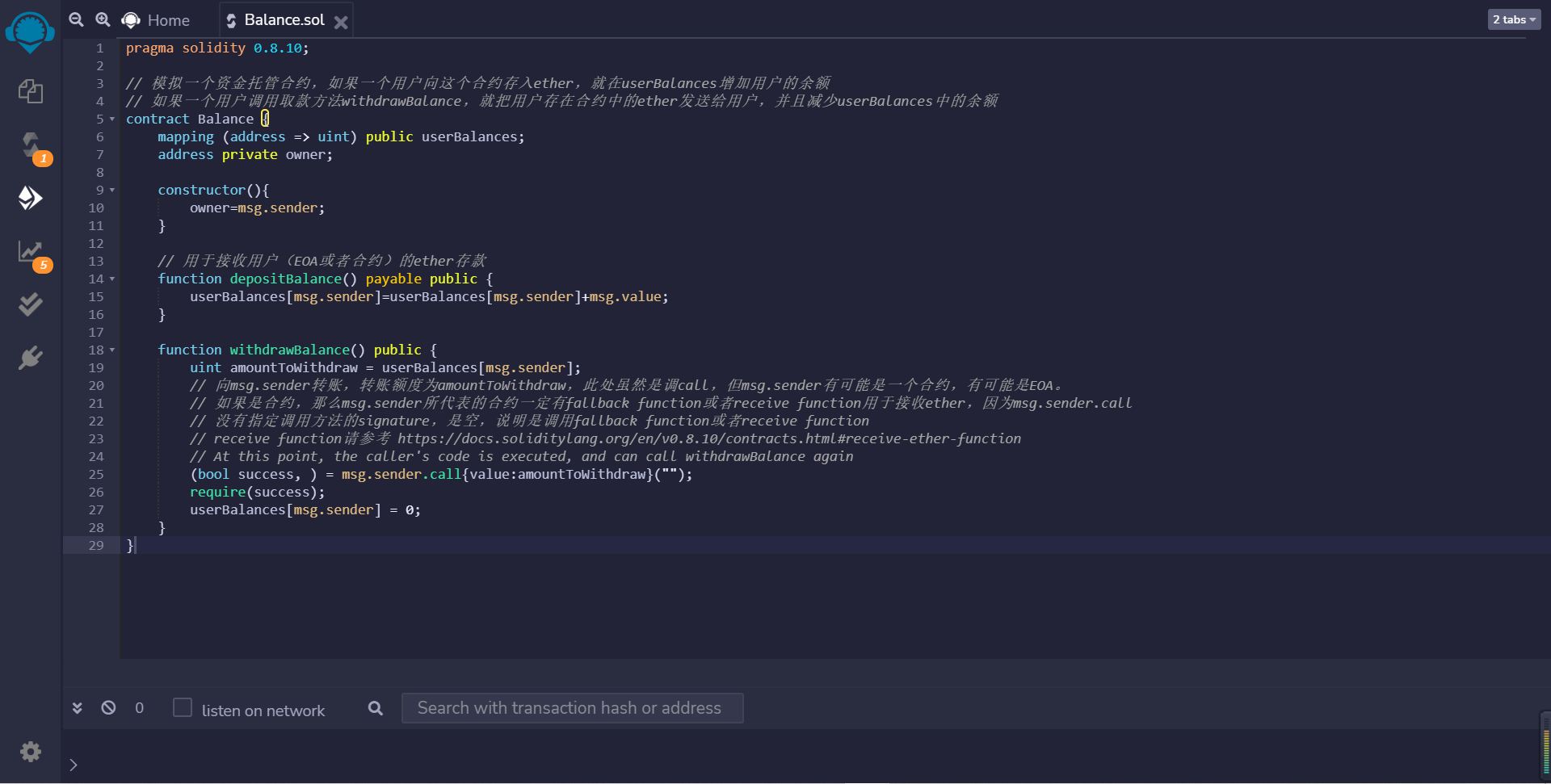
重入攻击和防范实操

* 攻击实操

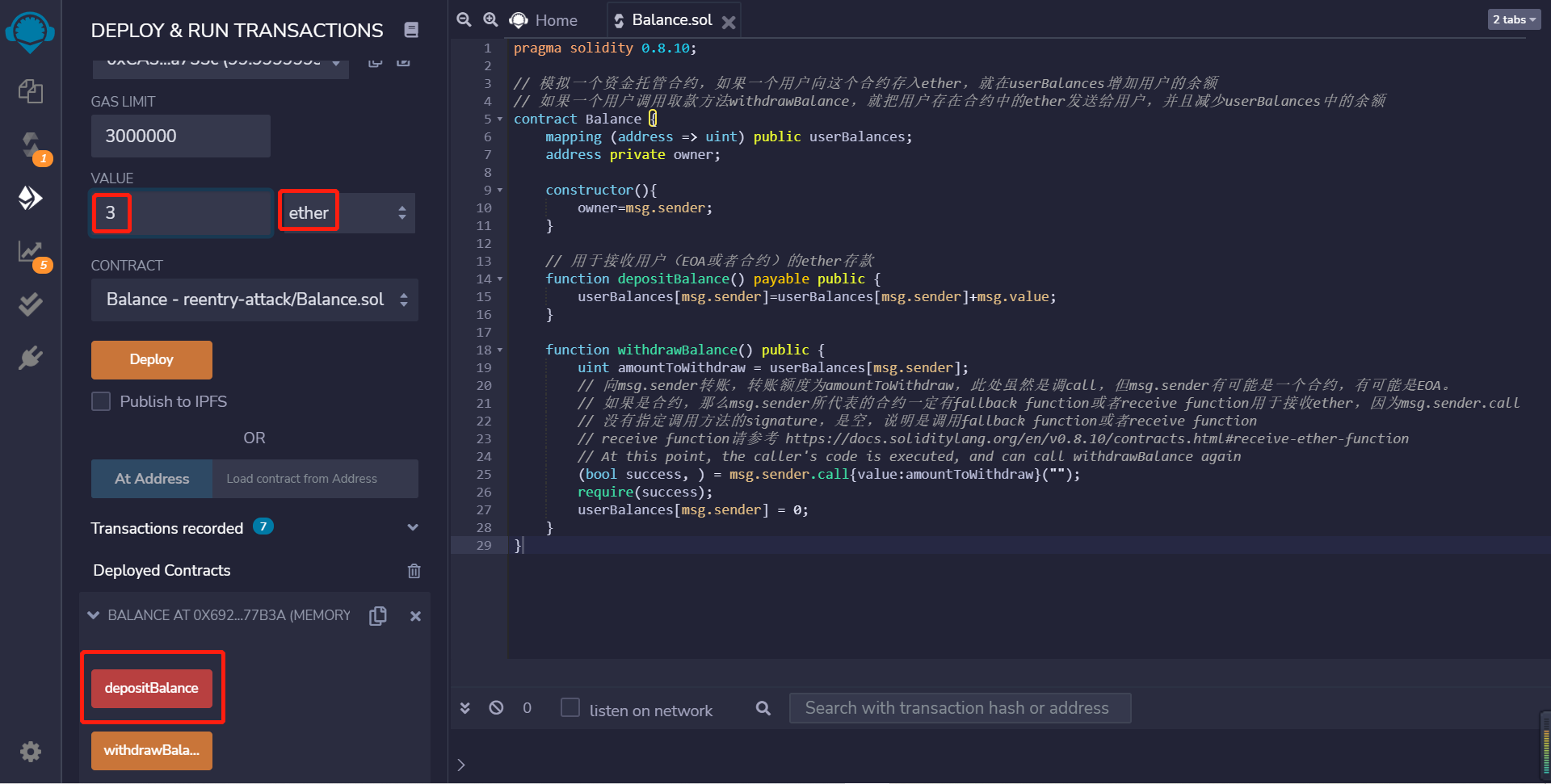
首先，写一个简单的ether托管合约，如下



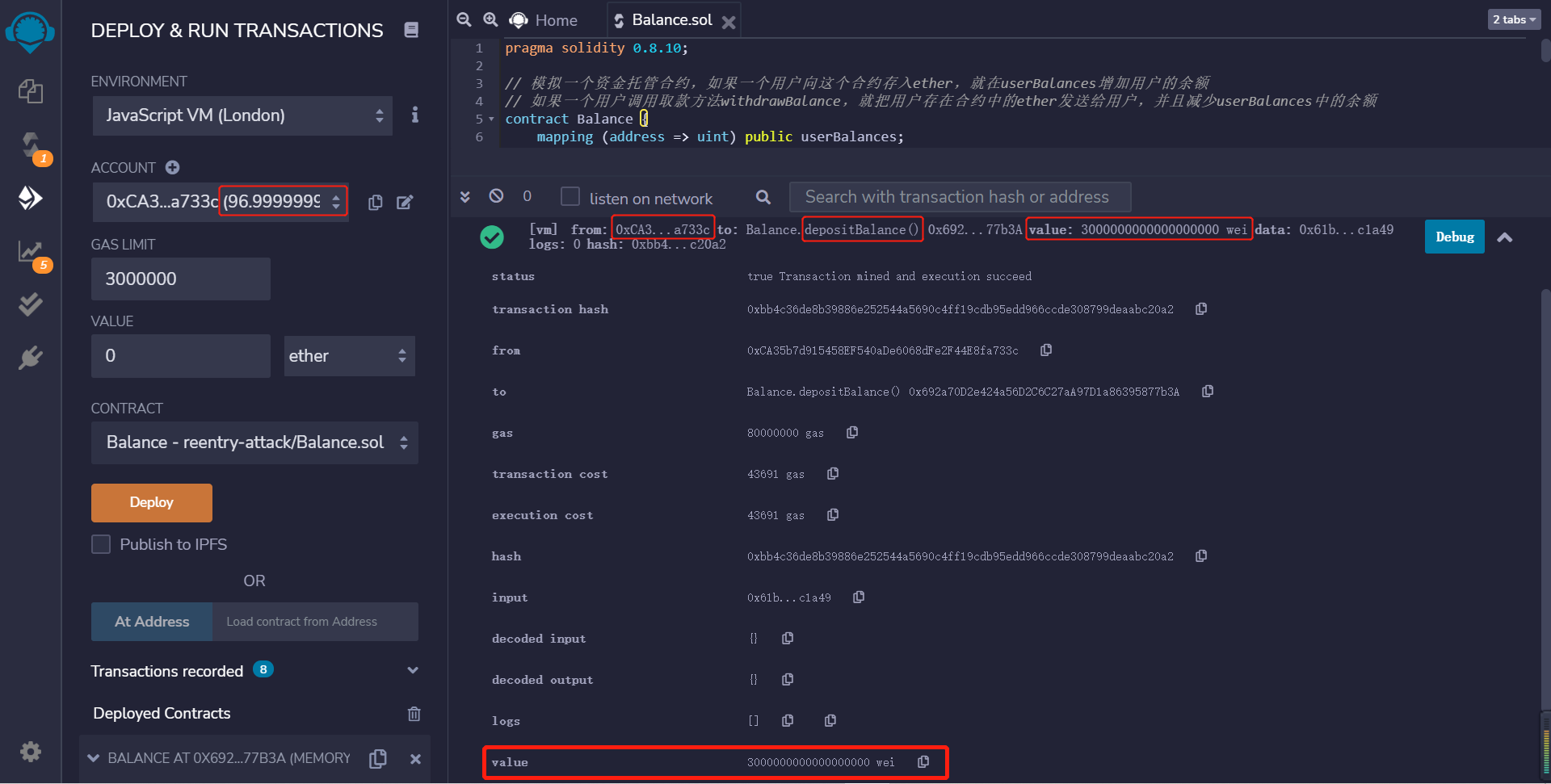
如果一个用户向这个合约存入ether，就在userBalances增加用户的余额；如果一个用户调用取款方法withdrawBalance，就把用户存在合约中的ether发送给用户，并且减少userBalances中的余额。

