USDx Protocol

智能合约安全审计报告

2019-07-17

概要

在本报告中，我们对 USDx Protocol 项目的智能合约代码进行安全审计。我们的任务是发现和指出项目里智能合约代码中的安全问题。

声明

慢雾仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，慢雾无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向慢雾提供的文件和资料(简称“已提供资料”)。慢雾假设：已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，慢雾对由此而导致的损失和不利影响不承担任何责任。

总结

在本报告中，我们对 USDx Protocol 项目的智能合约代码进行安全审计。审计没有发现严重、高危的问题，发现了一些**中危、低危程度的安全问题，**经双方沟通反馈，问题均已修复。

项目概述

项目描述

我们审计了 USDx Protocol 的智能合约代码，如下是相关的文件信息:

项目地址：https://github.com/dforce-network/USDx Protocol

审计初始 commit ： fa7f72917ec1be8a20c291cbc50fc20137fccf4a(v0.4)

最新修复 commit ： 07a53474c796906704888d97076b881487ac3bdb(v0.7)

项目结构

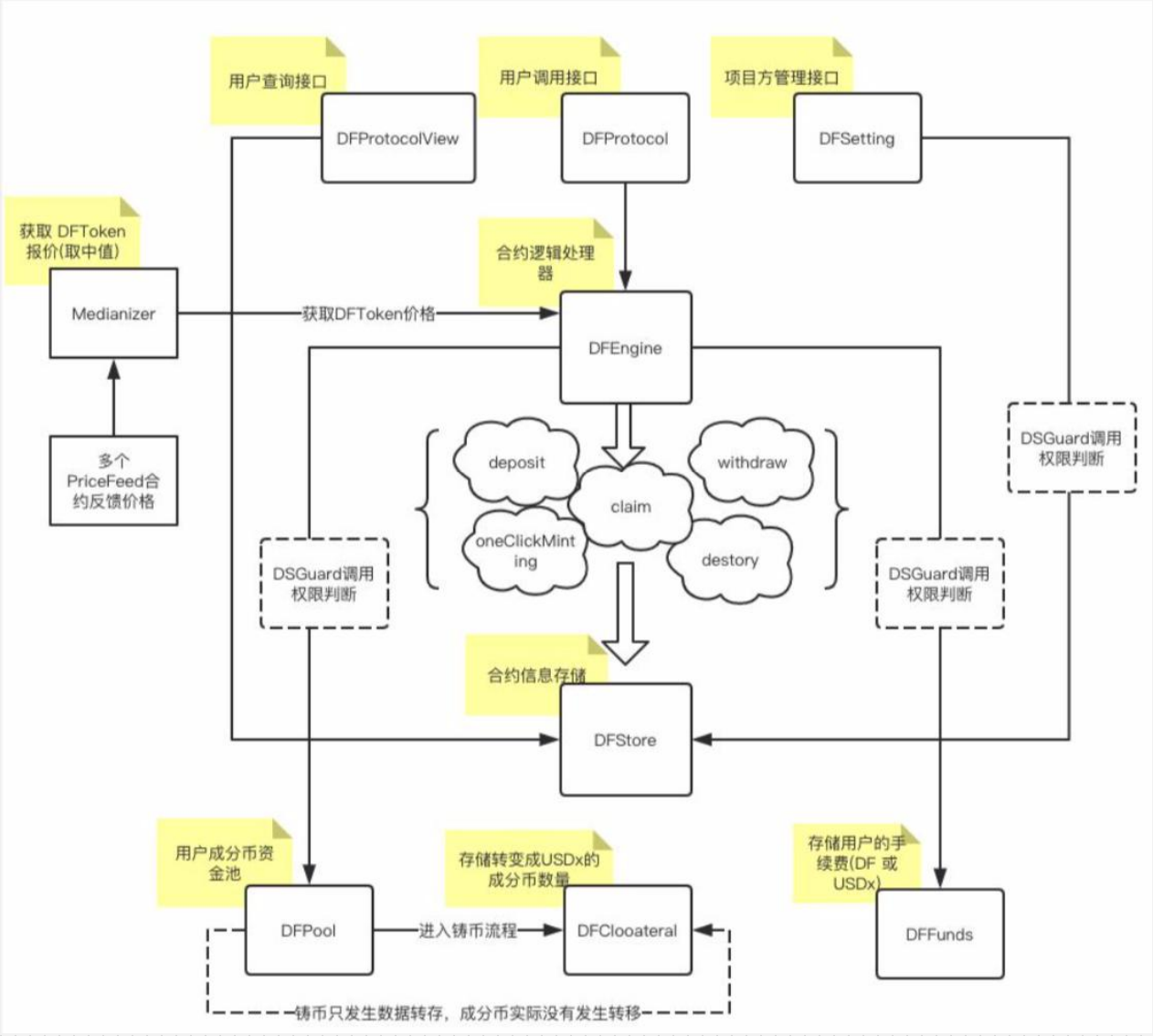
```
./contracts
├── converter
├── helpers
|   └── Migrations.sol ├── oracle
├── DFEngine.sol
├── DFProtocol.sol ├── DFProtocolView.sol ├── DFSetting.sol
└── interfaces
├── IDFEngine.sol └── IDFProtocol.sol
|   ├── Medianizer.sol
|   ├── PriceFeed.sol
|   └── interfaces
|   └── IMedianizer.sol ├── storage
|   ├── DFCollateral.sol |   ├── DFFunds.sol
|   ├── DFPool.sol
|   ├── DFStore.sol
|   └── interfaces
|   ├── IDFCollateral.sol
|   ├── IDFFunds.sol
|   ├── IDFPool.sol
|   └── IDFStore.sol ├── token
├── update
|   └── DFUpgrader.sol └── utility
├── DSToken.sol
├── DSWrappedToken.sol └── interfaces
```

├── IDSToken.sol
├── IDSWrappedToken.sol
└── IERC20Token.sol
├── DSAuth.sol
├── DSGuard.sol
├── DSMath.sol
├── DSNote.sol
├── DSThing.sol
├── DSValue.sol
└── Utils.so

合约架构

合约使用 dapphub/dappsys 框架进行开发，使用 DSGurad 合约对合约之间的互相调用进行权限管理。

总体分为逻辑层、用户接口层、管理层及存储层。合约总体架构如下图所示



审计方法

我们的智能合约安全审计流程包含两个步骤:

- 使用开源或内部自动化分析的工具对合约代码中常见的安全漏洞进行扫描和测试。
- 人工审计代码的安全问题，通过人工分析合约代码，查找代码中潜在的安全问题。

如下是合约代码审计过程中我们会重点审查的常见漏洞列表:

- 重入攻击
- 重放攻击
- 重排攻击
- 数据存储问题
- 短地址攻击
- 拒绝服务攻击
- 交易顺序依赖
- 条件竞争攻击
- 权限控制攻击
- 整数上溢/下溢攻击
- 时间戳依赖攻击
- Gas 使用，Gas 限制和循环
- 冗余的回调函数
- 不安全的接口使用
- 函数状态变量的显式可见性
- 业务逻辑缺陷
- 未声明的存储指针
- 算术精度误差
- tx.origin 身份验证
- 假充值漏洞
- Event 事件安全
- 编译器版本问题
- call 调用安全

