理使用商业安全技术，但 OWASP 不隶属于任何技术公司。和许多开源软件项目一样，OWASP 以一种协作、开放的方式制作了许多不同种类的材料。几乎每一个与 OWASP 中国相关的人都是一名志愿者，这包括了 OWASP 中国的主席、副主席、各区域分部负责人、项目负责人和项目成员。

我们期待您的加入！

前言

近几年，区块链技术的发展非常迅猛，安全形势也越来越严峻，仅安全事件导致的直接经济损失就高达 35 亿美元，很多公司甚至因此倒闭，给行业带来了巨额的经济损失和惨痛的教训。基于此，OWASP 中国成立专门研究小组，收集、整理和分析了 2011 年至 2019 年间共 160 个典型区块链安全事件，并在本文档中给出了排列和描述，希望能帮助到广大的区块链从业者和关注区块链安全的人们。

在参考了类似 CVSS（Common Vulnerability Scoring System）等安全威胁评估方法之后，本文以每类威胁历史安全事件所导致的直接经济损失总额为依据，通过客观数据评估威胁大小。直接经济损失总额包含了威胁评估的两个重要因素，一是威胁发生的数量（即威胁发生的次数）；二是威胁发生导致的影响（即直接经济损失）。使用直接经济损失表示威胁的大小的好处是， 数据相对客观，避免了主观数据导致评估结果误差较大问题，同时，这种评估方式更具有良好的解释性。

阅读本文档时需注意以下三点：（1）安全事件导致的经济损失以案件发生时的虚拟币价格计算；（2）统计分析过程中只计算了直接经济损失，未计算间接经济损失；（3）24.3%的安全事件（39 个）未公布经济损失，因而未计入损失统计。

致谢

感谢以下为本文档编写做成贡献的个人（排名不分先后，按姓氏拼音排列）

项目组长：付山阳

项目组成员：Kevin Gu、Victor Fang、Wei Quan、

侯欣杰、蒋旭宪、宋飞、王颉、张迅迪

目录

关于 OWASP 中国 1

前言 3

致谢 4

一、 高级可持续威胁 10

1. 风险描述 10

2. 危害描述 10

3. 实际案例 10

案例 1-1： 10

案例 1-2： 11

案例 1-3： 11

4. 修复方案 12

二、 失控的币值通胀 13

1. 风险描述 13

2. 危害描述 13

3. 实际案例 13

案例 2-1： 13

4. 修复方案 14

三、 失效的权限控制 16

1. 风险描述 16

2. 危害描述 16

3. 实际案例 16

案例 3-1： 16

案例 3-2： 17

案例 3-3： 17

4. 修复方案 18

四、 不安全的共识协议 19

1. 风险描述 19

2. 危害描述 19

3. 实际案例 19

案例 4-1： 19

案例 4-2： 20

4. 修复方案 21

五、 考虑不充分的程序逻辑 22

1. 风险描述 22

2. 危害描述 22

3. 实际案例 22

案例 5-1： 22

4. 修复方案 25

六、 不严谨的业务策略 26

1. 风险描述 26

2. 危害描述 26

3. 实际案例 26

案例 6-1： 26

4. 修复方案 27

七、 校验不严格的交易逻辑 28

1. 风险描述 28

2. 危害描述 28

3. 实际案例 28

案例 7-1： 28

案例 7-2： 29

案例 7-3： 30

案例 7-4： 31

4. 修复方案 32

八、 脆弱的随机数机制 33

1. 风险描述 33

2. 危害描述 33

3. 实际案例 33

案例 8-1： 33

4. 修复方案 34

九、 存在缺陷的激励机制 35

1. 风险描述 35

2. 危害描述 35

3. 实际案例 35

案例 9-1： 35

4. 修复方案 36

十、 日志记录和监控不足 37

1. 风险描述 37

2. 危害描述 37

3. 实际案例 37

4. 修复方案 37

后序 39

参考文献 40

一、 高级可持续威胁

1. 风险描述

高级可持续威胁类问题往往出现在区块链领域的中心化团队上，比如：交易所、钱包、矿池等。由于这些区块链团队在基础设施安全建设和安全运营上的欠缺，导致基础设施存在安全漏洞，从而被攻击者利用，实施数字货币的盗窃或者破坏。

2. 危害描述

高级可持续威胁会导致区块链系统被入侵，权限被控制，最终导致资金被盗取，敏感数据被泄露等系列严重问题。在 2008-2018 年间，就共有 37 起高级可持续威胁类安全事件，导致的直接经济损失高达 16 亿美元，是区块链领域的第一大威胁。

3. 实际案例

案例 1-1：

Mt.Gox 是区块链领域最著名的入侵事件之一。早在 2011 年 6 月，Mt. Gox 交易所就遭到黑客攻击。据称，该公司的一名审计人员的电脑受到了攻击，黑客利用这台电脑访问比特币交易所的内网，人为的将比特币的名义价值修改为 1 美分，然后从该交易所的客户账户转移了大约 2000 枚比特币。2014 年，Mt.Gox 遭受了更严重的黑客攻击，近 85 万的比特币被盗窃，攻击手法并没有被披露，但据 Mt.Gox 内部人员透露，其存在内部组织不一致、程序安全性不佳、网站源代码泄露等严重的问题。甚至有黑客怀疑他们是监守自盗，事后又渗透进去，可见安全性很差。这次的事件导致它们近 4.6 亿美元的损失，并最终走向破产，CEO 也受到法律的制裁。