通过账户733c部署合约，部署前733c有100 ether，部署完成之后还有99.999999... ether

然后通过733c调用depositBalance，存3 ether到Balance合约。



执行完之后还剩96.99999... ether



查询一下733c在Balance合约中的余额

文本

描述已自动生成

符合预期，注意，以上余额是以wei为单位。

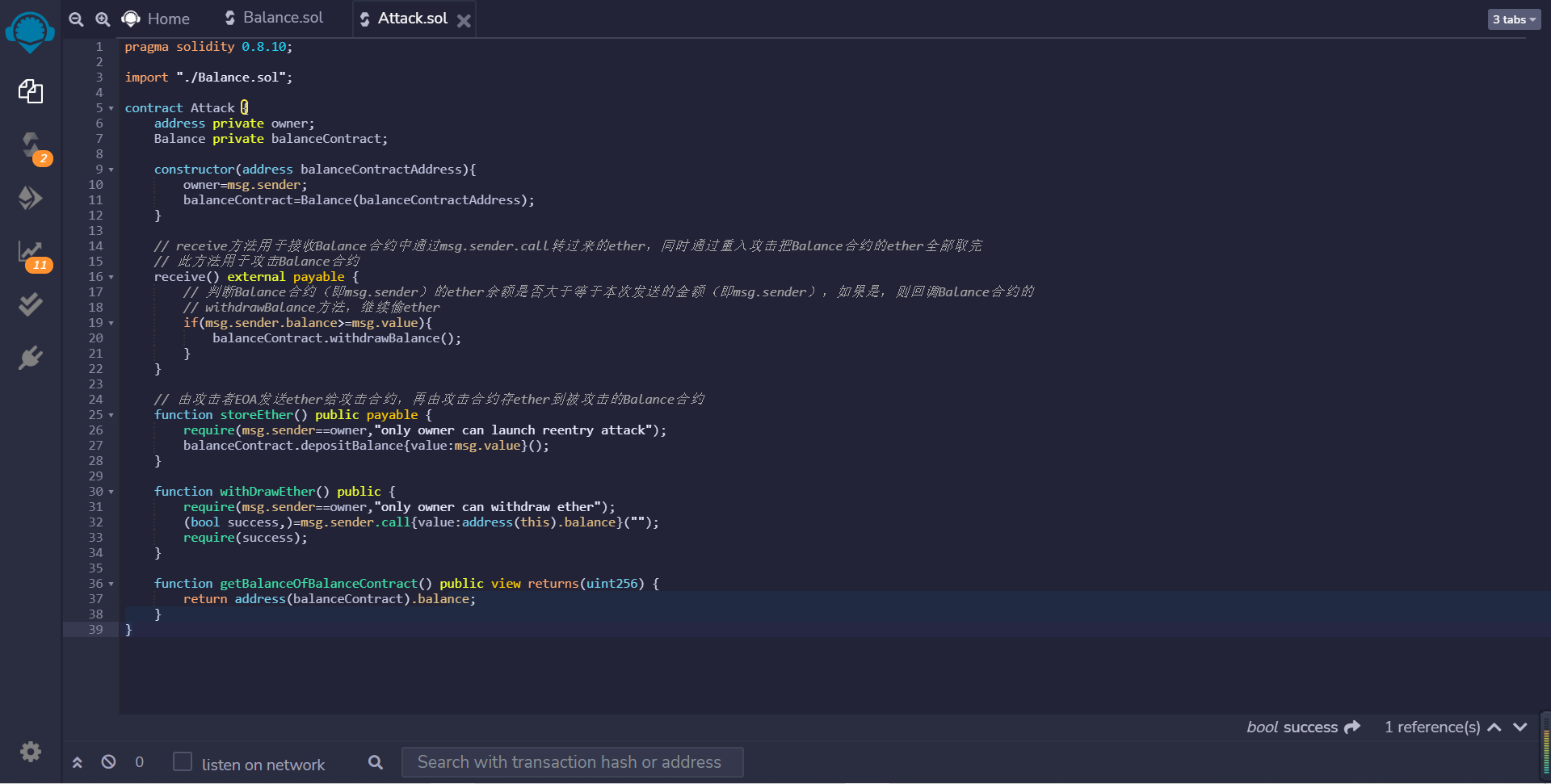
以此类推，要让账户02db存入4 ether，结果就不截图了。

至此，两个EOA账户已经总共向Balance合约存入了7 ether。

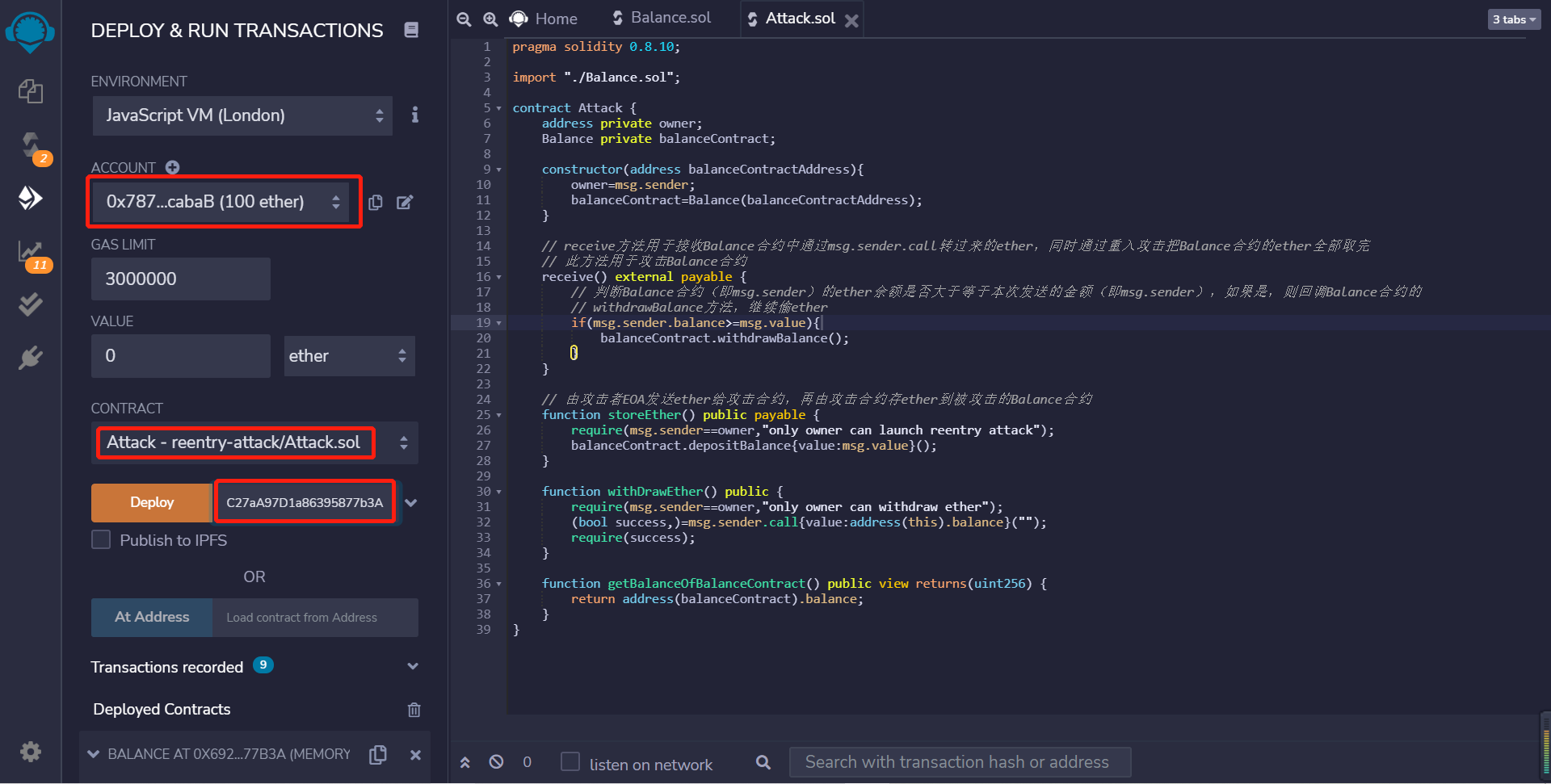
接下来，演示攻击的逻辑。

攻击者可以开发一个攻击合约，向攻击合约中存入1 ether，再由攻击合约向Balance合约存入1 ether，然后再由攻击合约调用Balance合约的withdrawBalance方法，用重入攻击的方式把Balance合约所拥有的ether全部取完。

