USDx Protocol

Smart Contract Security Audit

July 17, 2019

Abstract

This report provides a comprehensive view of the security audit results for the smart contract code for the USDx Protocol project. The task of SlowMist is to review and point out security issues in the audited smart contract code.

Disclaimer

This audit report does not imply any warranty on the security of the smart contract code. SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility. SlowMist is not responsible for anything that happens after the report is issued, because there is no guarantee that the security status of its smart contract will be issued after the report is issued. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). If the information provided is missing, tampered, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom.

Summary

In this report, SlowMist audited the smart contract code for the USDx Protocol project. The audit results showed that no critical severity or high severity issues were found. However, some medium severity and low severity issues were discovered. After communication and feedback, the issues have been fixed.

Project Overview

Description

The following are the details of the smart contract code of the audited project (USDx Protocol):

Project Site：https://github.com/dforce-network/USDx Protocol

Initial audit commit：fa7f72917ec1be8a20c291cbc50fc20137fccf4a(v0.4)

Lastest fix commit：07a53474c796906704888d97076b881487ac3bdb(v0.7)

Project Structure

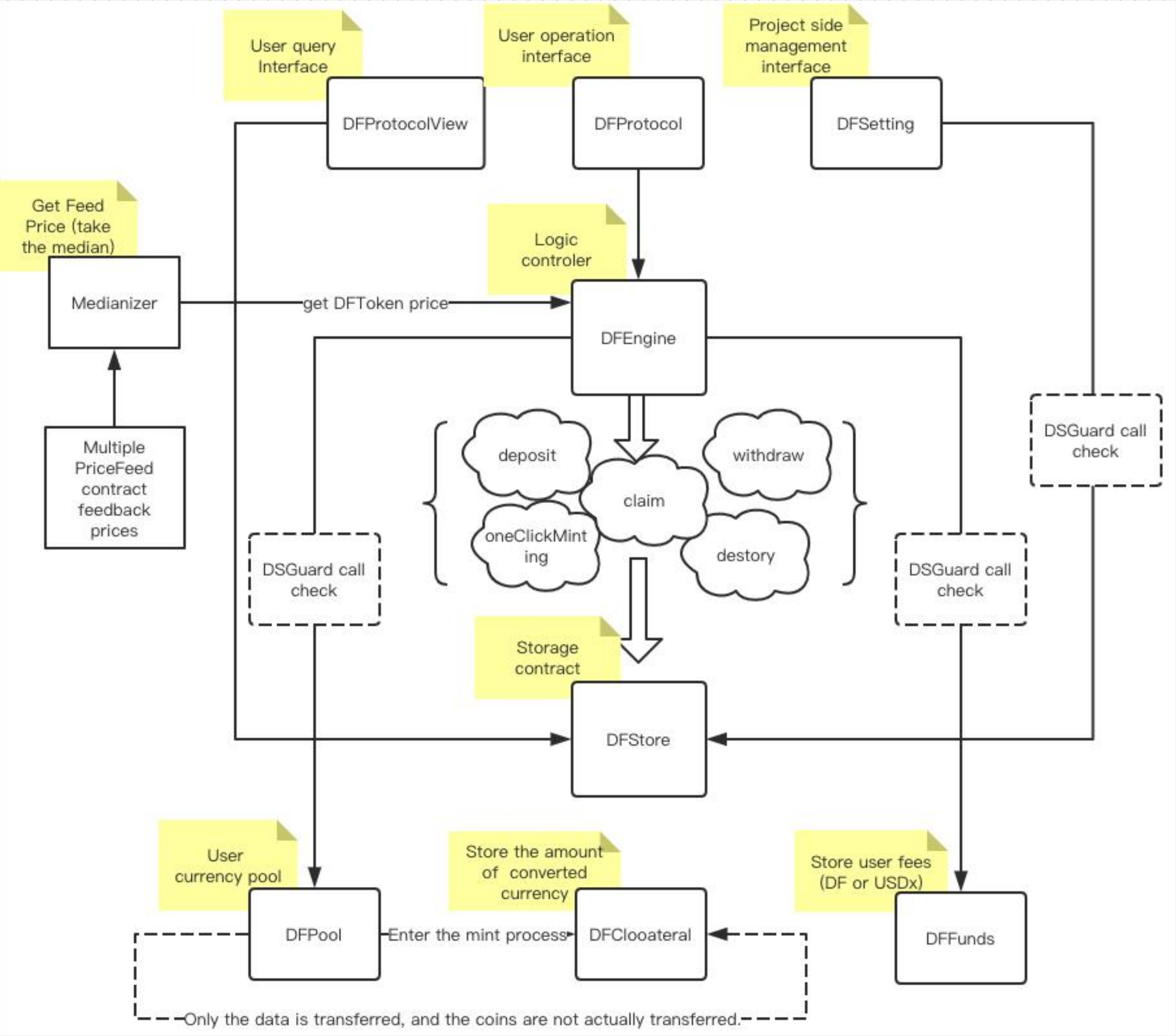
The project includes the following smart contract files:

```
./contracts
├── converter
|   ├── DFEngine.sol
|   ├── DFProtocol.sol
|   ├── DFProtocolView.sol
|   ├── DFSetting.sol
|   └── interfaces
|       ├── IDFEngine.sol
|       └── IDFProtocol.sol
├── helpers
|   └── Migrations.sol
├── oracle
|   ├── 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
│   ├── DSToken.sol
│   ├── DSWrappedToken.sol
│   └── interfaces
│       ├── IDSToken.sol
│       ├── IDSWrappedToken.sol
│       └── IERC20Token.sol
├── update
│   └── DFUpgrader.sol
└── utility
    ├── DSAuth.sol
    ├── DSGuard.sol
    ├── DSMath.sol
    ├── DSNote.sol
    ├── DSThing.sol
    ├── DSValue.sol
    └── Utils.sol
```