审计结果

严重漏洞

严重漏洞会对智能合约的安全造成重大影响，强烈建议修复严重漏洞。

经过审计该项目未发现严重漏洞。

高危漏洞

高危漏洞会影响智能合约的正常运行，强烈建议修复高危漏洞。

经过审计该项目未发现高危漏洞。

中危漏洞

中危漏洞会影响智能合约的运行，建议修复中危漏洞。

权限过大问题

1、storage/DFFunds.sol、storage/DFPool.sol 中 transferOut 函数 auth 权限设置过大，如果 owner 账号被攻击者控制，攻击者可以利用 deposit 函数把用户账户上所有的成分币转入 DFPool 中，再利用 DFPool 的 transferOut 函数把平台上所有的手续费及成分币盗走。

truffle test 代码证明：

```
it("黑客转移所有代币",async()=>{
await DFPool_contract.transferOut(USDC_contract.address,accounts[2],130*10**USDC_decimal); await
DFPool_contract.transferOut(PAX_contract.address,accounts[2],30*10**PAX_decimal); await
DFPool_contract.transferOut(TUSD_contract.address,accounts[2],30*10**TUSD_decimal); await
DFPool_contract.transferOut(DAI_contract.address,accounts[2],10*10**DAI_decimal);
```

```
}); it("黑客转移代币后数据没有变化",async()=>{
let USDC_balance = await xUSDC.balanceOf.call(DFPool_contract.address);
let PAX_balance = await xPAX.balanceOf.call(DFPool_contract.address);
let TUSD_balance = await xTUSD.balanceOf.call(DFPool_contract.address);
let DAI_balance = await xDAI.balanceOf.call(DFPool_contract.address);
assert.equal(USDC_balance.toString(16),(100*10**USDx_decimal).toString(16),"USDC 转移失败");
assert.equal(PAX_balance.toString(16),'0',"PAX 转移失败"); assert.equal(TUSD_balance.toString(16),'0',"TUSD 转移失败
"); assert.equal(DAI_balance.toString(16),'0',"DAI 转移失败");
let USDCToken_balance = await DFStore_contract.getTokenBalance.call(xUSDC.address);
let PAXToken_balance = await DFStore_contract.getTokenBalance.call(xPAX.address);
let TUSDToken_balance = await DFStore_contract.getTokenBalance.call(xTUSD.address);
let DAIToken_balance = await DFStore_contract.getTokenBalance.call(xDAI.address);
assert.equal(USDCToken_balance.toString(16),(100*10**USDx_decimal).toString(16),"TUSD 转移失败");
assert.equal(PAXToken_balance.toString(16),'0',"PAX 转移失败");
assert.equal(TUSDToken_balance.toString(16),'0',"TUSD 转移失败");
assert.equal(DAIToken_balance.toString(16),'0',"DAI 转移失败");
});
```

修复情况：v0.7 已修复，新增 disableOwnership 函数，可在项目稳定运行时销毁 owner 权限，消除该安全风险。

低危漏洞

低危漏洞可能会影响未来版本代码中智能合约的操作，建议项目方自行评估和考虑这些问题是否需要修复。

代码判断缺失问题

1、token/DSToken.sol line:32

setOwner(address owner_) 函数里，建议判断下 owner_ != address(0)，防止操作失误导致权限丢失。

```
function setOwner(address owner_)
        public
```

```
        onlyOwner
    {
        owner = owner_;
        emit LogSetOwner(owner);
    }
```

修复情况：v0.6 已修复，在最新代码中未发现问题。

2、token/DSToken.sol　line:40

setAuthority(address authority_) 函数里，建议判断下 authority_ != address(0)，避免设置错误的 authority 地址。

```
function setAuthority(address authority_)
        public
        onlyOwner
    {
        authority = authority_;
        emit LogSetAuthority(address(authority));
    }
```

修复情况：经与项目方协商沟通后，此问题不作修复。

代码冗余问题

1、converter/DFEngine.sol　　line:25 TokenType 枚举类型，代码冗余，

converter/DFProcotolView.sol　line:13 ProcessType 枚举类型，代码冗余　line:20 TokenType 枚举类型，代码冗余。

DFEngine.sol

```
contract DFEngine is DSMath, DSAuth {
    IDFStore public dfStore;
    IDFPool public dfPool;
    IDSToken public usdxToken;
    address public dfCol;
```

```
    address public dfFunds;

    enum ProcessType {
        CT_DEPOSIT,
        CT_DESTROY,
        CT_CLAIM,
        CT_WITHDRAW
    }

    //SlowMist// 此处代码冗余

    enum TokenType {
        TT_DF,
        TT_USDX
    }
```

DFProcotolView.sol

```
pragma solidity ^0.5.2;

import '../token/interfaces/IDSWrappedToken.sol';
import '../storage/interfaces/IDFStore.sol';
import '../oracle/interfaces/IMedianizer.sol';
import "../utility/DSMath.sol";

contract DFProtocolView is DSMath {
    IDFStore public dfStore;
    address public dfCol;
    address public dfFunds;

    //SlowMist// 此处代码冗余

    enum ProcessType {
        CT_DEPOSIT,
        CT_DESTROY,
        CT_CLAIM,
        CT_WITHDRAW
    }

    //SlowMist// 此处代码冗余

    enum TokenType {
        TT_DF,
        TT_USDX
    }
```

修复情况：v0.7 已修复，在最新代码中未发现问题。

事件声明问题

1、converter/DFProtocol.sol 中 Withdraw 事件中 _amount 可能和实际的提现金额(最后一个参数名 _balance)不符，用户可以自己构造任意提现金额，导致用户提现数额超过用户余额的时候，Withdraw 事件中的 _amount 和 _balance 不一致。可能会导致监听此事件的第三方机构的判断错误。

```
function withdraw(address _tokenID, uint _feeTokenIdx, uint _amount) public returns (uint) {
        uint _balance = iDFEngine.withdraw(msg.sender, _tokenID, _feeTokenIdx, _amount);
        emit Withdraw(_tokenID, msg.sender, _amount, _balance);
        return _balance;
```

修复情况 ：v0.7 已修复，在最新代码中未发现问题。

前端 UI 问题

1、testnet.dforce.network UI 上点击解锁成分币授权时，默认 approve 的 value 是 uint(-1)，这个值太大了，存在一定的风险，且用户看到这么大的值可能会误以为是 transfer 的 amount，用户体验不太好，建议每次 deposit 多少就 approve 多少，降低风险。