案例 1-2：

Coincheck 是另一起著名的区块链领域入侵事件。Coincheck 将钱存入热钱包，而且不实行多重签名系统，最终导致了黑客有机可乘，该平台全部 5.26 亿 NEM 币（新经币）被非法转移。根据估算，这批丢失的新经币价值高达 5.23 亿美金，这是目前历史上最大规模的数字货币盗窃案。

案例 1-3：

Fomo3D 以及类似的游戏（Lastwiner, FomoGame 等）遭受到的攻击是另一类基于以太坊合约的 APT，可以称之为 BAPT (区块链高级持续威胁)。 BAPT 的概念于 2018 年提出，区别于传统网络安全的 APT 在于，其攻击目标是区块链的基础设施，智能合约等。值得一提的是，很多加密货币交易所遭受的攻击，属于传统 APT 范畴。BAPT 攻击是需要长时间的，使用特定的合约不停的跟受害合约进行交易（内部交易），对受害合约进行扫描，发现其漏洞。然后改进针对其漏洞的攻击合约。

下图中显示的多笔失败交易就是攻击者的攻击合约漏洞产生的：

等到受害合约开始拥有一定数量的用户，同时奖励和资金池开始增长以后，便开始大规模的部署和运行攻击合约，利用用受害合约的漏洞获得高概率的回报。

更多信息请参考此链接：https://www.chaindd.com/3107866.html

4. 修复方案

不管是交易所、钱包，还是矿池等团队，都需要高度重视产品安全、办公安全和内部风险管理等安全方面的建设。产品安全可以参考 OWASP S-SDLC 项目提出的最佳安全实践；而办公安全侧重入侵检测和数据防泄漏，市场上 Top 5 的安全供应商都可以给出较好的方案；另外，内部的风险管理也都要尽快的建立和完善。

另外游戏开发者需要对玩家进行评估，区别机器玩家，合约玩家等非正常类型的玩家，这些非正常类型的玩家有很大的概率是用来对一个合约进行漏洞扫描和分析的，游戏开发者应该持续分析数据，评估可能存在的风险，及时作出防护，以降低安全风险。

二、 失控的币值通胀

1. 风险描述

比特币及其他区块链虚拟货币吸引大家的一个主要原因是其限定了铸币总数，从而避免了法币通胀和持币者长期承受经济损失。但若铸币总数限制部分的程序设计和实现有漏洞，则可能导致铸币总数失控，甚至超出设定发币总数多倍，致使币值失控通胀。

2. 危害描述

自区块链技术诞生以来，币值通胀类威胁导致了 10 亿多美元的直接经济损失，占整个区块链安全事件经济损失的 29.55%，而其中智能合约层案件最为突出，单 BEC 的币值通胀事件就损失了近 10 亿美元，直接导致了其价值归零。

3. 实际案例

案例 2-1：

币值通胀类最典型的案例就是 BEC 安全事件。BEC 主要由于币值控制部分的代码出现了整数溢出漏洞，导致出现无限铸币的情况。出现问题的代码如下图所示，红色框中的_fee 与_value 参数分别代表转账费用和转账金额，这两个参数都可以由攻击者控制。作者的本意是希望通过这行代码检查账户的余额是否足够支付转账金额和转账费用，如果不够支付则退出函数不进行支付操作。但当_value 的值足够大（比如_value=2^256-1），以至于_value+_fee 会产生上溢，_value+_fee 的值等于 0，这样绕过了代码的检查，仍然会实施支付操作。而实际转账的 Transfer 函数不再检查余

额是否足够，以至于 value=2^256-1 时，Transfer(_from,_to,_value)也可以转账成功，攻击者凭空获得大量的数字货币。

```
203     function transferProxy(address _from, address _to, uint256 _value, uint256 _fee,
204         uint8 _v,bytes32 _r, bytes32 _s) public transferAllowed(_from) returns (bool){
205
206         if(balances[_from] < _fee + _value) revert();
207
208         uint256 nonce = nonces[_from];
209         bytes32 h = keccak256(_from,_to,_value,_fee,nonce);
210         if(_from != ecrecover(h,_v,_r,_s)) revert();
211
212         if(balances[_to] + _value < balances[_to]
213             || balances[msg.sender] + _fee < balances[msg.sender]) revert();
214         balances[_to] += _value;
215         Transfer(_from, _to, _value);
216
217         balances[msg.sender] += _fee;
218         Transfer(_from, msg.sender, _fee);
219
220         balances[_from] -= _value + _fee;
221         nonces[_from] = nonce + 1;
222         return true;
223     }
```

4. 修复方案

对整数溢出漏洞导致的币值通胀问题，建议使用安全的算术函数做加减乘除处理。对币值控制的代码应该谨慎安全的处理。

https://ethereumdev.io/safemath-protect-overflows/

```
/**
  * @title SafeMath
  * @dev Math operations with safety checks that throw on error
  */
library SafeMath {
  function mul(uint256 a, uint256 b) internal constant returns (uint256) {
    uint256 c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal constant returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }
```

```
    function sub(uint256 a, uint256 b) internal constant returns (uint256) {
        assert(b <= a);
        return a - b;
    }

    function add(uint256 a, uint256 b) internal constant returns (uint256) {
        uint256 c = a + b;
        assert(c >= a);
        return c;
    }
}
```

