整数上溢和下溢攻击实操

* uint256上溢

在solidity中，uint就是uint256，如果一个数已经是uint256所能表达的最大的数了，即2的256次方减1，那么再对其做加法就会出现上溢。

下面的例子

文本

描述已自动生成

对其进行编译、部署，然后查询number的值，如下

图形用户界面, 文本

描述已自动生成

再调用add方法对其进行加1操作，交易成功。

图形用户界面, 文本

描述已自动生成

然后再查看number的值

电脑萤幕的截图

描述已自动生成

从上图可以看到，number的值加1之后变成了0，即uint256的最小值，也就是说发生了上溢。

* 解决方案
  + 解决方案1

升级合约solidity的版本，升级到solidity v0.8.0以上，在solidity v0.8.0的breaking changes中，对上溢和下溢的处理有所变化

图形用户界面, 文本, 应用程序

描述已自动生成

从v0.8.0开始，算术运算对下溢和溢出进行revert。可以使用unchecked { ... }来使用以前版本的包装行为。检查溢出是非常常见的，所以solidity将其作为默认选项，以增加代码的可读性，即使它会略微增加gas成本。

那么接下来修改solidity的版本，然后执行add加1操作，就会引发revert，并且number的值不会变化。

电脑萤幕的截图

描述已自动生成

* + 解决方案2

依然用solidity v0.6.10，但是使用openzeppelin的SafeMath.sol library

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v3.4/contracts/math/SafeMath.sol>

add方法定义如下

图形用户界面, 文本

描述已自动生成

接下来就用在我们的合约中

图形用户界面

描述已自动生成

部署合约，执行add加1操作

图形用户界面, 文本

描述已自动生成

交易revert，且number的值没有变化，还是2的256次方加1

* 上溢攻击的例子

先看一个合约

文本

描述已自动生成

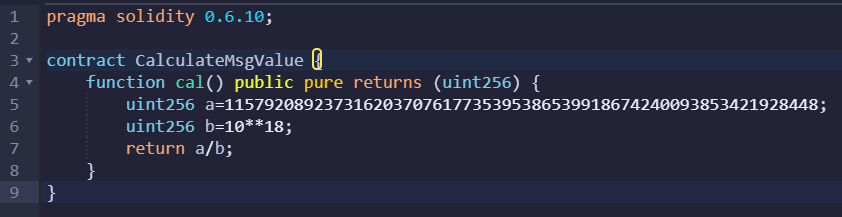
buy方法中的numTokens \* PRICE\_PER\_TOKEN存在乘法上溢的风险，PRICE\_PER\_TOKEN实际上就是1e18，即一个token的价格是1 ether。msg.value就是购买者发送的wei的数量，购买者如果存有恶意，可以花很少的wei，买很多的token，也就是说token的单价远远不及1 ether

首先通过5cb2账户部署合约，发送value为1 ether，部署后还剩98.999 ether。

再部署一个合约，专门用于计算(2\*\*256-1)/1e18，然后在计算的结果上加1，就得到buy方法的参数numTokens，这样就可以造成溢出，以低价买到tokens。

(2\*\*256-1)/1e18再加1的值为115792089237316203707617735395386539918674240093853421928448

接下来确定一下这个值乘以PRICE\_PER\_TOKEN即1e18，上溢后的值是多少，即确定msg.value的值，用一个合约来计算



文本

描述已自动生成

这就意味着可以用8284046750386698632065404255428212857888990415992086870360064 wei买到115792089237316203707617735395386539918674240093853421928448个token，每个token的单价为71 wei，远远低于1 ether，这样攻击者就大赚了。由于目前Javascript VM(London)环境中的账户没有这么多的ether，所以演示不了，但是攻击逻辑就是这么个逻辑。

解决方案同样是省级到v0.8.0以上或者用SafeMath.sol

* uint256下溢

看如下合约

文本

描述已自动生成

这个合约有无限取款的风险，因为无论何时require(balance[msg.sender] - amount >= 0)都是true，因为uint256-uint256始终非负。

模拟一下攻击：

用账户02db部署合约，并在部署时给合约转50 ether，然后查一下abaB的balance，期望为5

文本

描述已自动生成

目前账户abaB总共有100 ether，接下来用账户abaB发起多次取款，每次取1个ether，即1e18，可以取无限次，直到把合约中的50个ether全部取完。我发起了30次调用，取了30个ether，最后abaB的ether余额如下

图形用户界面, 网站

描述已自动生成

然后在合约中的balance为

图形用户界面, 应用程序

描述已自动生成

由此可以看到，在合约中的余额uint256下溢了，所以可以不断地取钱，直至把合约中的ether全部取光。

防范方法同样是升级到solidity v0.8.0或使用SafeMath.sol。

以上就是整数上溢和下溢攻击。