修复情况：经与项目方协商沟通后，approve 的问题，只涉及到前端修改，后面会根据用户反馈修改。

附录

truffle test 文件

```
//存储合约
const DFStore = artifacts.require("DFStore");
```

```
//设置合约
const DFSetting = artifacts.require("DFSetting");
//主逻辑合约
const DFEngine = artifacts.require("DFEngine");
//价格采集合约
const PriceFeed = artifacts.require("PriceFeed");
//平均价格计算合约
const Medianizer = artifacts.require("Medianizer");
//成分币
const StableCoin = artifacts.require("ERC20");
//USDx 稳定币
const USDx = artifacts.require("DSToken");
//WrapToken
const WrapToken = artifacts.require("DSWrappedToken");
//col 合约
const col = artifacts.require("DFCollateral")
//DFFunds 合约
const DFFunds = artifacts.require("DFFunds")
//DFPool 合约
const DFPool = artifacts.require("DFPool")
//guard 合约
const DSGuard = artifacts.require("DSGuard")
//protocol 合约
const DFProtocol = artifacts.require("DFProtocol")
//protocolView 合约
const ProtocolView = artifacts.require("DFProtocolView")
//有缺陷的成分币
const StableCoinFalse = artifacts.require("ERC20False")
contract('USDx test',async accounts =>{
    before(async()=>{
        amount = '100000000000000000000000000000';// 10000000000*10**18;
        allowance = '0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff';
        initial_amount = '10000000000';
        USDx_decimal = 18;
        USDC_decimal = 6;
        PAX_decimal =  12;
        DAI_decimal =  12;
        TUSD_decimal = 8;
        deposit_amount = 30;
        //成份币余额初始化
        USDC_contract = await StableCoin.new(accounts[1],web3.utils.toBN(USDC_decimal));
        PAX_contract  = await StableCoin.new(accounts[1],web3.utils.toBN(PAX_decimal));
```

```
        DAI_contract    = await StableCoin.new(accounts[1],web3.utils.toBN(DAI_decimal));
        TUSD_contract = await StableCoin.new(accounts[1],web3.utils.toBN(TUSD_decimal));
        //USDx 合约部署和余额初始化
        USDx_contract = await USDx.new(web3.utils.stringToHex("USDx"));
        //DF 合约部署和余额初始化
        DF_contract = await USDx.new(web3.utils.stringToHex("DF"));
        await
DF_contract.mint(accounts[1],web3.utils.toBN((50000*10**USDx_decimal).toString(16)),{from:accounts[0]});
        await
DF_contract.mint(accounts[2],web3.utils.toBN((50000*10**USDx_decimal).toString(16)),{from:accounts[0]});
        //部署 WrapToken
        xDAI = await WrapToken.new(DAI_contract.address,DAI_decimal,web3.utils.stringToHex("xDAI"));
        xPAX = await WrapToken.new(PAX_contract.address,PAX_decimal,web3.utils.stringToHex("xPAX"));
        xUSDC = await WrapToken.new(USDC_contract.address,USDC_decimal,web3.utils.stringToHex("xUSDC"));
        xTUSD = await WrapToken.new(TUSD_contract.address,TUSD_decimal,web3.utils.stringToHex("xTUSD"));
        //部署存储合约
        daiW = web3.utils.toBN(1*10**18);
        paxW = web3.utils.toBN(3*10**18);
        tusdW = web3.utils.toBN(3*10**18);
        usdcW = web3.utils.toBN(3*10**18);
        DFStore_contract = await
DFStore.new([xDAI.address,xPAX.address,xUSDC.address,xTUSD.address],[daiW,paxW,tusdW,usdcW]);
        //部署 col 合约
        col_contract = await col.new()
        //部署 DFFunds 合约
        DFFunds_contract = await DFFunds.new(DF_contract.address);
        //部署 DFPool 合约
        DFPool_contract = await DFPool.new(col_contract.address);
        //部署 Medianizer
        Medianizer_contract = await Medianizer.new();
        //部署 PriceFeed
        PriceFeed_contract = await PriceFeed.new();
        //部署 DFEngine 主逻辑合约
        DFEngine_contract = await
DFEngine.new(USDx_contract.address,DFStore_contract.address,DFPool_contract.address,col_contract.address,DFFun
ds_contract.address);
        //用户将 DF Token 和 USDx token 授权给 Engine
        await DF_contract.approve(DFEngine_contract.address,web3.utils.toBN(amount),{from:accounts[1]});
        await DF_contract.approve(DFEngine_contract.address,web3.utils.toBN(amount),{from:accounts[2]});
        await USDx_contract.approve(DFEngine_contract.address,web3.utils.toBN(amount),{from:accounts[1]});
        await USDx_contract.approve(DFEngine_contract.address,web3.utils.toBN(amount),{from:accounts[2]});
        //部署 Setting 合约
```

```
DFSetting_contract = await DFSetting.new(DFStore_contract.address,{from:accounts[0]});
//成分币授权给合约
USDC_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[1]})
PAX_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[1]})
TUSD_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[1]})
DAI_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[1]})
//授权给 DFEngine
await xDAI.setAuthority(DFEngine_contract.address,{from:accounts[0]});
await xPAX.setAuthority(DFEngine_contract.address,{from:accounts[0]});
await xTUSD.setAuthority(DFEngine_contract.address,{from:accounts[0]});
await xUSDC.setAuthority(DFEngine_contract.address,{from:accounts[0]});
//DFPool 授权 xToken 额度给 Engine
await DFPool_contract.approveToEngine(xDAI.address,DFEngine_contract.address,{from:accounts[0]});
await DFPool_contract.approveToEngine(xUSDC.address,DFEngine_contract.address,{from:accounts[0]});
await DFPool_contract.approveToEngine(xPAX.address,DFEngine_contract.address,{from:accounts[0]});
await DFPool_contract.approveToEngine(xTUSD.address,DFEngine_contract.address,{from:accounts[0]});
//col 授权给 Engine
await col_contract.approveToEngine(xDAI.address,DFEngine_contract.address,{from:accounts[0]});
await col_contract.approveToEngine(xTUSD.address,DFEngine_contract.address,{from:accounts[0]});
await col_contract.approveToEngine(xUSDC.address,DFEngine_contract.address,{from:accounts[0]});
await col_contract.approveToEngine(xPAX.address,DFEngine_contract.address,{from:accounts[0]});
//USDx 授权给 Engine
await USDx_contract.setAuthority(DFEngine_contract.address,{from:accounts[0]});
//部署 Guard
DSGuard_contract = await DSGuard.new({from:accounts[0]});
// guard => Pool
await DFPool_contract.setAuthority(DSGuard_contract.address);
// guard => Store
await DFStore_contract.setAuthority(DSGuard_contract.address);
// guard => collateral
await col_contract.setAuthority(DSGuard_contract.address);
// guard => Funds
await DFFunds_contract.setAuthority(DSGuard_contract.address);
// guard => Engine
await DFEngine_contract.setAuthority(DSGuard_contract.address)
// Store permit Engine
await DSGuard_contract.permitx(DFEngine_contract.address,DFStore_contract.address,{from:accounts[0]});
// Store permit Setting
await DSGuard_contract.permitx(DFSetting_contract.address,DFStore_contract.address,{from:accounts[0]});