三、 失效的权限控制

1. 风险描述

由于权限控制在设计或编码过程中的疏忽和缺陷，导致本来应该限制使用范围的重要函数或权限没有控制好范围，从而被攻击者调用这些重要函数或者使用权限实施攻击。

2. 危害描述

过去 7 年时间里，权限控制失效类问题导致的区块链安全事件共有 6 起，但导致的直接经济损失达 2.02 亿美元。其中，最著名就是以太坊钱包 Parity 的库权限没有控制好，被越权调用初始化函数，重置了钱包的所有者，导致了 1.68 亿美元的直接经济损失。

3. 实际案例

案例 3-1：

导致 Parity 钱包用户损失 1.68 亿美元的就是下面这几行代码，开发者不应该把初始化函数设置为 public，让其它合约也可以调用。攻击者利用了这个漏洞将钱包的所有者修改，以致用户损失巨大。

```
function initContract() public {
    owner = msg.sender;
}
```

更多详细的介绍可以参考下面的链接：

https://www.parity.io/the-multi-sig-hack-a-postmortem/

案例 3-2：

另一种情况是编译器漏洞，如果在编写构造函数过程中不注意也会导致越权调用。在小于 0.4.22 版本的 solidify 编译器语法要求中，合约构造函数必须和合约名字相等，名字受到大小写影响。如：

```
contract Owned {
    function Owned() public{
    }
```

而在 0.4.22 版本以后，引入了 constructor 关键字作为构造函数声明，但不需要 function

```
contract Owned {
    constructor() public {
    }
```

如果没有按照对应的写法，构造函数就会被编译成一个普通函数，可以被任意人调用，会导致 owner 权限被窃取等更严重的后果。更多详细的介绍可以参考链接：
https://paper.seebug.org/741/

案例 3-3：

Call 是最常用的调用方式，调用后内置变量 msg 的值会修改为调用者，但执行环境为被调用者的运行环境，如果合约中有函数以 msg.sender 作为关键变量的代码逻辑，比如向 msg.sender 转账，那么就会导致越权操作，引发安全问题。如下代码所示，withdraw() 函数设计的初衷为只能有合约拥有者和合约本身可以发起取款的操作；但由于 call 的问题，只要用户精心拼接字符序列调用 call，从而调用 withdraw() 函数，就可以绕过 isAuth() 并取款。

```
function callFunc(bytes data) public {
    this.call(data);
```

```
        //this.call(bytes4(keccak256("withdraw(address)")), target); //利用代码示意
}
```

```
function withdraw(address addr) public {
        require(isAuth(msg.sender));
addr.transfer(this.balance);
}
```

```
function isAuth(address src) internal view returns (bool) {
        if (src == address(this)) {              //检查是否合约拥有者或合约本身
                return true;
        }
        else if (src == owner) {
                return true;
        }
        else {
                return false;
        }
}
```

4. 修复方案

- 对初始化等重要函数不要设置为 pubic 权限，以致可以被外部调用。
- 要注意构造函数编写方式，以免被编译成普通函数，可以被任意调用。
- 注意 call 等的调用对 msg 的改变，以免导致绕过验证环节，被越权执行函数。
- 参考 OWASP ProActive Controls 项目中有关访问控制的内容。
- 参考 OWASP ASVS 项目中有关访问控制的内容。
- 参考 OWASP 测试指南项目中有关认证测试和授权测试的内容。

四、 不安全的共识协议

1. 风险描述

共识协议由于存在某些设计之初未考虑到的漏洞，导致漏洞可能被攻击者识别和利用，从而损害链上参与者的利益。

2. 危害描述

在总共 160 个安全案例中，有 15%（24 个）是共识机制漏洞导致的安全事件，远高于本 Top 10 中其它类型的安全事件，可见这类漏洞是攻击者最为关注的。导致的直接经济损失为 3823 万美元，与被攻击的区块链规模较小，使用范围较窄有关，往往处于在虚拟或还没发展壮大的时候（比如：Krypton）或者是衰落的时候（比如：ETC）。共识机制本身的健壮性受到了很大的影响，容易被攻击。

在共识类案例中，以 51%共识攻击最为常见。而不同类型的共识协议在面对 51%攻击的抵抗力又有所差别，虽然 POW 和 POS 两种共识协议在规模较小时都容易被攻击，但是 POW 类共识协议比 POS 类更容易被攻击，从攻击的角度来看，租用 51%算力的成本往往比控制 51%权益的成本更低，更可控。

3. 实际案例

案例 4-1：

Krypton 是一个基于以太坊的区块链，2016 年 8 月的时候被 51%攻击了，这次的攻击分为两部分，一部分用至少 51%的算力回滚交易，以实现同一个币支付两次；另一部分是用 DDOS 攻击网络中的多个节点。这次攻击者从 Nicehash 租用额外的哈希

能量，并且至少租用了 4 个矿池来实施攻击。我们由此可以看到，矿池的出现让算力实现了中心化，同时这些算力资源还是可以被任意租用的，这也使得攻击门槛大为降低。因此相比 POW 类共识协议，一般来说 POS 类共识协议更为安全。

