fabric-block-archiving的chaincode链码存在隐私泄露漏洞

**合约地址：**

<https://github.com/hyperledger-labs/fabric-block-archiving/blob/b25619767b536d1fed83713b4b24e1800211d132/integration/chaincode/keylevelep/chaincode.go>

上述合约为hyperledger-labs官方仓库合约，12star，10fork

**部分代码：**

211 // setVal sets the value of the KVS key

212 func setVal(stub shim.ChaincodeStubInterface) pb.Response {

213 args := stub.GetArgs()

214 if len(args) != 3 {

215 return shim.Error("setval expects two arguments")

216 }

217 var err error

218 if string(args[1]) == "pub" {

219 err = stub.PutState("pub", args[2])

220 } else if string(args[1]) == "priv" {

221 err = stub.PutPrivateData("col", "priv", args[2])

222 } else {

223 return shim.Error("Unknown key specified")

224 }

225 if err != nil {

226 return shim.Error(err.Error())

227 }

228 return shim.Success([]byte{})

229 }

**漏洞说明：**

chaincode链码使用private data collection（简称PDC）来保护用户的隐私信息，即创建并维护一个指定节点才能访问的分布式数据库col，在数据库中存储不希望其他节点获取的隐私信息。但是该链码由于未规范使用相关API，导致隐私数据被永久公开记录在区块链中，区块链网络中的全体成员均可访问。

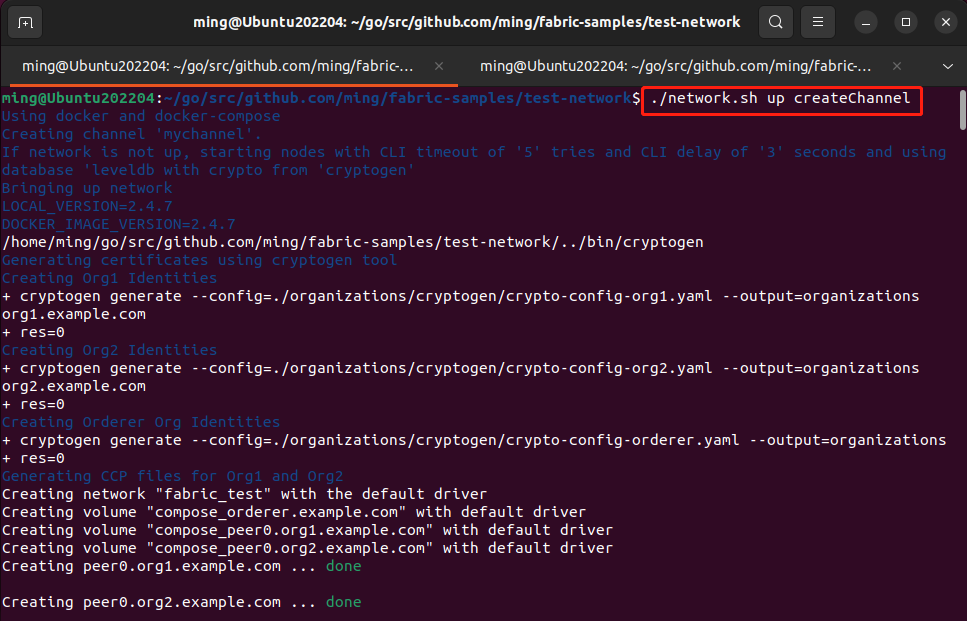
chaincode链码的setVal()函数在213行从参数中获取需要被存入隐私数据库的数据，并在221行调用PutPrivateData()API将该数据args[2]存入隐私数据库col。当本次调用完成后，调用参数也会作为交易的一部分被记录在区块中，即本应仅指定成员才能访问的隐私数据args[2]，变得公开所有成员可见。这显然违背了隐私保护机制的设计初衷。