// Pool permit Engine
await DSGuard_contract.permitx(DFEngine_contract.address,DFPool_contract.address,{from:accounts[0]});
// collateral permit Engine
```

```
        await DSGuard_contract.permitx(DFEngine_contract.address,col_contract.address,{from:accounts[0]});
        // Funds to Engine
        await DSGuard_contract.permitx(DFEngine_contract.address,DFFunds_contract.address,{from:accounts[0]});
        //部署 Protocol
        DFProtocol_contract = await DFProtocol.new();
        // Engine permit Protocol
        await DSGuard_contract.permitx(DFProtocol_contract.address,DFEngine_contract.address);
        await DFProtocol_contract.requestImplChange(DFEngine_contract.address)
        await DFProtocol_contract.confirmImplChange();
        // set commission rate deposit = 0
        await DFSetting_contract.setCommissionRate(0,0,{from:accounts[0]});
        // set commission rate destory = 0.001
        await DFSetting_contract.setCommissionRate(1,10,{from:accounts[0]});
        // set commission token == DF
        await DFSetting_contract.setCommissionToken(0,DF_contract.address,{from:accounts[0]});
        // set destory usdx threshold == 0.01
        th = web3.utils.toBN(0.01 * 10 **18);
        await DFSetting_contract.setDestroyThreshold(th);
        // set DF medianizer
        await
DFSetting_contract.setCommissionMedian(DF_contract.address,Medianizer_contract.address,{from:accounts[0]});
        // Medianizer
        await Medianizer_contract.set(PriceFeed_contract.address);
        // PriceFeed
        price = web3.utils.toBN(2*10**18);
        await PriceFeed_contract.post(price,2058870102,Medianizer_contract.address);
        //部署 ProtocolView 合约
        ProtocolView_contract = await ProtocolView.new(DFStore_contract.address,col_contract.address);


    });
    it("成分币部署正确",async() =>{
        let USDC_balance =  await USDC_contract.balanceOf.call(accounts[1]);
        let PAX_balance  =  await PAX_contract.balanceOf.call(accounts[1]);
        let DAI_balance  =  await DAI_contract.balanceOf.call(accounts[1]);
        let TUSD_balance =  await TUSD_contract.balanceOf.call(accounts[1]);
        assert.equal(USDC_balance.toString(),10000000000*10**USDC_decimal,"USDC balance init balance is not
correct");
        assert.equal(PAX_balance.toString(),10000000000*10**PAX_decimal,"PAX balance init balance is not correct");
        assert.equal(DAI_balance.toString(),10000000000*10**DAI_decimal,"DAI balance init balance is not correct");
        assert.equal(TUSD_balance.toString(),10000000000*10**TUSD_decimal,"TUSD balance init balance is not
correct");
    });
```

```
it("USDx  合约部署正确",async()=>{
    let isOwner = await USDx_contract.isOwner.call(accounts[0]);
    assert.equal(isOwner.toString(),"true","Owner is not set correctly");
    let owner_balance = await USDx_contract.balanceOf.call(accounts[0]);
    assert.equal(owner_balance.toString(),0,"owner balance not set correctly");
});
it("DF  合约部署正确",async()=>{
    let isOwner = await DF_contract.isOwner.call(accounts[0]);
    assert.equal(isOwner.toString(),"true","Owner is not set correctly");
    let owner_balance = await DF_contract.balanceOf.call(accounts[1]);
    let DF_allowance = await DF_contract.allowance.call(accounts[1],DFEngine_contract.address);
    assert.equal(owner_balance.toString(16),(50000*10**18).toString(16),"owner balance not set correctly");
    assert.equal(DF_allowance.toString(),amount,"DF allowance to Engine incorrect");
});
it("x 成分币合约部署正确",async()=>{
    let xDAI_addr = await xDAI.getSrcERC20.call();
    let xUSDC_addr = await xUSDC.getSrcERC20.call();
    let xTUSD_addr = await xTUSD.getSrcERC20.call();
    let xPAX_addr = await xPAX.getSrcERC20.call();
    let xPAX_decimal = await xPAX.srcDecimals.call();
    let xUSDC_decimal = await xUSDC.srcDecimals.call();
    let xDAI_decimal = await xDAI.srcDecimals.call();
    let xTUSD_decimal = await xTUSD.srcDecimals.call();
    assert.equal(xDAI_addr, DAI_contract.address,"xDAI address is not correct");
    assert.equal(xTUSD_addr,TUSD_contract.address,"xTUSD address is not correct");
    assert.equal(xUSDC_addr,USDC_contract.address,"xUSDC address is not correct");
    assert.equal(xPAX_addr,PAX_contract.address,"xPAX address is not correct");
    assert.equal(xPAX_decimal,PAX_decimal,"xPAX decimal is not correct");
    assert.equal(xTUSD_decimal,TUSD_decimal,"xTUSD decimal is not correct");
    assert.equal(xDAI_decimal,DAI_decimal,"xDAI decimal is not correct");
    assert.equal(xUSDC_decimal,USDC_decimal,"xUSDC decimal is not correct");

});
it("存储合约部署正确",async()=>{
    let SectionData = await DFStore_contract.getSectionData.call(0);
    let minted = SectionData['0'];
    let burned = SectionData['1'];
    let backupIdx = SectionData['2'];
    let collDs = SectionData['3'];
    let cw = SectionData['4'];
    assert.equal(minted.toString(),0,"mint incorrect");
    assert.equal(burned.toString(),0,"burned incorrect");
```

```
            assert.equal(backupIdx.toString(),0,"backupIdx incorrect");
            for(i=0;i<collDs.length;i++){
                assert.equal(collDs[i],[xDAI.address,xPAX.address,xUSDC.address,xTUSD.address][i],"collDs is not
correct");
            }
            for(i=0;i<cw.length;i++){
                assert.equal(cw[i].toString(),[daiW,paxW,tusdW,usdcW][i],"cw incorrect");
            }

    });
    it("成分币授权给 Pool 正确",async ()=>{
        let allow_USDC = await USDC_contract.allowance.call(accounts[1],DFPool_contract.address);
        let allow_TUSD = await TUSD_contract.allowance.call(accounts[1],DFPool_contract.address);
        let allow_PAX = await PAX_contract.allowance.call(accounts[1],DFPool_contract.address);
        let allow_DAI = await DAI_contract.allowance.call(accounts[1],DFPool_contract.address);
        assert.equal('0x'+allow_USDC.toString(16),allowance,"USDC approve to Engine incorrect")
        assert.equal('0x'+allow_TUSD.toString(16),allowance,"TUSD approve to Engine incorrect")
        assert.equal('0x'+allow_PAX.toString(16),allowance,"PAX approve to Engine incorrect")
        assert.equal('0x'+allow_DAI.toString(16),allowance,"DAI approve to Engine incorrect")
    });
    it("Pool ApprovetoEngine 正确",async()=>{