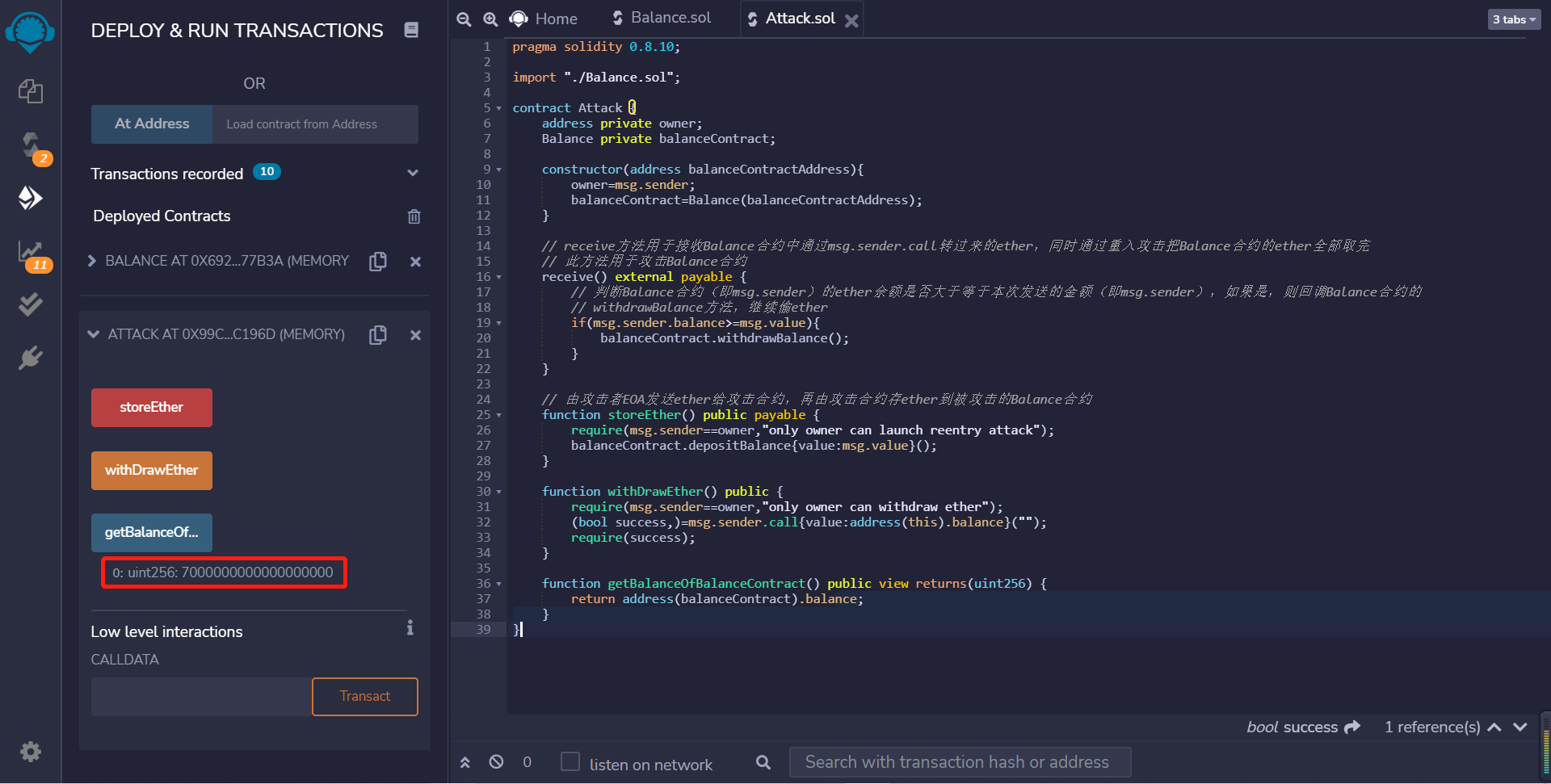
攻击合约的代码如下



假设攻击者账户是abaB，有100 ether，由其部署Attack合约，把Balance合约的地址作为部署参数。

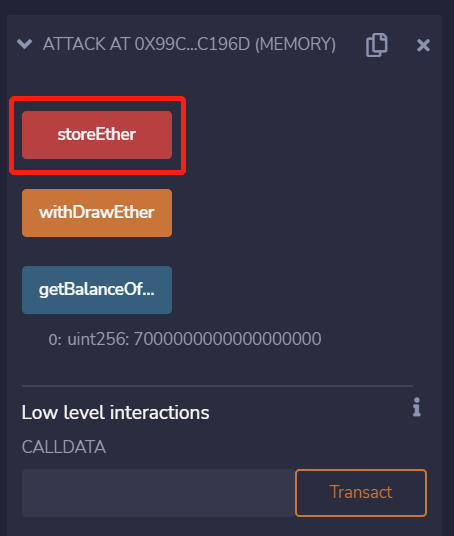
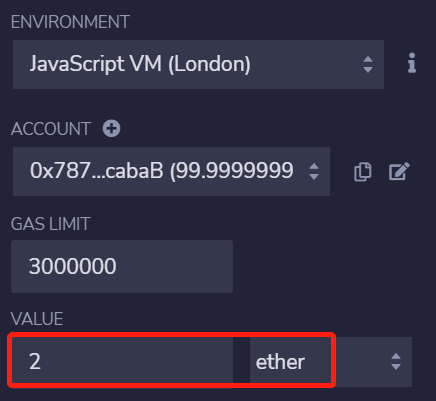