Contracts Structure

The contracts are developed using the dapphub/dappsys framework, and the DSGurad contract is used to manage permissions for calls between contracts. The contracts structure are divided into logical layer, user interface layer, management layer and storage layer. The overall structure of the contracts is shown below:



Audit Methodology

The security audit process for smart contracts consists of the following two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house automated analysis tools.
- Manually audit the security of the code. Discover the potential security issues in the code by manually analyzing the contract code.

The following is a list of common vulnerabilities that will be highlighted during the contract code audit:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Data Storage issue
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack
- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Bussiness Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication

- "False top-up" Vulnerability
- Event Security
- Compiler version issues
- Call function Security

Audit Result

Critical Severity

Critical severity issues can have a major impact on the security of smart contracts, and it is highly recommended to fix critical severity issues.

The audit has shown no critical severity issues.

High Severity

High severity issues can affect the normal operation of smart contracts, and it is highly recommended to fix high severity issues.

The audit has shown no high severity issues.

Medium Severity

Medium severity issues can affect the operation of a smart contract, and it is recommended to fix medium severity issues.

Excessive authority issue

1. In storage/DFFunds.sol and storage/DfPool.sol, the auth privilege setting of transferOut

function is too large. If the owner account is controled by hacker, he or she can transfer all users' coin to the DFPool through deposit function and then steal all of it through transferOut function in DFPool.

Proof of truffle test code：

```
it("hacker transfer all of the token",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("after transfering all the token, data is shown changed",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 transfer fail");
            assert.equal(PAX_balance.toString(16),'0',"PAX transfer fail");
            assert.equal(TUSD_balance.toString(16),'0',"TUSD transfer fail");
            assert.equal(DAI_balance.toString(16),'0',"DAI transfer fail");
            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 transfer fail");
            assert.equal(PAXToken_balance.toString(16),'0',"PAX transfer fail");
            assert.equal(TUSDToken_balance.toString(16),'0',"TUSD transfer fail");
            assert.equal(DAIToken_balance.toString(16),'0',"DAI transfer fail");
       });
```

Fix Status：The issue has been fixed in v0.7, by adding disableOwnership function，which can renounce ownership while the project is run stablely.

Low Severity

Low severity issues can influence smart contracts operation in future versions of code. We recommend taking them into account.

Missing check of the code

1. token / DSToken.sol line: 32

In the setOwner(address owner_) function, it is recommended to check if owner_ != address(0) to prevent loss of permissions due to mistakes.

```
function setOwner(address owner_)
        public
        onlyOwner
    {
        owner = owner_;
        emit LogSetOwner(owner);
    }
```

Fix Status: The issue has been fixed in v0.6 and is not present in the latest version of the code.