        let allow_USDC = await xUSDC.allowance.call(DFPool_contract.address,DFEngine_contract.address);
        assert.equal('0x'+allow_USDC.toString(16),allowance,"Pool USDC to Engine incorrect");
    })
    it("部署 col 合约正确",async()=>{
        let col_owner = await col_contract.owner.call();
        assert.equal(col_owner,accounts[0],"col owner incorrect")
    })
    it("充值 30USDC 成功",async()=>{
        await
DFProtocol_contract.deposit(USDC_contract.address,web3.utils.toBN(0),web3.utils.toBN(deposit_amount*10**USDC_d
ecimal),{from:accounts[1]})
        let USDC_balance = await DFStore_contract.getDepositorBalance.call(accounts[1],xUSDC.address);
        assert.equal(USDC_balance.toString(16),(30*10**USDx_decimal).toString(16),"USDC deposit faild");
    });
    it("充值 30PAX 成功",async()=>{
        await
DFProtocol_contract.deposit(PAX_contract.address,web3.utils.toBN(0),web3.utils.toBN(deposit_amount*10**PAX_deci
mal),{from:accounts[1]})
        let PAX_balance = await DFStore_contract.getDepositorBalance.call(accounts[1],xPAX.address);
        assert.equal(PAX_balance.toString(16),(30*10**USDx_decimal).toString(16),"PAX deposit faild");
    });
```

```
    it("充值 30TUSD 成功",async()=>{
        await DFProtocol_contract.deposit(TUSD_contract.address,web3.utils.toBN(0),web3.utils.toBN(deposit_amount*10**TUSD_decimal),{from:accounts[1]})
        let TUSD_balance = await DFStore_contract.getDepositorBalance.call(accounts[1],xTUSD.address);
        assert.equal(TUSD_balance.toString(16),(30*10**USDx_decimal).toString(16),"TUSD deposit faild");
    });
    it("充值 30DAI 成功",async()=>{
        await DFProtocol_contract.deposit(DAI_contract.address,web3.utils.toBN(0),web3.utils.toBN(deposit_amount*10**DAI_decimal),{from:accounts[1]})
        let DAI_balance = await DFStore_contract.getDepositorBalance.call(accounts[1],xDAI.address);
        assert.equal(DAI_balance.toString(16),(20*10**USDx_decimal).toString(16),"DAI deposit faild");
    });
    it("查询第一次铸币后 col 各币种状态正确",async()=>{
        let colBalance = await ProtocolView_contract.getColStatus.call();
        for(let i=0;i<colBalance.length;i++){
            assert.equal(colBalance[i],deposit*amount**[12,12,6,8][i],"col Status is not correct")
        }
    });
    it("查询第一次铸币后获得 USDx 数量正确",async()=>{
        let mintAmount = await USDx_contract.balanceOf.call(accounts[1]);
        assert.equal(mintAmount.toString(),100*10**USDx_decimal,"USDx mint incorrect")
    });
    it("查询第一次铸币后 Token Pool 数量正确",async()=>{
        let DAI_Res = await DFStore_contract.getTokenBalance.call(xDAI.address);
        let USDC_Res = await DFStore_contract.getTokenBalance.call(xUSDC.address);
        let TUSD_Res = await DFStore_contract.getTokenBalance.call(xTUSD.address);
        let PAX_Res = await DFStore_contract.getTokenBalance.call(xPAX.address);
        assert.equal(DAI_Res.toString(),20*10**USDx_decimal,"DAI Pool incorrect");
        assert.equal(USDC_Res.toString(),'0',"DAI Pool incorrect");
        assert.equal(PAX_Res.toString(),'0',"DAI Pool incorrect");
        assert.equal(TUSD_Res.toString(),'0',"DAI Pool incorrect");
    });
    it("查询用户第一次充值后余额正确",async()=>{
        let user_DAIamount = await DFStore_contract.getDepositorBalance.call(accounts[1],xDAI.address);
        assert.equal(user_DAIamount.toString(),20*10**USDx_decimal,"User DAI balance incorrect")
        let user_PAXamount = await DFStore_contract.getDepositorBalance.call(accounts[1],xPAX.address);
        assert.equal(user_PAXamount.toString(),Number(0).toString(),"User TUSD balance incorrect")
        let user_USDCamount = await DFStore_contract.getDepositorBalance.call(accounts[1],xUSDC.address);
        assert.equal(user_USDCamount.toString(),Number(0).toString(),"User USDC balance incorrect")
        let user_TUSDamount = await DFStore_contract.getDepositorBalance.call(accounts[1],xTUSD.address);
```

```
        assert.equal(user_TUSDamount.toString(),Number(0).toString(),"User TUSD balance incorrect")
    });
    it("用户尝试提现第一次充值剩余的 DAI 中的 30 个",async()=>{
        let withdraw_amount = 30 * 10**DAI_decimal;
        await
DFProtocol_contract.withdraw(DAI_contract.address,web3.utils.toBN(0),web3.utils.toBN(withdraw_amount),{from:acco
unts[1]});
        let Pool_amount = await DFStore_contract.getTokenBalance.call(xDAI.address);
        assert.equal(Pool_amount.toString(16),'0',"Pool amount after withdraw incorrect");
    });
    it("用户继续充值 100 个 USDC 成功",async()=>{
        await
DFProtocol_contract.deposit(USDC_contract.address,web3.utils.toBN(0),web3.utils.toBN(100*10**USDC_decimal),{fro
m:accounts[1]})
        let USDC_balance = await xUSDC.balanceOf.call(DFPool_contract.address);
        assert.equal(USDC_balance.toString(16),(100*10**USDx_decimal).toString(16),"Second deposit faild");
    });
    it("黑客转移所有代币",async()=>{
        await DFPool_contract.transferOut(USDC_contract.address,accounts[2],130*10**USDC_decimal);
        await DFPool_contract.transferOut(PAX_contract.address,accounts[2],30*10**PAX_decimal);
        await DFPool_contract.transferOut(TUSD_contract.address,accounts[2],30*10**TUSD_decimal);
        await DFPool_contract.transferOut(DAI_contract.address,accounts[2],10*10**DAI_decimal);
    });
    it("黑客转移代币后数据没有变化",async()=>{
        let USDC_balance = await xUSDC.balanceOf.call(DFPool_contract.address);
        let PAX_balance = await xPAX.balanceOf.call(DFPool_contract.address);
        let TUSD_balance = await xTUSD.balanceOf.call(DFPool_contract.address);
        let DAI_balance = await xDAI.balanceOf.call(DFPool_contract.address);
        assert.equal(USDC_balance.toString(16),(100*10**USDx_decimal).toString(16),"USDC 转移失败");
        assert.equal(PAX_balance.toString(16),'0',"PAX 转移失败");
        assert.equal(TUSD_balance.toString(16),'0',"TUSD 转移失败");
        assert.equal(DAI_balance.toString(16),'0',"DAI 转移失败");
        let USDCToken_balance = await DFStore_contract.getTokenBalance.call(xUSDC.address);
        let PAXToken_balance = await DFStore_contract.getTokenBalance.call(xPAX.address);
        let TUSDToken_balance = await DFStore_contract.getTokenBalance.call(xTUSD.address);
        let DAIToken_balance = await DFStore_contract.getTokenBalance.call(xDAI.address);
        assert.equal(USDCToken_balance.toString(16),(100*10**USDx_decimal).toString(16),"TUSD 转移失败");