更多详细的介绍可以参考链接：

https://cryptohustle.com/krypton-recovers-from-a-new-type-of-51-network-attack

案例 4-2：

2019 年 1 月，ETC 遭到 51% 算力攻击。各大交易所（Coinbase，Gate.io，Bitrue 等）都报导出现双花的交易。（在 51% 算力攻击中还有一个关键概念是所谓的网络重组（Block Reorganization），就是网络中六到八个块甚至更长的块都已经被矿工挖出来了，但是这些块突然全部被移除，产生了一些新的块去代替这些已经被确认过的块。）黑客在一个网站上租用到一定规模的算力（超过 51%），然后向 ETC 网络发起攻击，利用网络重组之前的块往交易所进行两笔数额较大的充值，赶在网络重组之前把币兑现，然后使用自身超过 51%的算力算出新的块来代替已确认的块以及块中的交易，使得那些被替换的块里的交易无法追溯。

Attacker's Deposits Onto Bitrue

Deposit (ETC)	TXID	Block Height	Attacker Address	Wallet Address
4,000	0xbba16320ec10701e615db5a65d47a8714b097c1cda3d1874793edb9540babcd1	7254430 0x6eb19e	0x3ccc8f7415e09bead930dc2b23617bd39ced2c06	0x2c9a81a120d11a4c2db041d4ec377a4c6c401e69
9,000	0xfe2da37fd908edf2411b39c1ec36361246b14e551f4dd9cc95eb17fac375cb8d	7255212 0x6eb4ac	0x3ccc8f7415e09bead930dc2b23617bd39ced2c06	0x2c9a81a120d11a4c2db041d4ec377a4c6c401e69

更详细的内容请参考以下链接：

https://medium.com/@AnChain.AI/anchain-ai-partners-with-bitrue-exchange-to-secure-blockchain-ecosystem-945580d1dd3f

https://www.bishijie.com/shendu_16552

4. 修复方案

51%的攻击会让交易所等参与交易的对手方损失惨重。针对交易所，建议提升提款前的确认次数，比如：Krypton 就将他们的提款时间增加到了 1000 次确认。

对共识协议来说，小规模的 POW 类数字货币，以及权益集中度高的 POS 类数字货币容易更容易受攻击，总体来说 POW 比 POS 更容易被攻击，建议根据具体的业务场景，谨慎选择共识协议。

五、 考虑不充分的程序逻辑

1. 风险描述

由于软件设计和编码的错误原因，存在没有考虑到的异常分支，导致程序逻辑可以被攻击者利用，以至于陷入设计者未预期的流程，造成重大损失。

2. 危害描述

很多区块链安全研究者都把这类会导致重入的程序逻辑问题列为头号风险，主要还是因为当年的 The DAO 事件（损失 6000 万美元）影响太大。不过，除了 2016 年的 The DAO 事件外，此后并没有发生其它有比较严重损失的案例，其它的 4 个案件总共也只损失不到 7.14 万美元，这就是我们把这类事件只排在第 5 的原因。

3. 实际案例

案例 5-1：

以 The DAO 事件为案例，首先要介绍两个背景知识，ETH 智能合约的发送过程和 fallback 函数。ETH 智能合约发送过程，在智能合约中用代码向某个地址（这个地址可以是人，也可以是智能合约）发送以太币，比较常见的两个方式是：一是调用 send 函数，比如：msg.sender.send(100)；二是使用 message call，msg.sender.call.value(100)。这两个方式不同的是发送的 gas 数量。当调用 send 方法时，只会发送一部分 gas，一般是 2300gas，一旦 gas 耗尽就可能抛出异常。而使用 message call 的时候，则是发送全部的 gas，执行完之后剩余的 gas 会退还给发起调用的合约。fallback 函数，智能合约中可以有唯一的一个未命名函数，称为 fallback

函数。该函数不能有实参，不能返回任何值。如果其他函数都不能匹配给定的函数标识符，则执行 fallback 函数。当合约接收到以太币但是不调用任何函数的时候，就会执行 fallback 函数。

如下两段代码是有重入漏洞的合约 Bank 和攻击合约 Attack，攻击实施的基本流程如下：

（1）假设 Bank 合约中有 100wei，攻击者 Attack 合约中有 10wei；

（2）Attack 合约先调用 deposit 方法向 Bank 合约发送 10wei；

(3)之后 Attack 合约调用 withdraw 方法，从而调用 Bank 的 withdrawBalance；

（4）Bank 的 withdrawBalance 方法发送给了 Attack 合约 10wei；

（5）Attack 收到 10wei 之后，又会触发调用 fallback 函数；

（6）这时，fallback 函数又调用了两次 Bank 的 withdrawBalance，从而转走了 20wei；

（7）之后 Bank 合约才修改 Attack 合约的 balance，将其置为 0。

从这个攻击流程中，我们可以看出，出问题的地方在 4-7 这几个步骤，开发者原本预期步骤 4 完成后就会到步骤 7，没有预料到中间还有 5-6 的 fallback 处理这个流程，以至于攻击者可以构造编写 fallback 代码，让处理流程走入异常分支，实现递归调用，直到 gas 耗尽。

```
contract Bank{

    mapping(address=>uint) userBalances;

    function Bank() payable{
        uint a = 1;
    }

    function getUserBalance(address user) constant returns(uint) {
      return userBalances[user];
    }

    function addToBalance() payable{
      userBalances[msg.sender] = userBalances[msg.sender] + msg.value;
    }

    function withdrawBalance() {
      uint amountToWithdraw = userBalances[msg.sender];
      if (msg.sender.call.value(amountToWithdraw)() == false) {
          throw;
      }
      userBalances[msg.sender] = 0;
    }

    function getBalance() returns (uint){
        return this.balance ;
     }
}