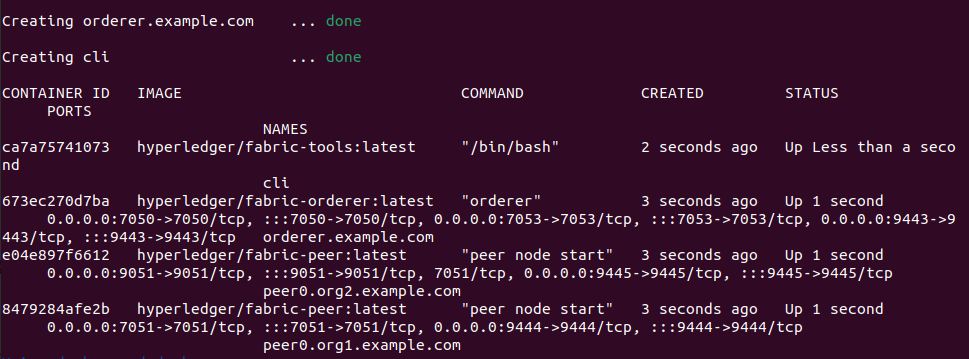
针对该链码的攻击演示如下：

**环境部署：**

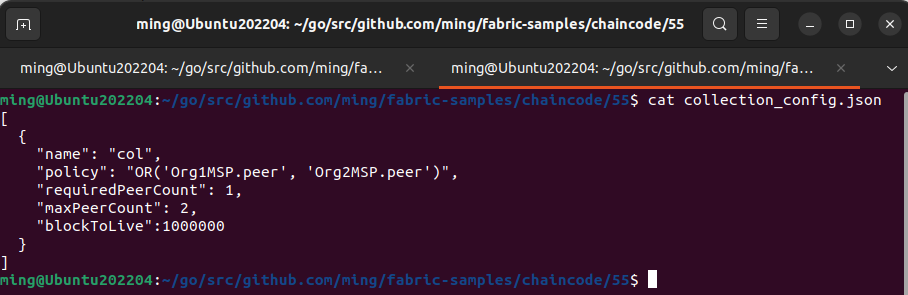
Hyperledger Fabric2.4.7

官方提供的测试网络，网络配置：一个通道内包含三个节点，一个属于Org1的peer节点，一个属于Org2的peer节点，还有一个提供订阅服务的order节点。



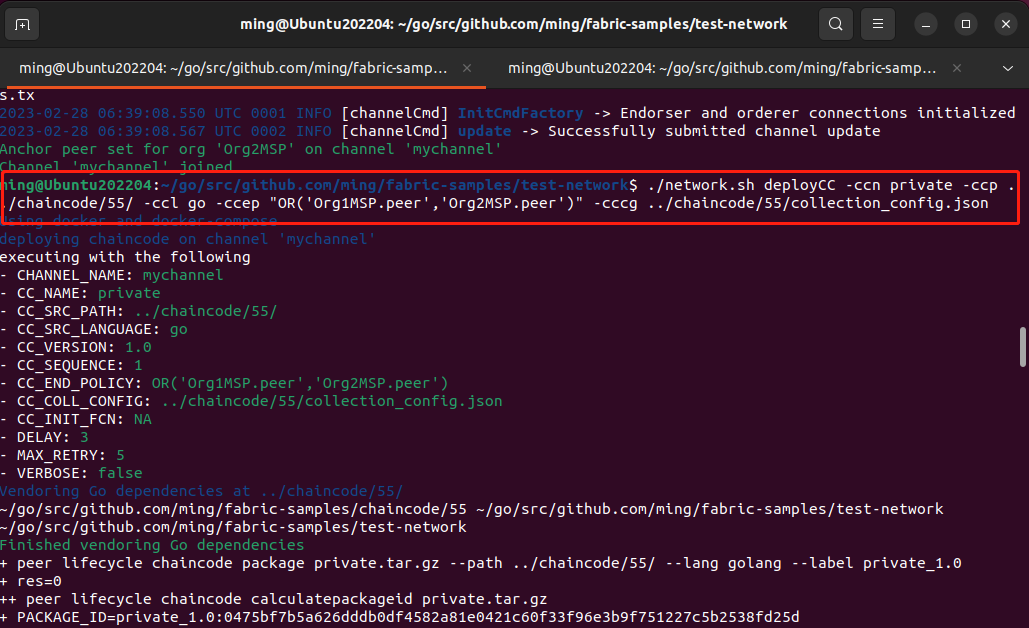