        assert.equal(PAXToken_balance.toString(16),'0',"PAX 转移失败");
        assert.equal(TUSDToken_balance.toString(16),'0',"TUSD 转移失败");
        assert.equal(DAIToken_balance.toString(16),'0',"DAI 转移失败");
    });
    it("用户尝试提币,但是失败了",async()=>{
```

```
        try{
            await
DFProtocol_contract.withdraw(USDC_contract.address,web3.utils.toBN(0),web3.utils.toBN(1),{from:accounts[1]});
        }catch(err){
            assert.include(err.message,"transfer balance not enough");
        }
    });
    it("用户 B 授权给 DFPool",async()=>{
        USDC_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[2]})
        PAX_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[2]})
        TUSD_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[2]})
        DAI_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[2]})
        let allow_USDC = await USDC_contract.allowance.call(accounts[2],DFPool_contract.address);
        let allow_PAX = await PAX_contract.allowance.call(accounts[2],DFPool_contract.address);
        let allow_TUSD = await TUSD_contract.allowance.call(accounts[2],DFPool_contract.address);
        let allow_DAI = await DAI_contract.allowance.call(accounts[2],DFPool_contract.address);
        assert.equal('0x'+allow_USDC.toString(16),allowance,"USDC approve to Engine incorrect")
        assert.equal('0x'+allow_PAX.toString(16),allowance,"PAX approve to Engine incorrect")
        assert.equal('0x'+allow_TUSD.toString(16),allowance,"TUSD approve to Engine incorrect")
        assert.equal('0x'+allow_DAI.toString(16),allowance,"DAI approve to Engine incorrect")
    });
    it("用户 B 充值 30USDC 成功",async()=>{
        await
DFProtocol_contract.deposit(USDC_contract.address,web3.utils.toBN(0),web3.utils.toBN(deposit_amount*10**USDC_decimal),{from:accounts[2]})
        let USDC_balance = await USDC_contract.balanceOf.call(DFPool_contract.address);
        assert.equal(USDC_balance.toString(16),(deposit_amount*10**USDC_decimal).toString(16),"USDC deposit faild");
    });
    it("在攻击发生后，由于数据错乱，用户 A 提现用户 B 的 30 USDC 成功",async()=>{
        let withdraw_amount = 30 * 10 ** USDC_decimal;
        await
DFProtocol_contract.withdraw(USDC_contract.address,web3.utils.toBN(0),web3.utils.toBN(withdraw_amount),{from:accounts[1]});
        let USDCToken_balance = await USDC_contract.balanceOf.call(DFPool_contract.address);
        assert.equal(USDCToken_balance.toString(16),'0',"user A withdraw fail");
    });
    /*
    *此时状态：1、用户 A(accounts[1])充值了 30 USDC， 30 PAX，30 TUSD，30 DAI 触发了铸币，用户 A 余额 USDC=PAX=TUSD=0，DAI=20
                    用户提币 20 DAI，用户 A 余额 USDC=DAI=PAX=TUSD=0
    *           2、用户 A 继续充值了 100 个 USDC 用户 A 余额 USDC=100，TUSD=PAX=TUSD=DAI=0
```

```
    *           3、黑客转移了所有代币到 accounts[2]
    *           4、此时用户 B(accounts[2])充值 30 USDC 进入合约 用户 B 余额 USDC:30 PAX=TUSD=DAI=0
    *           5、用户 A 取走 用户 B 30 USDC 的份额
    *           ----------+-----------------+------------------+--------------------+------------+--------------+
    *          |    用户    | UserBalance       | TokenBalance       | PoolRealTokenBalance | USDxBalance |
Resbalance     |
    *           ----------+-----------------+------------------+--------------------+------------+--------------+
    *          |    userA   | USDC:70 DAI:0     | USDC:100 DAI:0     | USDC:0 DAI:0          |    100        |USDC:0
PAX:0    |
    *          |            | PAX:0 TUSD:0      | PAX:0 TUSD:0       | PAX:0 TUSD:0          |               |DAI:0
TUSD:0    |
    *           ----------+-----------------+------------------+--------------------+------------+--------------+
    *          |   userB    | USDC:30 DAI:0     | USDC:100 DAI:0     | USDC:0 DAI:0          |     100       |USDC:0
PAX:0    |
    *          |            | PAX:0 TUSD:0      | PAX:0 TUSD:0       | PAX:0 TUSD:0          |               |DAI:0
TUSD:0    |
    *          +----------+-----------------+------------------+--------------------+------------+--------------+
    */
    it("第一轮结束用户 A 状态检查成功",async()=>{
        let userAUSDC_balance = await DFStore_contract.getDepositorBalance(accounts[1],xUSDC.address);
        let userADAI_balance = await DFStore_contract.getDepositorBalance(accounts[1],xDAI.address);
        let userAPAX_balance = await DFStore_contract.getDepositorBalance(accounts[1],xPAX.address);
        let userATUSD_balance = await DFStore_contract.getDepositorBalance(accounts[1],xTUSD.address);
        assert.equal(userAUSDC_balance.toString(16),(70*10**USDx_decimal).toString(16),"userA USDC balance
incorrect");
        assert.equal(userADAI_balance.toString(16),'0',"userA DAI balance incorrect");
        assert.equal(userAPAX_balance.toString(16),'0',"userA PAX balance incorrect");
        assert.equal(userATUSD_balance.toString(16),'0',"userA TUSD balance incorrect");
    })
    it("第一轮结束用户 B 状态检查成功",async()=>{
        let userAUSDC_balance = await DFStore_contract.getDepositorBalance(accounts[2],xUSDC.address);
        let userADAI_balance = await DFStore_contract.getDepositorBalance(accounts[2],xDAI.address);
        let userAPAX_balance = await DFStore_contract.getDepositorBalance(accounts[2],xPAX.address);
        let userATUSD_balance = await DFStore_contract.getDepositorBalance(accounts[2],xTUSD.address);
        assert.equal(userAUSDC_balance.toString(16),(30*10**USDx_decimal).toString(16),"userB USDC balance
incorrect");
        assert.equal(userADAI_balance.toString(16),'0',"userB DAI balance incorrect");
        assert.equal(userAPAX_balance.toString(16),'0',"userB PAX balance incorrect");
        assert.equal(userATUSD_balance.toString(16),'0',"userB TUSD balance incorrect");
    });
    it("第一轮结束后 Pool 状态校验成功",async()=>{
        let PoolUSDC_balance = await DFStore_contract.getTokenBalance(xUSDC.address);
```

```
        let PoolPAX_balance = await DFStore_contract.getTokenBalance(xPAX.address);
        let PoolDAI_balance = await DFStore_contract.getTokenBalance(xDAI.address);
        let PoolTUSD_balance = await DFStore_contract.getTokenBalance(xTUSD.address);
        assert.equal(PoolUSDC_balance.toString(16),(100*10**USDx_decimal).toString(16),"Pool USDC balance incorrect");
        assert.equal(PoolPAX_balance.toString(16),'0',"Pool PAX balance incorrect");
        assert.equal(PoolDAI_balance.toString(16),'0',"Pool DAI balance incorrect");