```

```
contract Attack{

    address addressOfBank ;
    uint attackCount ;

    function Attack(address addr) payable{
        addressOfBank = addr ;
        attackCount = 2 ;
    }

    function() payable{
        while(attackCount > 0){
            attackCount -- ;
            Bank bank = Bank(addressOfBank) ;
            bank.withdrawBalance() ;
        }
    }

    function deposit(){
        Bank bank = Bank(addressOfBank) ;
        bank.addToBalance.value(10)();
    }

    function withdraw(){
        Bank bank = Bank(addressOfBank) ;
        bank.withdrawBalance();
    }

    function getBalance() returns (uint){
        return this.balance ;
    }

}

```

4. 修复方案

注意代码中像 fallback 一类的隐含分支，避免处理逻辑出现异常。在编写智能合约进行以太币发送的时候，使用 send 或者 transfer，不要使用 message call 的方式。更多详细的信息可以参考如下链接：

https://blog.csdn.net/u011721501/article/details/79450122

六、 不严谨的业务策略

1. 风险描述

目前比较突出的一种业务策略类问题是利用高额的以太费用和技术手段让以太坊堵塞，从而获得游戏中的大奖。但根据场景不一样，会有许多不一样的业务问题，这些也都会对产品的公平性，乃至产品的生存造成重大影响，需要特别注意！

2. 危害描述

目前各类区块链游戏是人气比较高的，因为区块链的去中心，更有利的确保了游戏规则公正公平，吸引了不少玩家。但如果业务策略设计有漏洞，就容易导致公平性丧失，游戏奖励被特定的人利用漏洞来获得。2018 年共发生了 2 起业务策略漏洞类的安全事件，共导致了 359 万美元的直接经济损失，都发生在 FOMO3D 上。

3. 实际案例

案例 6-1：

以 FOMO3D 为例，其游戏规则如下：

（1）规则 1：最后一个购买 KEY 的人获得奖池中的大奖；

（2）规则 2：每有人购买一个 KEY，倒计时时间会增加 30 秒；

（3）规则 3：游戏启动后从 24 小时开始倒计时。

曾经，大部分人都认为这是一个不可能结束的游戏，因为一旦快到倒计时结束的时候就会很多人蜂拥而至去购买 KEY。但是 2018 年 8 月 22 日下午 14 点 48 分，这个游戏终止了，一个地址获得了共 291 万美元的奖励。那么他是怎么获得的呢？是因为运

气好，刚好在他购买之后没有其他人购买了，还是其它原因？分析认为这位获奖者是一位黑客，他在自己购买 KEY 后，使用高额的以太费用和技术手段让以太坊堵塞了 3 分钟，进而使得其他玩家无法打包购买 KEY 的交易，最终等到了倒计时结束，获得了最后的大奖！所以当业务策略设计时不考虑到以太坊网络是有可能被人为的拥堵住的话，你就可能付出和这个案例一样的代价。同时也让这个游戏没落和失败。更多详细的介绍可以参考如下链接：

https://mp.weixin.qq.com/s/s_RCF_EDlptQpm3d7mzApA

https://www.bcsec.org/index/detail/id/285

4. 修复方案

由于目前的几个主要公链的性能都比较差，使用很少的成本就能让交易网络阻塞，所以在设计业务策略的时候一定要考虑这些因素，攻击可能让交易出现阻塞和回滚，以至于影响游戏公平性。

七、 校验不严格的交易逻辑

1. 风险描述

交易的校验逻辑不够严密，以致攻击者可以构造假的交易行为，但却被校验方验证通过，误认为真的交易行为。或者提交了真的交易操作，但是通过时间差，黑名单等因素，让交易操作产生回滚来实现，实际上的交易动作并没有完成，而商家仅验证是否有交易行为的话，就可能导致资金上的损失。

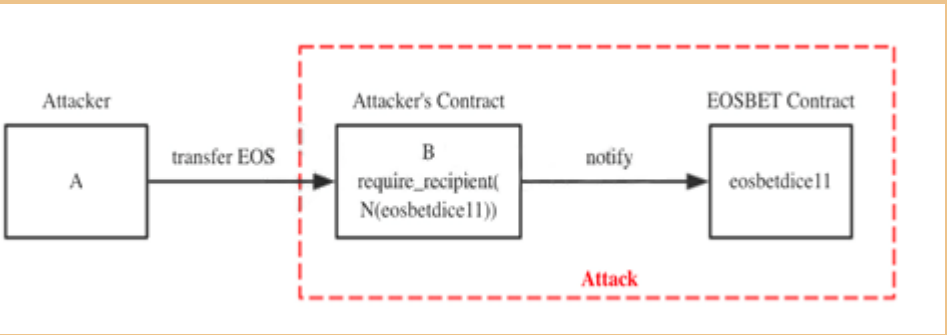
2. 危害描述

这里问题通常包含两大类——假转账类问题和回滚类问题，截止 2019 年 1 月共发生了 26 起，损失金额 233.72 万美元，大多数在于校验方的校验逻辑存在问题，从而可以被攻击者绕过校验，以假的充值行为却获得真实的数字货币入账；或者通过时间差，黑名单等因素让之前的交易操作回滚，从而欺骗商户获得商品。从古典互联网的安全案例来看，这类漏洞很常见，也是容易造成大额经济损失的漏洞，预计在区块链领域也不例外。