PDC配置为，Org1MSP和Org2MSP的节点均有权限访问数据库col。

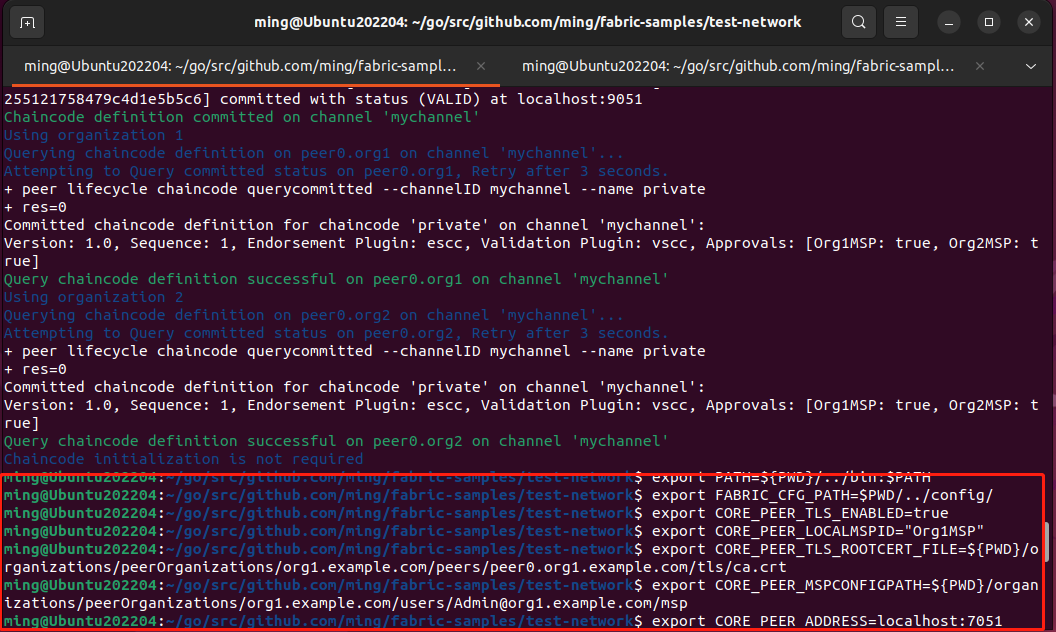


**漏洞利用：**

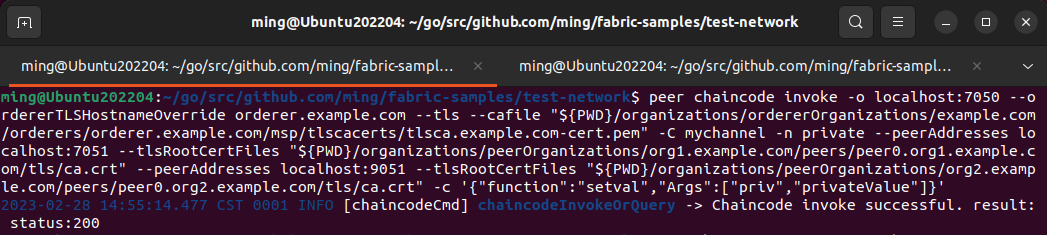
1. 部署智能合约private，命令如图



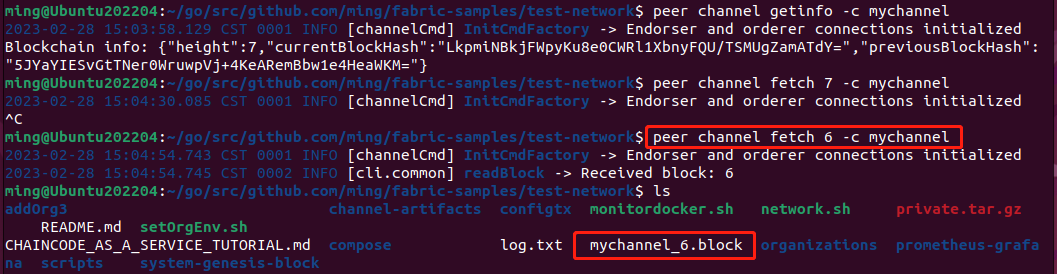
1. 如图，链码已成功部署，设置环境变量，由peer0.org1.example.com节点完成后续操作