这时候，攻击者可以调用getBalanceOfBalanceContract查询一下被攻击合约Attack合约的ether余额，期望是7个ether。

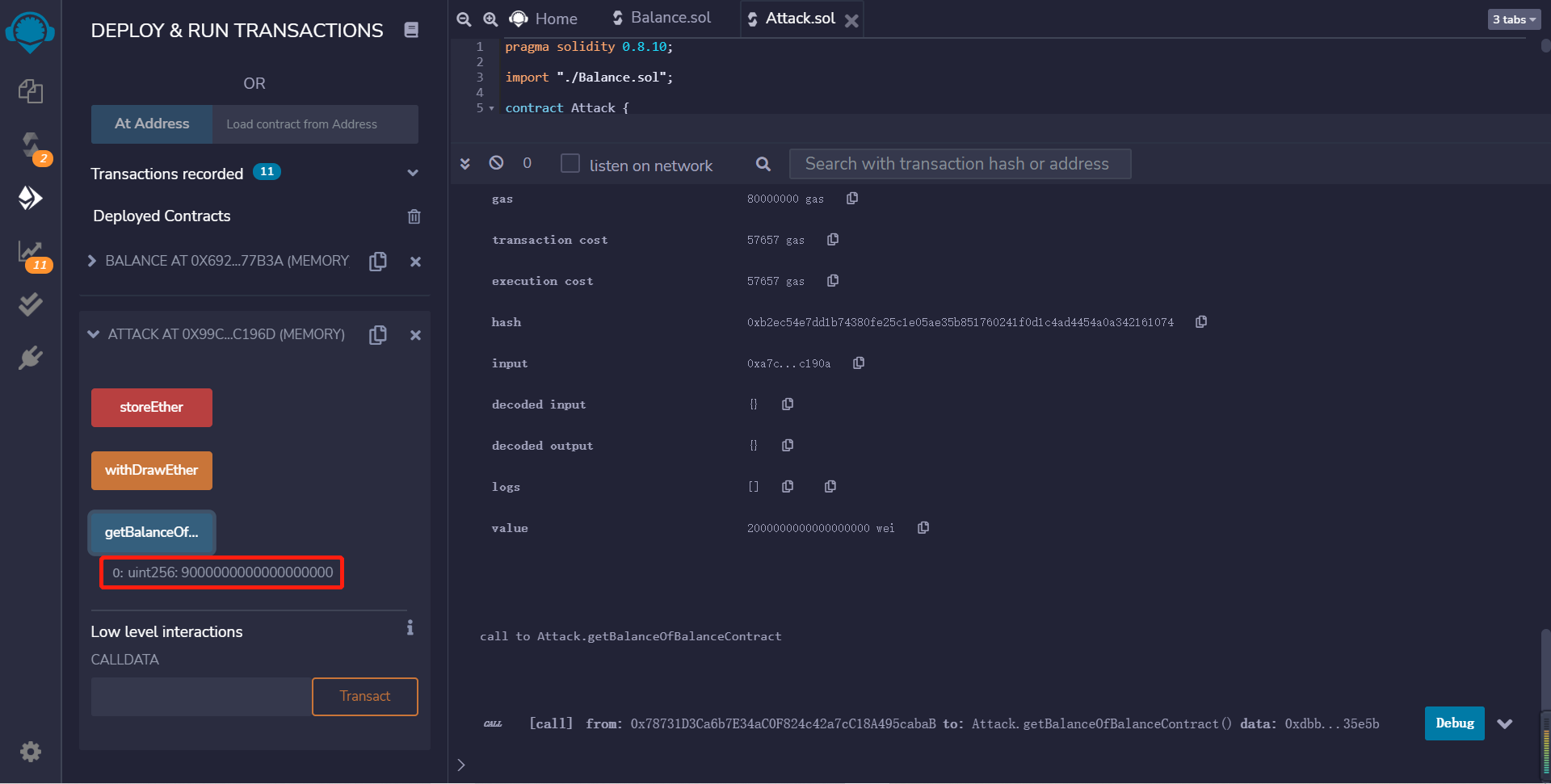


然后攻击者存储2 ether到攻击合约，同时攻击合约把收到的2 ether存储到Balance合约。