3. 实际案例

案例 7-1：

EOSBET 案例是假转账攻击案例的典型代表，账号 A 和账号 B 都是由攻击者控制的账号，攻击者使用账号 A 给帐号 B 转账，正常的转账只有账号 A 和帐号 B 能收到转账通知。然而攻击者在账号 B 上部署了合约，并将这个转账消息转发给 EOSBet 合约，这样 EOSBet 也会收到 A 给 B 的转账通知。关键的问题是，EOSBet 合约。



（图示：假转账攻击过程 ）

在收到转账通知后，并没有校验 transfer 中的 to 是否为_self，就将其错误判断为一笔正常转账，然后根据平台游戏规则给了账号 A 发送相应的 EOS 奖励，但实际上，账号 A 和账号 B 都是黑客的账号，黑客正是通过控制两个不同账号互相转账，就这样以“零成本”骗取了平台巨额奖励。

https://www.chaindd.com/3135554.html

案例 7-2：

回滚类攻击是利用交易广播的时间差来实施攻击，达到后提交的交易先被打包的效果。区块链的底层网络基于 P2P 网络，正常情况下 P2P 网络的交易广播涉及到节点发现和路由，速度比较慢，而攻击者可以对它的交易广播路径进行优化，对当前的出块节点实现直连，不再经过节点发现和 P2P 路由阶段，那么它的网络速度就会快很多，很可能比先提交的但通过正常网络通信的交易更快到达出块节点，实现后发先至，从而提前被打包。从攻击场景来说，攻击可以先通过正常网络成功发起一个交易，然后再通过优化后的网络发起一个将当前余额清零或者其他可以让上一个交易不成功的攻击交易，当攻击交易比正常交易先被打包后，正常的交易条件不再满足，交易被回滚，攻击者没

有任何的支出。但很多 DAPP 的处理只会判断交易是否被成功发起，不判断后面是否出现了回滚，这样它们仍会认为交易已成功，而给攻击者发放对应的数字商品。

案例 7-3：

BetDice 是所有案件中损失最大的一个，攻击手法分为两种，一种是利用时间差来实现如上节所述的回滚攻击，另一种是利用黑名单功能，提交不可能被打包的交易。利用时间差的手法：如上文所述交易广播是通过 P2P 网络一个节点一个节点广播的，扩散是比较缓慢的，路径并不一定是最优的，存在通过最优路径后发先至的可能，也就是说即使节点 A 发送交易 a 比 节点 B 发送交易 b 要早那么一点点，但是节点 B 跟当前出块节点直连，那么交易 b 会比交易 a 先被广播到出块 BP 并且被打包。具体的攻击流程如下所示：

（1）攻击者往游戏开奖节点下注，发起下注交易；

（2）然后立刻往当前出块节点的 seed 发送转账清空账号余额的交易，以便于比下注交易更快被打包；

（3）此时，开奖节点不知道攻击者的账号已经清空了（ 因为该节点还不知道有情况账号余额的交易 ），所以下注交易在开奖节点成功了（而当下注交易广播到出块 BP 时，因为攻击者账号余额为空，所以下注交易失败，所以攻击者无成本）；

（4）当游戏方直接根据节点的 state 获取开奖数据，但经过上面的分析我们知道，这个 state 其实是需要被回滚的，所以该数据不能为准。但游戏方还是根据这个数据进行开奖了，而开奖交易发送到 BP， 因为 DAPP 的合约里没有对下注单是否存在进行判断，所以打包成功，游戏方转账给攻击者。

利用黑名单的手法：ECAF 是 EOS 的一个仲裁组织，对一些恶意事件发起仲裁，对一些恶意账号会列入黑名单，并下发给各打包节点，恶意账号就不能进行交易，攻击者利用了 ECAF 的这个特性实施了攻击，具体攻击流程如下所示：