        assert.equal(PoolTUSD_balance.toString(16),'0',"Pool TUSD balance incorrect");

    });
    it("用户 A 转账 10000 USDC 给 用户 B 成功",async()=>{
        await USDC_contract.transfer(accounts[2],web3.utils.toBN(10000*10**USDC_decimal),{from:accounts[1]});
        let userBUSDC_balance = await USDC_contract.balanceOf(accounts[2]);
        assert.equal(userBUSDC_balance.toString(16),(10100*10**USDC_decimal).toString(16),"UserB USDC balance incorrect")
    });
    it("用户 A 转账 10000 PAX 给 用户 B 成功",async()=>{
        await PAX_contract.transfer(accounts[2],web3.utils.toBN(10000*10**PAX_decimal),{from:accounts[1]});
        let userBPAX_balance = await PAX_contract.balanceOf(accounts[2]);
        assert.equal(userBPAX_balance.toString(16),(10030*10**PAX_decimal).toString(16),"UserB PAX balance incorrect")
    });
    it("用户 A 转账 10000 DAI 给 用户 B 成功",async()=>{
        await DAI_contract.transfer(accounts[2],web3.utils.toBN(10000*10**DAI_decimal),{from:accounts[1]});
        let userBDAI_balance = await DAI_contract.balanceOf(accounts[2]);
        assert.equal(userBDAI_balance.toString(16),(10010*10**DAI_decimal).toString(16),"UserB DAI balance incorrect")
    });
    it("用户 A 转账 10000 TUSD 给 用户 B 成功",async()=>{
        await TUSD_contract.transfer(accounts[2],web3.utils.toBN(10000*10**TUSD_decimal),{from:accounts[1]});
        let userBTUSD_balance = await TUSD_contract.balanceOf(accounts[2]);
        assert.equal(userBTUSD_balance.toString(16),(10030*10**TUSD_decimal).toString(16),"UserB TUSD balance incorrect")
    });
    it("用户 B 充值 100 USDC 成功",async()=>{
        await DFProtocol_contract.deposit(USDC_contract.address,web3.utils.toBN(0),web3.utils.toBN(100*10**USDC_decimal),{from:accounts[2]});
        let userBUSDC_balance = await DFStore_contract.getDepositorBalance.call(accounts[2],xUSDC.address);
        assert.equal(userBUSDC_balance.toString(16),(130*10**USDx_decimal).toString(16),"userB USDC balance incorrect");
    });
```

```
    it("用户 B 充值 100 PAX 成功",async()=>{
        await DFProtocol_contract.deposit(PAX_contract.address,web3.utils.toBN(0),web3.utils.toBN(100*10**PAX_decimal),{from:accounts[2]});
        let userBPAX_balance = await DFStore_contract.getDepositorBalance.call(accounts[2],xPAX.address);
        assert.equal(userBPAX_balance.toString(16),(100*10**USDx_decimal).toString(16),"userB PAX balance incorrect");
        });
    it("用户 B 充值 100 TUSD 成功",async()=>{
        await DFProtocol_contract.deposit(TUSD_contract.address,web3.utils.toBN(0),web3.utils.toBN(100*10**TUSD_decimal),{from:accounts[2]});
        let userBTUSD_balance = await DFStore_contract.getDepositorBalance.call(accounts[2],xTUSD.address);
        assert.equal(userBTUSD_balance.toString(16),(100*10**USDx_decimal).toString(16),"userB TUSD balance incorrect");
        });
    it("用户 B 充值 100 DAI 成功",async()=>{
        await DFProtocol_contract.deposit(DAI_contract.address,web3.utils.toBN(0),web3.utils.toBN(100*10**DAI_decimal),{from:accounts[2]});
        let userBDAI_balance = await DFStore_contract.getDepositorBalance.call(accounts[2],xDAI.address);
        assert.equal(userBDAI_balance.toString(16),(67*10**USDx_decimal).toString(16),"userB DAI balance incorrect");
        });
    it("检查铸币状态",async()=>{
        let USDx_balance = await DFStore_contract.getTotalMinted.call();
        assert.equal(USDx_balance.toString(16),(430*10**USDx_decimal).toString(16),"USDx Minted incorrect");
    });
    it("第二次铸币后 Pool 状态检查",async()=>{
        let USDCTokenBalance = await DFStore_contract.getTokenBalance.call(xUSDC.address);
        assert.equal(USDCTokenBalance.toString(16),(101*10**USDx_decimal).toString(16),"Pool USDC balance incorrect");
        let PAXTokenBalance = await DFStore_contract.getTokenBalance.call(xPAX.address);
        let DAITokenBalance = await DFStore_contract.getTokenBalance.call(xDAI.address);
        let TUSDTokenBalance = await DFStore_contract.getTokenBalance.call(xTUSD.address);
        assert.equal(PAXTokenBalance.toString(16),(1*10**USDx_decimal).toString(16),"Pool USDC balance incorrect");
        assert.equal(DAITokenBalance.toString(16),(67*10**USDx_decimal).toString(16),"Pool DAI balance incorrect");
        assert.equal(TUSDTokenBalance.toString(16),(1*10**USDx_decimal).toString(16),"Pool TUSD balance incorrect");
    });
    it("用户 A 充值 30 PAX",async()=>{
```

```
        await DFProtocol_contract.deposit(PAX_contract.address,web3.utils.toBN(0),web3.utils.toBN(30*10**PAX_decimal),{from:accounts[1]});
        let userAPAX_balance = await DFStore_contract.getDepositorBalance.call(accounts[1],xPAX.address);
        assert.equal(userAPAX_balance.toString(16),(30*10**USDx_decimal).toString(16),"user A PAX balance incorrect")
    });
    it("用户 A 充值 30 TUSD",async()=>{
        await DFProtocol_contract.deposit(TUSD_contract.address,web3.utils.toBN(0),web3.utils.toBN(30*10**TUSD_decimal),{from:accounts[1]});
        let userATUSD_balance = await DFStore_contract.getDepositorBalance.call(accounts[1],xTUSD.address);
        assert.equal(userATUSD_balance.toString(16),'0',"user A TUSD balance incorrect")
    });
    it("用户 B claim USDx",async()=>{
        await DFProtocol_contract.claim(web3.utils.toBN(0),{from:accounts[2]});
        let userBUSDx_balance = await USDx_contract.balanceOf.call(accounts[2]);
        assert.equal(userBUSDx_balance.toString(16),(340*10**USDx_decimal).toString(16),"user B USDx balance incorrect");
    });
    it("用户 B 一键铸币",async()=>{
        await DFProtocol_contract.oneClickMinting(web3.utils.toBN(0),web3.utils.toBN(10*10**USDx_decimal),{from:accounts[2]});
        let userBUSDx_balance = await USDx_contract.balanceOf.call(accounts[2]);
        assert.equal(userBUSDx_balance.toString(16),(350*10**USDx_decimal).toString(16),"user B USDx balance incorrect");
    });
    it("用户 B 尝试融币 30",async()=>{
        await DFProtocol_contract.destroy(web3.utils.toBN(0),web3.utils.toBN(30*10**USDx_decimal),{from:accounts[2]});
    });
    //脚本功能测试完毕，开始进行危机测试
    it("用户 A 尝试使用低于铸币额度铸币,但是失败了",async()=>{
        try{
            await DFProtocol_contract.oneClickMinting(web3.utils.toBN(0),web3.utils.toBN(1),{from:accounts[2]});
        }catch(err){
            assert.include(err.message,"OneClickMinting: amount error.");
        }
    });
    it("用户 B 尝试充值 0.99 TUSD 成功",async()=>{