由攻击者abaB账号调用Attack合约的storeEther方法，指定2 ether。

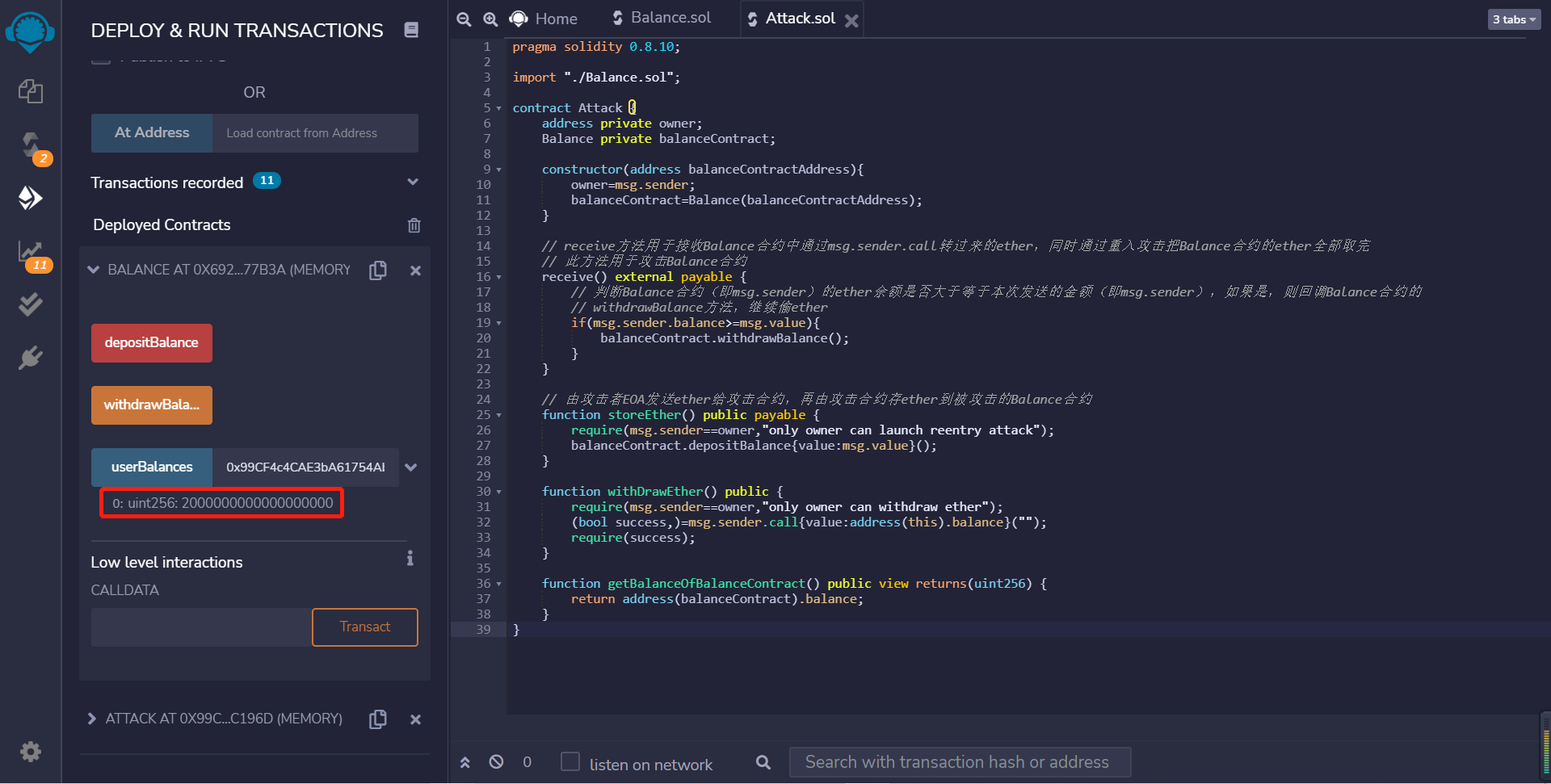


再次查询Balance合约的ether余额，期望是9 ether



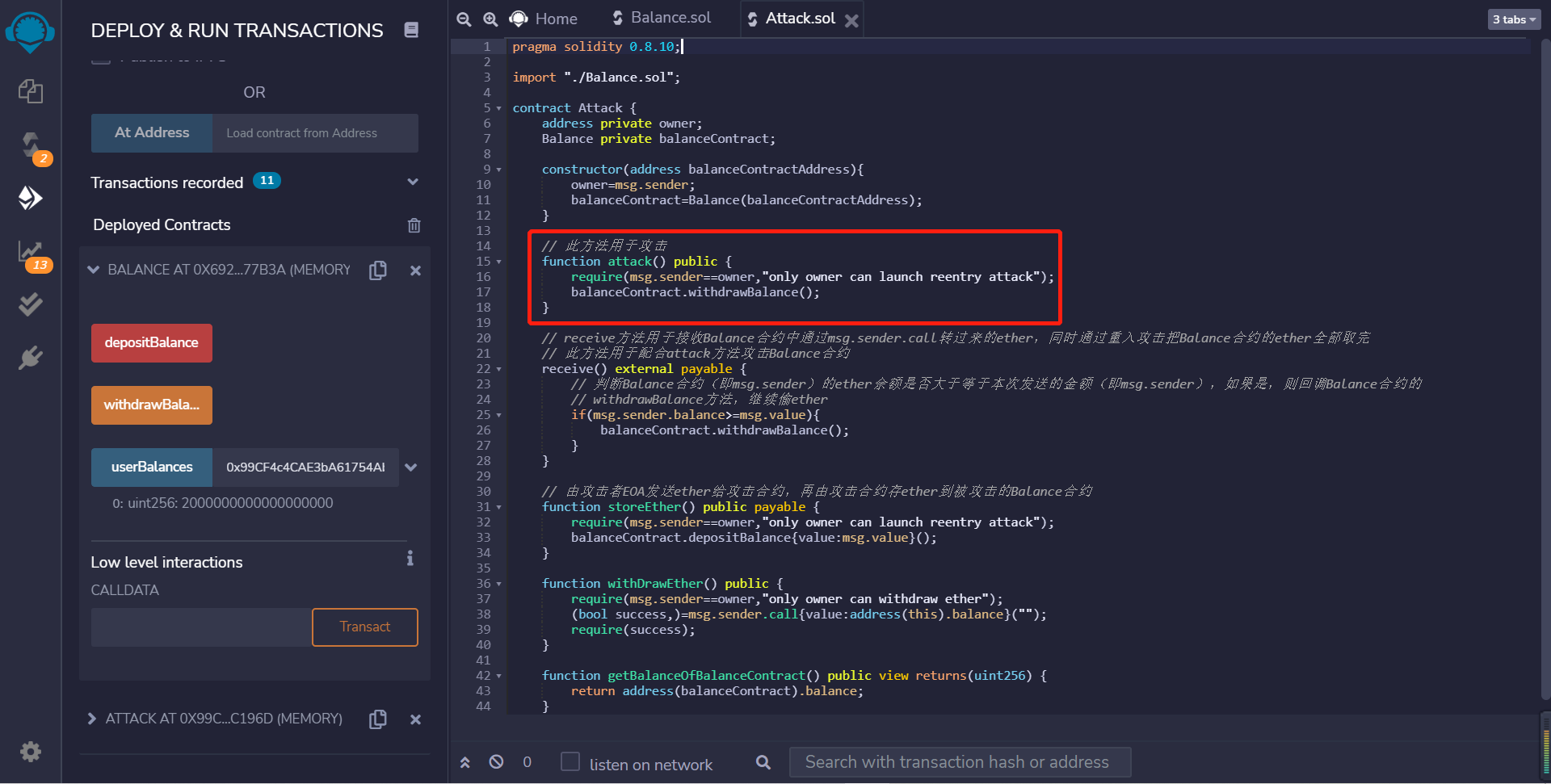
另外，这时攻击者的余额还剩97.9999...