（1）attack 合约部署攻击合约；

（2）用 blacklist 发起转账请求，推送下注交易；

（3）此时游戏节点执行成功并开奖但 出块节点 BP 发现黑名单不予打包；

（4）则下注交易回滚，但开奖交易依旧打包成功。

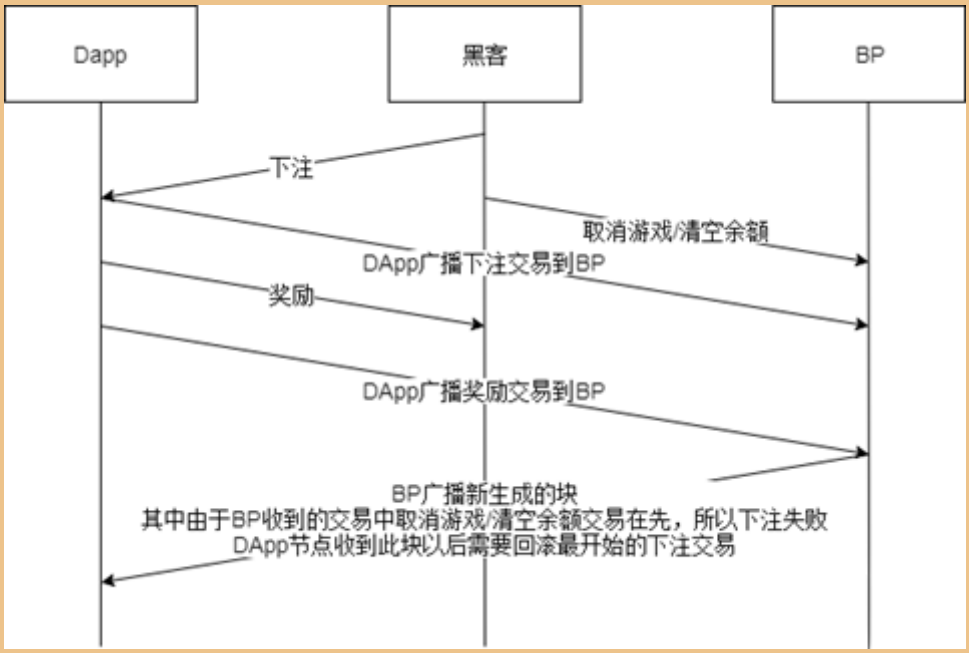
更多详细的介绍，可以参考如下两个链接的文章：

https://eos.live/detail/19215

https://eos.live/detail/19255

案例 7-4：

北京时间 2018 年 12 月 19 日凌晨，EOS 网络中，包括 BetDice，Royal Online Vegas 在内的数个游戏 DApp 遭受黑客攻击，损失数十万枚 EOS 通证。 本次事件是由于部分游戏 DApp 为增强游戏体验，自建节点运行 DApp，游戏的奖励结算完全基于本地 EOS 节点的交易记录。由于自建节点的交易存在回滚的可能，黑客就利用了 BP 与自建节点的交易时间差完成了回滚攻击，无成本赢得奖励。此部分的具体攻击如下图所示：



（来源：AnChain.ai）

4. 修复方案

- 关键逻辑不仅要校验是否真实的转账消息，还需要校验转账的目标账户是否正确。
- 不仅要确认支付是否成功提交，还需要等待一定时间，看支付是否回滚，再确认支付是否成功。
- 游戏的奖励机制应该同下注机制紧密相关。一旦发现下注交易产生问题发生回滚，则奖励交易也应该一起回滚。

八、 脆弱的随机数机制

1. 风险描述

这里指的不仅仅是伪随机函数的问题，而是整个随机数生成机制不够安全，导致可以被攻击者提前获取或猜测到随机数的结果，以实施攻击。

2. 危害描述

随机数问题也是近期比较常发的安全事件，特别是随着一些智能合约类游戏的爆发。随机数作为游戏公平性的保证，一旦有漏洞就会导致游戏机制被破坏，就可以从中获利，所以常常被各种攻击者盯上，以至于 2018-2019 年就发生了至少 15 起案件，包括 2018 年非常火热的 Last Winner 和 Fomo3D 项目，部分 DAPP 还因为安全事件而关闭。虽然总的损失金额不大，只有 194 万美金，但跟这些游戏的规模都还不大有关系，黑客在这些游戏早期就开始关注并开始攻击窃夺资金，以至于他们没有机会长大。另外，攻击数量多也说明弱随机数攻击的门槛不高，以至于很多黑客都可以发起攻击，所以区块链开发者需要特别注意弱随机数带来的安全风险。

3. 实际案例

案例 8-1：

如下是 Fomo3D 判断是否空投奖励的代码，像 last winner 这类山寨 Fomo3D 的游戏基本也都是类似的代码：

```
function airdrop() private view returns(bool)
{
    uint256 seed = uint256(keccak256(abi.encodePacked(
    (block.timestamp).add
```

```
    (block.difficulty).add
    ((uint256(keccak256(abi.encodePacked(block.coinbase)))) / (now)).add
    (block.gaslimit).add
    ((uint256(keccak256(abi.encodePacked(msg.sender)))) / (now)).add
    (block.number))));
    if((seed - ((seed / 1000) * 1000)) < airDropTracker_)
        return(true);
    else
        return(false);
}
```

随机数 seed 由 timestamp、difficulty、coinbase、gaslimit、msg、sender、number 构成。因此，只要知道这几个数据，我们就可以预测出是否可以获得空投奖励。其中 msg.sender 是可以由用户或攻击者控制，因此攻击者可以不断尝试构造出满足空投奖励条件的 msg.sender 以获得比正常用户高很多的空投奖励。其中在 last winner 中，攻击者在 6 天内就获得了 170 万美金的空投奖励。更多详细的内容可以参考如下链接：

https://www.reddit.com/r/ethereum/comments/916xni/how_to_pwn_fomo3d_a_beginners_guide/

https://paper.seebug.org/672/

4. 修复方案

1) 一般来说随机数的构成元素不应该可以被用户控制；

2) 不应该在开奖前让用户可以预测到开奖结果。

九、 存在缺陷的激励机制

1. 风险描述

激励机制是区块链的重要环节，但若开发者设计的激励机制存在可以被利用的漏洞，就可能会给开发者或用户带来损失。

2. 危害描述

激励机制跟报酬的多少直接相关，因此会被很多的攻击者盯上，带来不少的安全风险，有漏洞的激励机制将会给团队带来巨大的经济损失。Eligius 矿池遭受的“块代扣攻击”就让他损失了 120 万美元。攻击方利用了矿池激励机制的漏洞，参与分红，但是不做贡献，挖到了矿就自己藏起来，给一起挖矿的矿工和矿池都带来了损害，长期来看矿池的吸引力也会受到影响。