```

```
        await DFProtocol_contract.deposit(TUSD_contract.address,web3.utils.toBN(0),web3.utils.toBN(0.99*10**TUSD_decimal),{from:accounts[2]});
        let userBTUSD_balance = await DFStore_contract.getDepositorBalance.call(accounts[2],xTUSD.address);
        assert.equal(userBTUSD_balance.toString(16),(1.99 * 10 ** USDx_decimal).toString(16),"user B TUSD balance incorrect");
    });
    it("用户 B 尝试充值 0.99 PAX 成功",async()=>{
        await DFProtocol_contract.deposit(PAX_contract.address,web3.utils.toBN(0),web3.utils.toBN(0.99*10**PAX_decimal),{from:accounts[2]});
        let userBPAX_balance = await DFStore_contract.getDepositorBalance.call(accounts[2],xPAX.address);
        assert.equal(userBPAX_balance.toString(16),(1.99 * 10 ** USDx_decimal).toString(16),"user B PAX balance incorrect");
    });
    it("用户 B 尝试充值 0.99 USDC 成功",async()=>{
        await DFProtocol_contract.deposit(USDC_contract.address,web3.utils.toBN(0),web3.utils.toBN(0.99*10**USDC_decimal),{from:accounts[2]});
        let userBUSDC_balance = await DFStore_contract.getDepositorBalance.call(accounts[2],xUSDC.address);
        assert.equal(userBUSDC_balance.toString(16),(31.99 * 10 ** USDx_decimal).toString(16),"user B USDC balance incorrect");
    });
    it("用户 B 尝试充值 0.99 DAI 成功",async()=>{
        await DFProtocol_contract.deposit(DAI_contract.address,web3.utils.toBN(0),web3.utils.toBN(0.99*10**DAI_decimal),{from:accounts[2]});
        let userBDAI_balance = await DFStore_contract.getDepositorBalance.call(accounts[2],xDAI.address);
        assert.equal(userBDAI_balance.toString(16),(57.99 * 10 ** USDx_decimal).toString(16),"user B DAI balance incorrect");
    });
    it("因为没有达到铸币阀值所以没有铸币",async()=>{
        let USDx_minted = DFStore_contract.getTotalMinted.call();
        assert(USDx_minted.toString(16),(530 * 10 ** USDx_decimal,"USDx minted amount incorrect"));
    });
    it("部署 GUSD 假充值合约成功",async()=>{
        GUSDFalse_contract = await StableCoinFalse.new(accounts[2],web3.utils.toBN(TUSD_decimal));
        let GUSD_falseBalance = await GUSDFalse_contract.balanceOf.call(accounts[2]);
        assert(GUSD_falseBalance.toString(16),'1000000000000000000'.toString(16),"TUSDFalse contract deploy failed")
    });
    it("部署 xGUSD 假充值合约成功",async()=>{
```

```
        xGUSDFalse = await
WrapToken.new(GUSDFalse_contract.address,DAI_decimal,web3.utils.stringToHex("xGUSDFalse"));
        //成分币授权给合约
        GUSDFalse_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[2]})
        GUSDFalse_contract.approve(DFPool_contract.address,web3.utils.toBN(allowance),{from:accounts[1]})
        //授权给 Engine
        await xGUSDFalse.setAuthority(DFEngine_contract.address,{from:accounts[0]});
        //DFPool 授权额度给  Engine
        await DFPool_contract.approveToEngine(xGUSDFalse.address,DFEngine_contract.address,{from:accounts[0]});
        //col  授权给 Engine
        await col_contract.approveToEngine(xGUSDFalse.address,DFEngine_contract.address,{from:accounts[0]});
    });

    it("更新成分币",async()=>{
        let gusdfalseW = web3.utils.toBN(1*10**USDx_decimal);
        let newtusdW = web3.utils.toBN(1*10**USDx_decimal);
        let newusdcW = web3.utils.toBN(3*10**USDx_decimal);
        let newdaiW = web3.utils.toBN(3*10**USDx_decimal);
        let newpaxW = web3.utils.toBN(2*10**USDx_decimal);
        await
DFSetting_contract.updateMintSection([xGUSDFalse.address,xTUSD.address,xUSDC.address,xDAI.address,xPAX.addres
s],[gusdfalseW,newtusdW,newusdcW,newdaiW,newpaxW]);
    });
    it("检查更新成分币后的状态",async()=>{
        let SectionData = await DFStore_contract.getSectionData(web3.utils.toBN(1));
        let minted = SectionData['0'];
        let gusdfalseW = web3.utils.toBN(1*10**USDx_decimal);
        let newtusdW = web3.utils.toBN(1*10**USDx_decimal);
        let newusdcW = web3.utils.toBN(3*10**USDx_decimal);
        let newdaiW = web3.utils.toBN(3*10**USDx_decimal);
        let newpaxW = web3.utils.toBN(2*10**USDx_decimal);
        let burned = SectionData['1'];
        let backupIdx = SectionData['2'];
        let collDs = SectionData['3'];
        let cw = SectionData['4'];
        assert.equal(minted.toString(16),0,"new section mint incorrect");
        assert.equal(burned.toString(16),0,"new burned incorrect");
        assert.equal(backupIdx.toString(16),0,"new backupIdx incorrect");
        for(i=0;i<collDs.length;i++){

assert.equal(collDs[i],[xGUSDFalse.address,xTUSD.address,xUSDC.address,xDAI.address,xPAX.address][i],"new collDs is
not correct");
```

```
        }
        for(i=0;i<cw.length;i++){

assert.equal(cw[i].toString(16),[gusdfalseW,newtusdW,newusdcW,newdaiW,newpaxW][i].toString(16),"new cw incorrect");
        }
    });
    //成分币 true/false 模型 假充值测试
    it("用户 A 尝试利用 GUSD 假充值合约漏洞进行攻击，但是失败了",async()=>{
        await GUSDFalse_contract.transfer(accounts[1],web3.utils.toBN(20*10**TUSD_decimal),{from:accounts[2]});
        let userAGUSDFalse_balance = await GUSDFalse_contract.balanceOf(accounts[1]);
        assert.equal(userAGUSDFalse_balance.toString(16),(20*10**TUSD_decimal).toString(16),"userA GUSD balance incorrect");
        try{
            await DFProtocol_contract.deposit(GUSDFalse_contract.address,web3.utils.toBN(0),web3.utils.toBN(30),{from:accounts[1]});
        }catch(err){
            assert.include(err.message,'');
        }
    });
    it("用户 B 尝试提现 60 DAI,但他实际只有 57.99，事件声明他实际只提现了 57.99",async()=>{
        let withdraw_amount = await web3.utils.toBN(60*10**DAI_decimal);
        let withdraw_result = await DFProtocol_contract.withdraw(DAI_contract.address,web3.utils.toBN(0),web3.utils.toBN(withdraw_amount),{from:accounts[2]});

assert.equal(withdraw_result.logs[0].args._expectedAmount.toString(16),withdraw_amount.toString(16),'withdraw event incorrect');

assert.equal(withdraw_result.logs[0].args._actualAmount.toString(16),(57.99*10**DAI_decimal).toString(16),'withdraw event incorrect');

    });
    it("销毁 owner 权限后黑客无法继续作恶",async()=>{
        await DFPool_contract.disableOwnership({from:accounts[0]});
        try{
        await DFPool_contract.transferOut(USDC_contract.address,accounts[2],1*10**USDC_decimal);
        }catch(err){
            assert.include(err.message,'ds-auth-unauthorized');
        }
    });
})
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account