查询Attack合约在Balance合约的balance，期望是2 ether



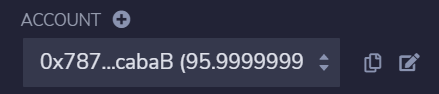
接下来就是发起攻击的时候了，不过突然发现上面犯了个错误，忘记在Attack合约中写一个攻击方法调用Balance合约的withdrawBalance方法发起攻击了，尴尬。。。

在合约中加上这个重要的攻击方法

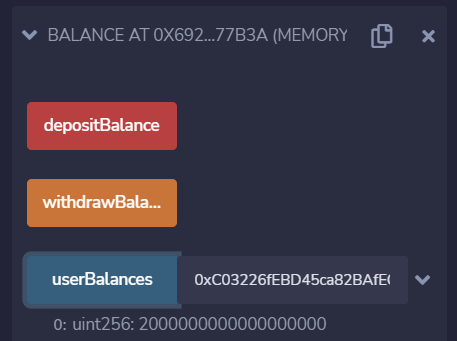


由攻击者abaB重新部署Attack合约，并再次调用storeEther方法，向Balance合约中存入2 ether

这时abaB还剩95.999... ether



同时新部署的Attack合约在Balance合约中的余额为2 ether

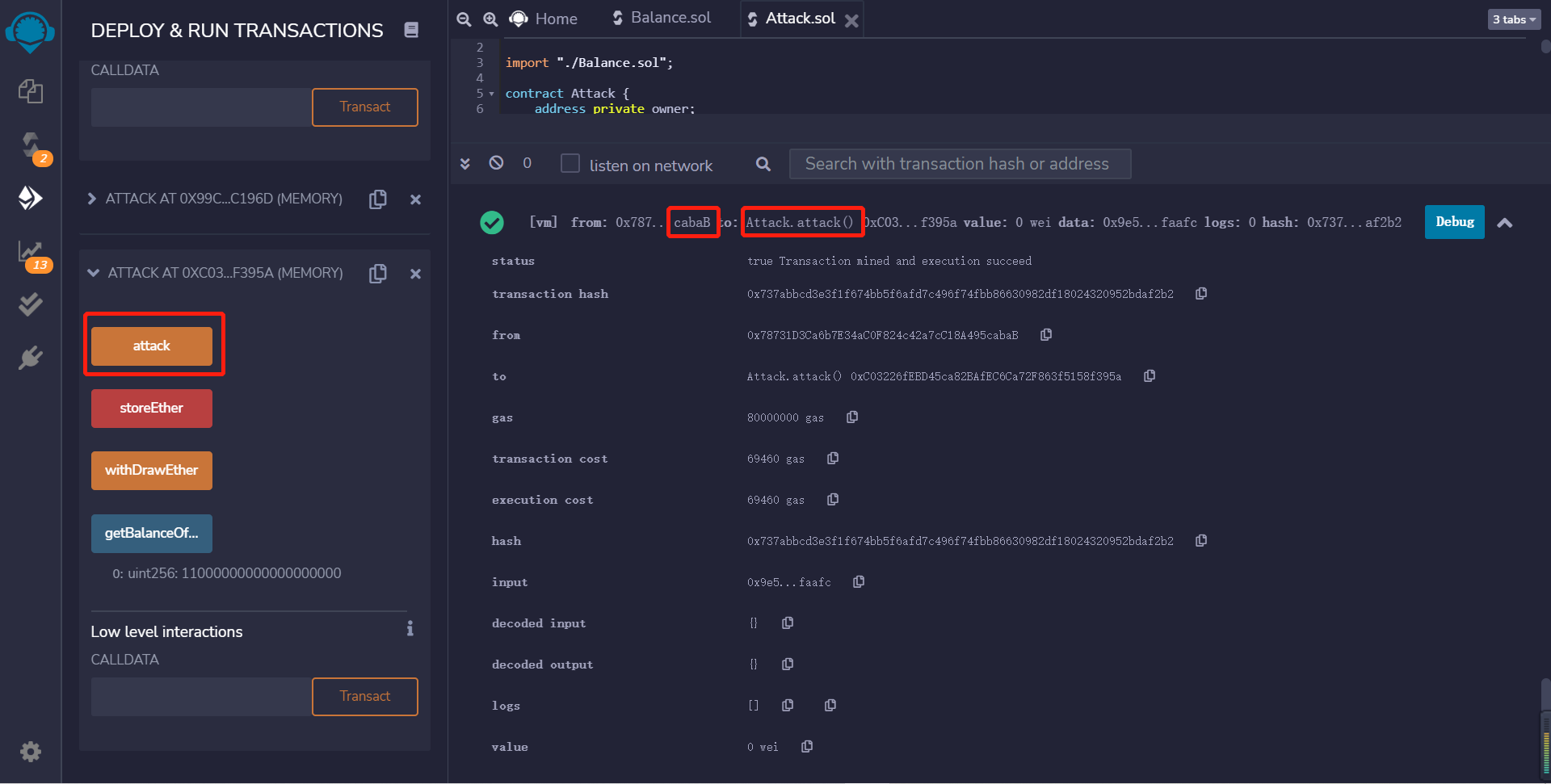


同时，Balance合约的ether余额应为11 ether

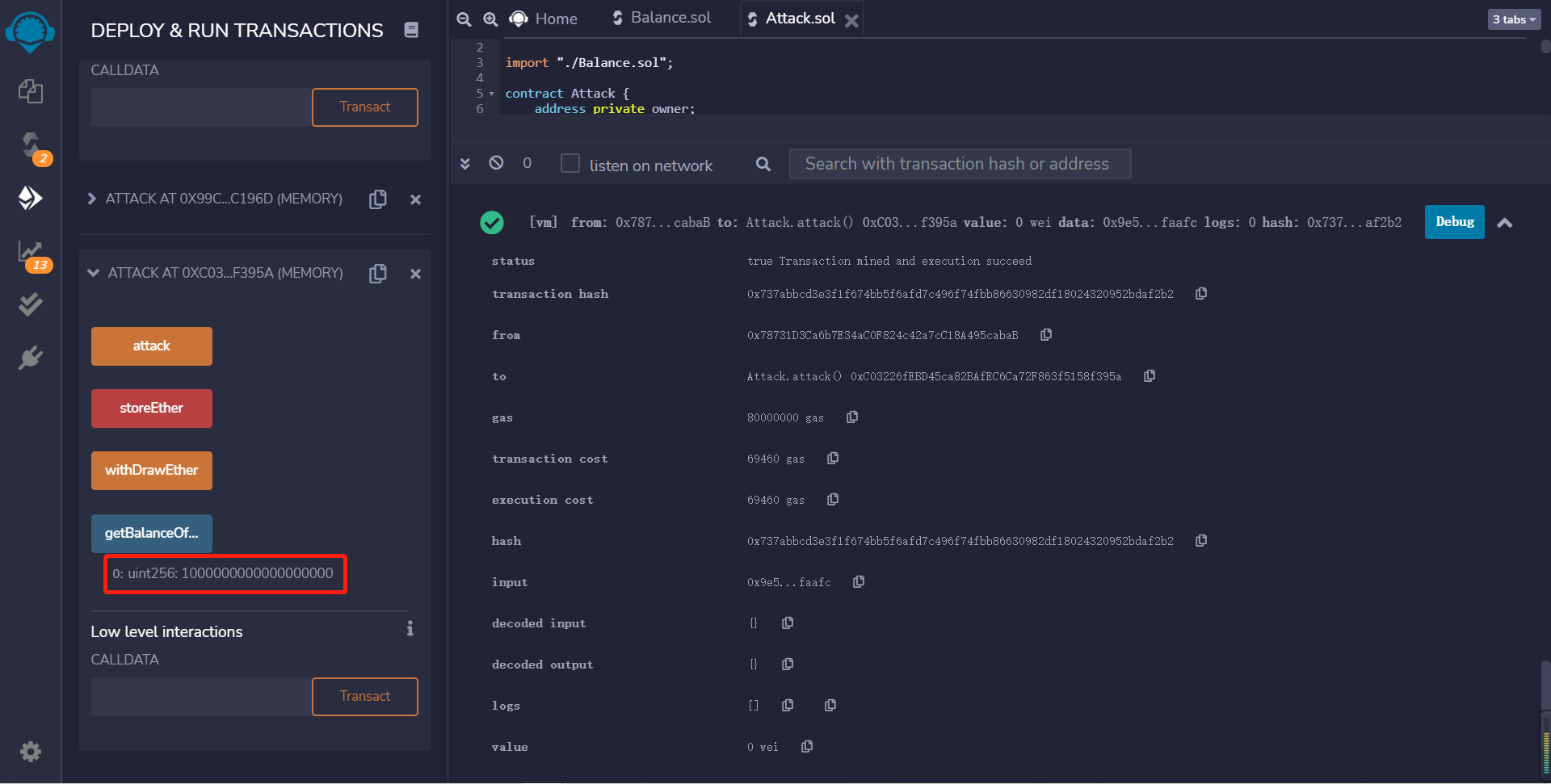
图形用户界面, 网站

描述已自动生成

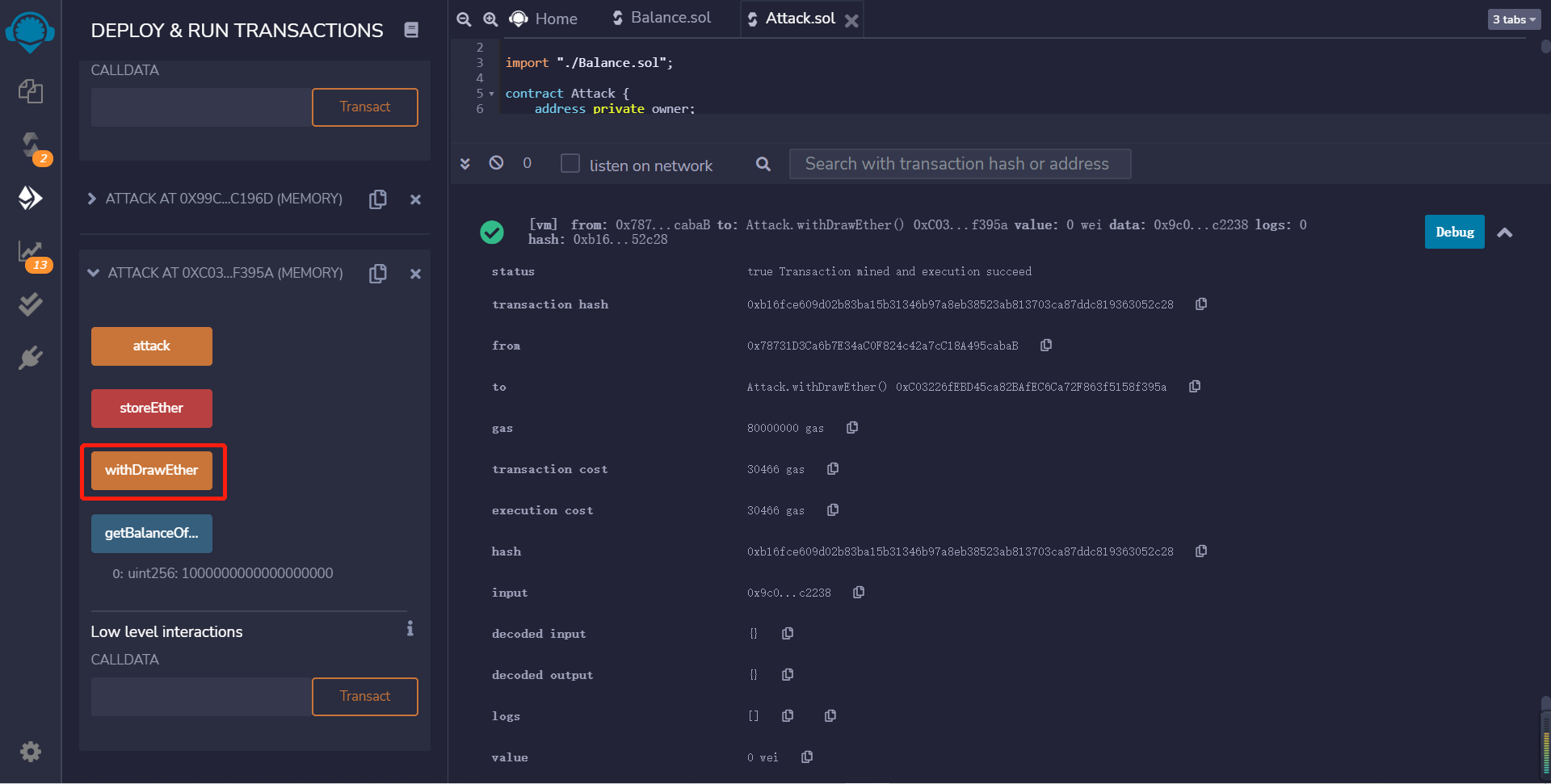
这时候，攻击者abaB调用Attack合约的attack方法发起重入攻击。



期望结果是Balance合约中的11 ether通过5次分批转账，每次转账2 ether，转给了Attack合约，总共转了10 ether，直到转到Balance合约的ether余额小于2 ether位置，不停重复这个转账的过程，Balance合约应该还剩1 ether。查询Balance合约的ether余额，符合期望。



接下来攻击者从Attack合约中取出所有的ether到自己的账上，总共取10 ether，调用withDrawEther。



再次查看攻击者abaB的余额，应该增加10 ether到105.999...

图形用户界面, 文本

描述已自动生成

至此，攻击成功，攻击者原来只有100 ether，现在拥有了105.999 ether，几乎把Balance合约的ether都取光了。

* 防范实操

以上实操了重入攻击的全过程，接下来演示如何防范重入攻击，有几种方法

* + 在方法中先完成内部逻辑（如状态变更），最后再进行外部调用

在以上的例子中，Balance. withdrawBalance方法的执行逻辑是先查询msg.sender的余额，再外部调用转账给msg.sender，再修改余额。可以改为先修改余额，再转账，这样就保证只会转一次，转了之后余额修改为0，再重入的时候也不会真正地转ether了。同时可以加上余额的判断，只有大于0，才执行msg.sender.call。代码如下