3. 实际案例

案例 9-1：

2014 年的时候，Eligius 矿池遭受“块代扣攻击”，损失了 120 万美元。块代扣攻击是指，一个恶意的矿工可以运行一个自己开发的挖矿软件，该怎么挖怎么挖，但是碰到真正解的话就扣下不提交给矿池，因为从矿池方看来，他们似乎正在为矿池工作，并继续得到分配的报酬，而实际上，并不是真正的为矿池做有益的贡献。对矿池和矿工来说，在不同的激励机制下，他们受到的危害也不一样。如果矿池是按照工作量给钱的，那么这种攻击不会伤害其他矿工，却会从矿池老板那里白领报酬。如果矿池是按比例分账的，那么块代扣攻击就危害其他矿工的利益，因为他分酬劳而不做贡献。

由于 BTC 挖矿机制的设置，这种攻击并没有办法让攻击者独吞扣下的挖矿奖励，因为他挖的是矿池指定的块，而这些块的奖励只能指向矿池。所以这类的攻击还不算多，但我们在设计其它激励机制的时候一定好仔细考虑，以防出现可以被利用的漏洞，造成重大损失。

https://www.8btc.com/article/20341

4. 修复方案

从上面的案例来看，我们在设计激励机制时千万不要假设所有的参与者都是诚实的，一定要防范恶意者的攻击，以保护大家的利益。

十、 日志记录和监控不足

1. 风险描述

日志记录和监控的问题对区块链团队和公司来说非常很重要。如果没有足够的日志记录和监控，被攻击的过程中，你会不知道，不能及时止损。发生安全事件后，你没有办法调查，也没有办法改进以避免再犯同样的错误。所以日志记录和监控不足的问题也很严重！

2. 危害描述

缺少日志和监控会让黑客攻击变得更肆无忌惮，因为你无法发现攻击行为，无法知道是哪里存在安全漏洞，也就没有办法找到修复的方案，整体的安全风险处于失控的状态。

3. 实际案例

比如前文提到的 Mt.Gox（案例 1-1）等安全事件，黑客的攻击路径和手法并没有清晰的披露出来，若排除交易所本身不愿意披露这个因素外，更大的可能是交易所本身并没有完善的日志记录和监控，以至于无法复盘整个攻击流程，无法确切的知道黑客攻击行为的细节。

4. 修复方案

要确保所有登录、访问控制失败、输入验证失败都能够被记录到日志中去，并保留足够的用户上下文信息，以识别可疑或恶意帐户，并为后期取证预留足够时间；确保日志以一种

能被集中日志管理解决方案使用的形式生成；确保高额交易有完整性控制的审计信息，以防止篡改或删除，例如审计信息保存在只能进行记录增加的数据库表中；建立有效的监控和告警机制，使可疑活动在可接受的时间内被发现和应对；建立或采取一个应急响应机制和恢复计划。

后序

主编在研究区块链安全的过程中，发现并没有权威的指导性安全文档，所以联合了各区块链安全公司和各一线企业的安全专家，一起给出了区块链安全的 Top10，由于作者们时间和水平有限，如有任何错误的地方，或者更好的方案，请联系（项目组邮箱：project@owasp.org.cn），以便我们可以不断改进和提升文档的质量。

参考文献

【1】 https://www.bcsec.org/analyse

【2】 https://www.bcsec.org/event

【3】 https://xz.aliyun.com/t/3737

【4】 https://www.parity.io/the-multi-sig-hack-a-postmortem/

【5】 https://paper.seebug.org/741/

【6】 https://www.anquanke.com/post/id/147913

【7】 https://cryptohustle.com/krypton-recovers-from-a-new-type-of-51-network-attack

【8】 https://blog.csdn.net/u011721501/article/details/79450122

【9】 https://www.reddit.com/r/ethereum/comments/916xni/how_to_pwn_fomo3d_a_beginners_guide/

【10】 https://paper.seebug.org/672/

【11】 https://www.chaindd.com/3135554.html

【12】 https://ethereumdev.io/safemath-protect-overflows/

【13】 https://www.freebuf.com/vuls/169741.html

【14】 https://www.chaindd.com/3135554.html

【15】 https://eos.live/detail/19215

【16】 https://eos.live/detail/19255

【17】 https://www.bcsec.org/index/detail/id/134

【18】 https://www.bcsec.org/index/detail/id/223

【19】 https://mp.weixin.qq.com/s/s_RCF_EDlptQpm3d7mzApA

【20】 https://www.bcsec.org/index/detail/id/285

【21】 https://www.8btc.com/article/20341