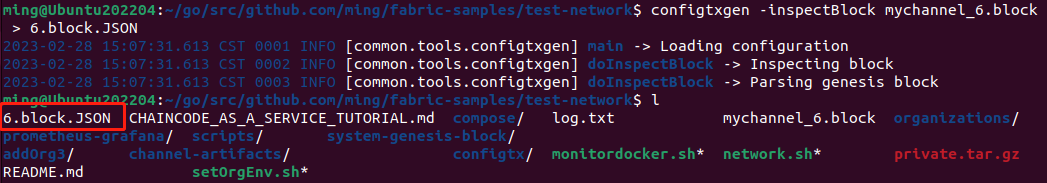


1. 调用setval函数并传入参数priv，privateValue，将privateValue存入隐私数据库col，privateValue应且仅应由Org1MSP和Org2MSP的成员通过访问col获得。



1. 网络中的任何成员均可以作为攻击者，通过查询可知上述链码执行后的相关交易被存放在序号为6的区块中，我们下载该区块并将其解析为JSON文件。



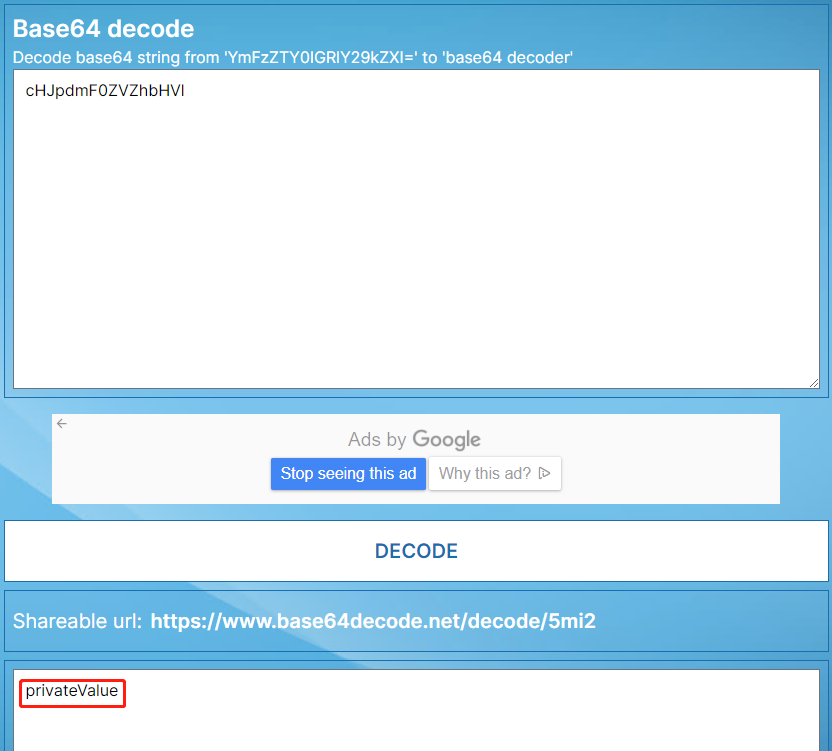


1. 攻击者在JSON文件中找到交易的参数字段“args”，其中第三个参数是被泄露的隐私数据，



1. 利用：攻击者将编码后的参数进行解码，即可获取隐私数据privateValue。

https://www.base64decode.net/



**攻击结果：**

本应仅由PDC成员访问的隐私数据被明文记录在区块中，网络中的任何成员无论是否得到授权均可获取到该隐私数据。

**防御措施：**

如果调用链码的参数中可能会包含隐私数据，则这些参数应当编码后通过调用getTransient()获取，从而避免参数被明文记录在区块中。