拒绝服务攻击和防范实操

关于什么是拒绝服务攻击，首先来看一个简单的例子，智能合约如下

文本

描述已自动生成

在合约A的foo方法中，向调用者发送1 ether，但如果调用者是一个合约，没有fallback方法，就无法接收ether。例子中，合约B的callFoo方法调用了合约A的foo方法，但是合约B没有fallback方法，无法接收ether，所以导致在调用合约A的foo方法的时候，foo方法执行失败，require失败，就无法执行接下来的逻辑。在这个例子中，合约B就在对合约A进行拒绝服务攻击，方式就是拒绝接收来自合约A的ether。在这个例子中似乎还不能看出这种攻击会带来什么后果，接下来我们看另一个例子，这个能够很清晰地说明拒绝服务攻击能够带来什么后果。

文本

描述已自动生成

上面的合约是一个竞选国王的合约，谁发给合约的ether多，谁就成为新国王，并把旧国王的余额退还给旧国王。比如，张三发1 ether给合约，成为国王，然后李四发2 ether给合约，合约把张三的1 ether退还给张三，李四成为新国王，以此类推。

表面上看起来逻辑好像没什么问题，但是实际上是存在拒绝服务攻击的漏洞的，接下来来实验一下通过下面两行代码执行拒绝服务攻击：

(bool sent,) = king.call{value: balance}("");

require(sent,"Failed to send Ether");

如果能通过上面两行代码执行拒绝服务攻击，那么下面两行代码永远得不到执行

balance=msg.value;

king=msg.sender;

* 攻击过程

实施攻击的手段很简单，使用一个没有fallback方法的合约来调用KingOfEther合约的claimThrone方法。攻击合约代码如下

文本

描述已自动生成

攻击合约定义了一个攻击方法attack，这个方法被定义被payable的，即可以接收ether。攻击逻辑是：用一个外部账户调用attack方法，并传入一些ether（大于KingOfEther的balance），然后调用kingOfEther的claimThrone方法声明为新国王，Attack合约成为新国王之后，其他人再想声明成为新国王就永远不能成功，因为Attack合约没有fallback方法，无法接收ether，(bool sent,) = king.call{value: balance}("");就会失败，那么设置新国王的代码永远不能执行，这样的话Attack合约就能永远留在国王的位置上。

接下来实际执行一下攻击过程，首先部署KingOfEther和Attack合约

图形用户界面, 应用程序

描述已自动生成

ddC4账户第一个申请成为国王，发送1 ether给KingOfEther合约

图形用户界面, 应用程序

描述已自动生成

文本

描述已自动生成

执行成功后，可以看到当前的国王是ddC4，余额是1 ether。

然后5cb2账户调用Attack合约的attack方法发起攻击，参数为KingOfEther合约的地址，发送2 ether（大于1 ether）

文本

描述已自动生成

执行攻击后，Attack合约成为了新国王，新国王的余额变成了2 ether。

然后，再让733c账户发送3 ether给KingOfEther合约，申请成为新国王，由于Attack合约执行了拒绝服务攻击，那么733c就不能顺利地成为新国王。

电脑萤幕的截图

描述已自动生成

从上图执行结果可以看到，交易执行失败，国王还是Attack合约。KingOfEther变成了不可用的状态，国王永远是Attack合约，其他人永远不能成为国王。

* 避免拒绝服务攻击的方案

KingOfEther合约采用了push的方式，即主动把ether退还给旧国王，这种方式存在问题。解决方案就是采用pull的方式，不主动把ether退还给旧国王，而是让旧国王自己来取回自己的资金。改造后的合约如下

文本

描述已自动生成

如图所示，在claimThrone方法中不会主动把旧国王的余额退还给旧国王，而是把其余额记录下来，让旧国王后续自己来取。取的方法就是通过withdraw方法。

在withdraw方法中，首先判断当前的国王不能取款，只能旧国王取款。然后把余额减为0，并把余额发送给旧国王。balances[msg.sender]=0;放在(bool sent,)=msg.sender.call{value: amount}("");之前执行的原因在于避免重入攻击（我已在另一篇文章讲了重入攻击）。

通过采用pull的方法，如果一个Attack合约执行拒绝服务攻击，那么并不能影响其他账户继续申请成为新国王，只能影响自己，导致自己取不出存进KingOfEther的ether，而不能影响其他任何人，通俗来说，就是偷鸡不成蚀把米。

总结：用pull，而不是push，就能够避免拒绝重入攻击。用pull取代push是在solidity中的一种很重要的设计范式。