文本

描述已自动生成

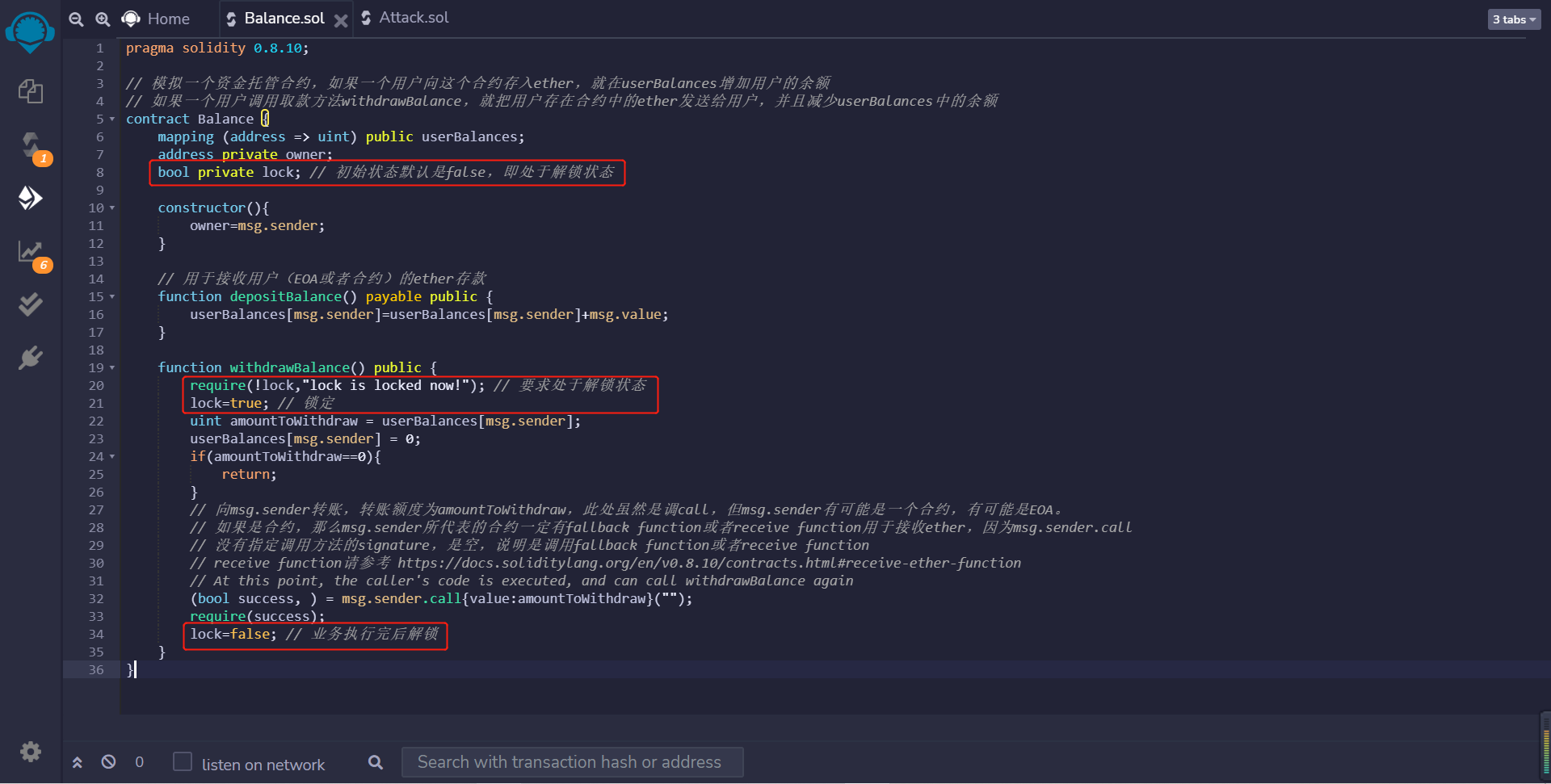
值得注意的是，如果Balance合约有另一个方法调用了withdrawBalance方法，那么这个方法可能同样面临潜在的重入攻击，所以，对于所有调用了外部合约方法的方法，都要将其视为不可信的方法，要格外小心。

所以，重入攻击可以跨多个方法，甚至多个合约，所以任何旨在在单一方法中防范重入攻击的方案都是不够的。

接下来介绍第二个方法防范重入攻击。

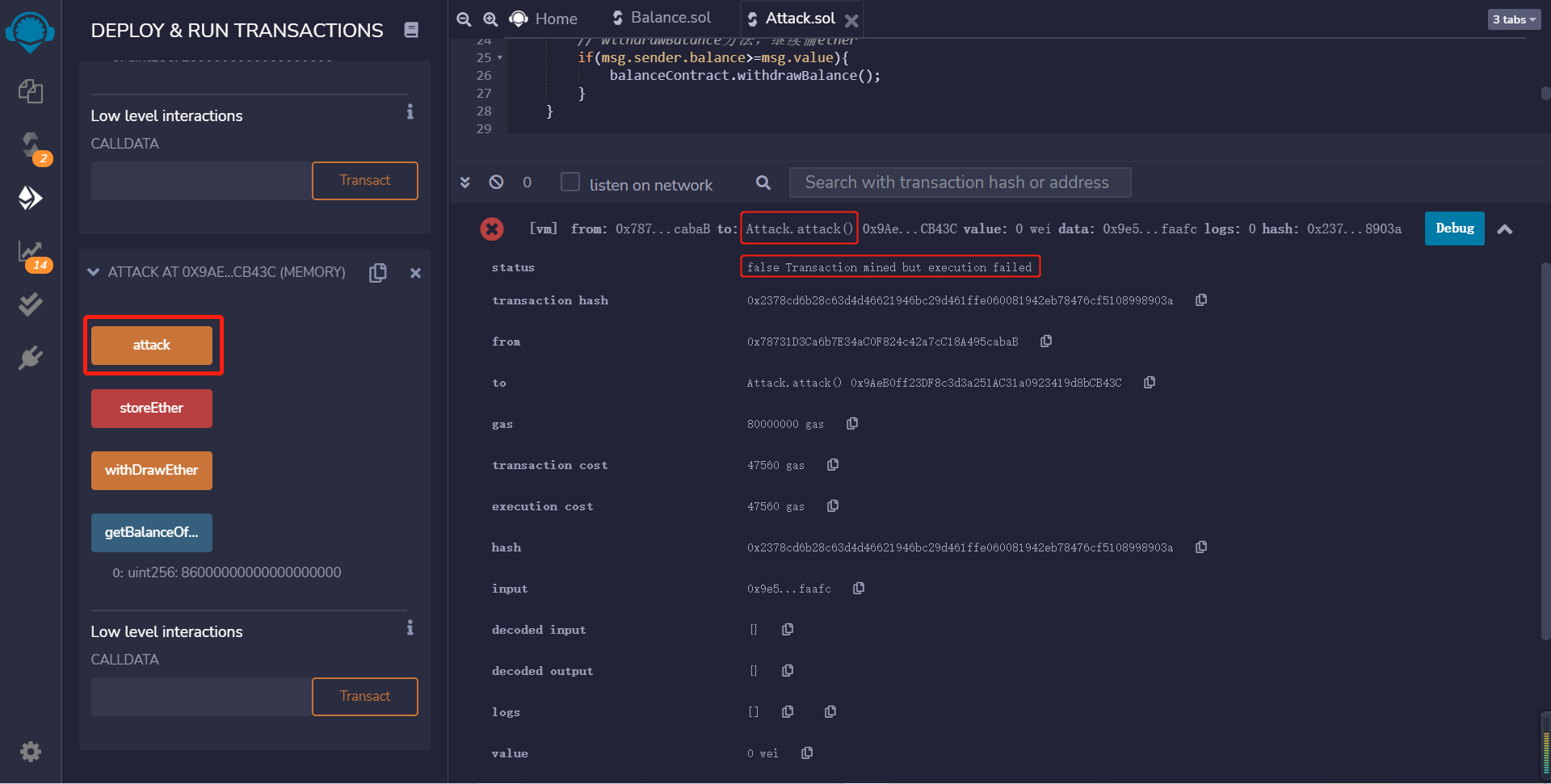
* + 互斥锁

在合约中加上一个全局互斥锁，在需要保护的方法中，要执行方法首先需要互斥锁处于解锁状态，如果处于解锁状态，就加锁，然后执行业务逻辑，我们可以修改一下我们的Balance合约



接下来重新部署Balance合约和Attack合约，再按照之前的部署，先用几个账户往Balance合约里存ether，再由一个攻击者通过Attack合约往Balance里面存入2 ether，然后调用attack方法进行重入攻击，看能不能成功。

就不重复截图了，只截最后调用attack方法攻击的结果。



电脑萤幕的截图

描述已自动生成

由此可见，在Attack合约的receive方法里回调Balance.withdrawBalance时不满足require中的lock解锁条件，所以revert。

可以看出，互斥锁可以防范重入攻击。