2. token/DSToken.sol   line:40

In the setOwner(address owner_) function, it is recommended to check if authority_ != address(0) to avoid setting a wrong authority address.

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

Fix Status: After communicating with project side, this issue will not be fixed.

Code redundancy issue

1. converter / DFEngine.sol line: 25 TokenType enumeration type, code redundancy,

converter/DFProcotolView.sol line:13 ProcessType enumeration type, code redundancy line:20

TokenType enumeration type, code redundancy.

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// code redundancy here
    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// code redundancy here
    enum ProcessType {
        CT_DEPOSIT,
        CT_DESTROY,
        CT_CLAIM,
        CT_WITHDRAW
    }

    //SlowMist// code redundancy here
    enum TokenType {
        TT_DF,
        TT_USDX
    }
```

Fix Status: The issue has been fixed in v0.7 and is not present in the latest version of the code.

Event declaration issue

1. The "_amount" in the Withdraw event in converter/DFProtocol.sol may not match the actual withdrawal amount (the last parameter name "_balance"). The user can construct any amount by himself, causing the user to withdraw the currency that exceeds the balance . "_amount" and "_balance" in the Withdraw event are inconsistent. It may cause a wrong judgment by a third party that is listening to this event.

```
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;
}
```

Fix Status: The issue has been fixed in v0.7 and is not present in the latest version of the code.

Front-end UI issues

1. When the testnet.dforce.network UI is clicked to unlock the component currency authorization, the value of the default approve is uint(-1). This value is too large, there is a certain risk, and the user may mistakenly think that it is a transfer. The amount of the mount, the user experience is not very good, it is recommended to approve the actual amount that user need to deposit, reduce the risk.

Fix Status: After communicating with the project party, the issue of approve only involves the front-end modification, which will be modified according to user feedback.

Appendix

truffle test file

```
//Storage Contract
const DFStore = artifacts.require("DFStore");
//Setting Contract
const DFSetting = artifacts.require("DFSetting");
//Logic Contract
const DFEngine = artifacts.require("DFEngine");
//PriceFeed contract
const PriceFeed = artifacts.require("PriceFeed");
//Average Price Calculate contract
const Medianizer = artifacts.require("Medianizer");
//xToken
const StableCoin = artifacts.require("ERC20");
//USDx
const USDx = artifacts.require("DSToken");
//WrapToken
const WrapToken = artifacts.require("DSWrappedToken");
```

```
//col contract
const col = artifacts.require("DFCollateral")
//DFFunds contract
const DFFunds = artifacts.require("DFFunds")
//DFPool contract
const DFPool = artifacts.require("DFPool")
//guard contract
const DSGuard = artifacts.require("DSGuard")
//protocol contract
const DFProtocol = artifacts.require("DFProtocol")
//protocolViewcontract
const ProtocolView = artifacts.require("DFProtocolView")
//A Defective ERC20 Contract
const StableCoinFalse = artifacts.require("ERC20False")
contract('USDx test',async accounts =>{
    before(async()=>{
        amount = '10000000000000000000000000000';// 10000000000*10**18;
        allowance = '0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff';
        initial_amount = '10000000000';
        USDx_decimal = 18;
        USDC_decimal = 6;
        PAX_decimal =   12;
        DAI_decimal =   12;
        TUSD_decimal = 8;
        deposit_amount = 30;
        //Token balance initialize
        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 contract deployment and balance initialize
        USDx_contract = await USDx.new(web3.utils.stringToHex("USDx"));
        //DF contract deployment and balance initialize
        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]});
        //WrapToken deployment
        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"));
        //Storage Contract deployment
        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]);
        //col contract deployment
        col_contract = await col.new()
        //DFFunds contract deployment
        DFFunds_contract = await DFFunds.new(DF_contract.address);
        //DFPool contract deployment
        DFPool_contract = await DFPool.new(col_contract.address);
        //Medianizer contract deployment
        Medianizer_contract = await Medianizer.new();
        //PriceFeed
        PriceFeed_contract = await PriceFeed.new();
        //DFEngine contract deployment
        DFEngine_contract = await
DFEngine.new(USDx_contract.address,DFStore_contract.address,DFPool_contract.address,col_contract.address,DFFun
ds_contract.address);
        //user approve DF Token 和 USDx token to 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]});
        //Setting contract deployment
        DFSetting_contract = await DFSetting.new(DFStore_contract.address,{from:accounts[0]});
        //Token approve to Pool
        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]})
        //approve to 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 approve xToken allowance to   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 approve to 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 approve to Engine
        await USDx_contract.setAuthority(DFEngine_contract.address,{from:accounts[0]});
        //Guard contract deployment
        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]});
        //Protocol contract deployment
        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);
        //ProtocolView contract deployment
        ProtocolView_contract = await ProtocolView.new(DFStore_contract.address,col_contract.address);

    });
    it("Token contract deploy correctly",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 deploy correctly",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 contract deploy correctly",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("xToken contract deploy correctly",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("Storage contract deploy correctly",async()=>{
        let SectionData = await DFStore_contract.getSectionData.call(0);
        let minted = SectionData['0'];
        let burned = SectionData['1'];
        let backupIdx = SectionData['2'];
        let colIDs = 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<colIDs.length;i++){
            assert.equal(colIDs[i],[xDAI.address,xPAX.address,xUSDC.address,xTUSD.address][i],"colIDs is not
correct");
        }
        for(i=0;i<cw.length;i++){
            assert.equal(cw[i].toString(),[daiW,paxW,tusdW,usdcW][i],"cw incorrect");
        }

    });
    it("Token approve to Pool correctly",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 correctly",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("col contract deploy correctly",async()=>{
            let col_owner = await col_contract.owner.call();
            assert.equal(col_owner,accounts[0],"col owner incorrect")
        })
        it("deposit 30USDC success",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("deposit 30PAX success",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("deposit30TUSD success",async()=>{
            await
DFProtocol_contract.deposit(TUSD_contract.address,web3.utils.toBN(0),web3.utils.toBN(deposit_amount*10**TUSD_d
ecimal),{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("deposit 30DAI success",async()=>{
            await
DFProtocol_contract.deposit(DAI_contract.address,web3.utils.toBN(0),web3.utils.toBN(deposit_amount*10**DAI_deci
mal),{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("check col each token status is correct after the fist mint",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("check user obtain correct USDx token",async()=>{
        let mintAmount = await USDx_contract.balanceOf.call(accounts[1]);
        assert.equal(mintAmount.toString(),100*10**USDx_decimal,"USDx mint incorrect")
    });
    it("check Token Pool token amount is correct after the first mint",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("check user balance after first deposit ",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("user try to withdraw 30 DAI",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("user deposit 100 USDC success",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("hacker transfer all of the token",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("after transfering all the token, data is shown changed",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 transfer fail");
        assert.equal(PAX_balance.toString(16),'0',"PAX USDC transfer fail");
        assert.equal(TUSD_balance.toString(16),'0',"TUSD USDC transfer fail");
        assert.equal(DAI_balance.toString(16),'0',"DAI USDC transfer fail");
        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),"TUSDUSDC transfer
fail");
        assert.equal(PAXToken_balance.toString(16),'0',"USDC transfer fail");
        assert.equal(TUSDToken_balance.toString(16),'0',"USDC transfer fail");
        assert.equal(DAIToken_balance.toString(16),'0',"DAI USDC transfer fail");
    });
    it("user try to withdraw but fail",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("user B approve to 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("user B deposit 30USDC success",async()=>{
            await
DFProtocol_contract.deposit(USDC_contract.address,web3.utils.toBN(0),web3.utils.toBN(deposit_amount*10**USDC_d
ecimal),{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("after attacked by hacker,because if the incorrect data,user A withdraw user B’  s USDC success",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:ac
counts[1]});
            let USDCToken_balance = await USDC_contract.balanceOf.call(DFPool_contract.address);
            assert.equal(USDCToken_balance.toString(16),'0',"user A withdraw fail");
      });
      /*
      *Current Status ： 1、user A(accounts[1]) deposit 30 USDC，  30 PAX，30 TUSD，30 DAI and trigger mint，user A
balance USDC=PAX=TUSD=0，DAI=20
                           user A withdraw 20 DAI，user A balance   USDC=DAI=PAX=TUSD=0
      *                2、user A deposit 100 个 USDC user A balance USDC=100，TUSD=PAX=TUSD=DAI=0
      *                3、hacker transfer all token to accounts[2]
      *                4、user B(accounts[2])deposit   30 USDC into contract user B balance USDC:30 PAX=TUSD=DAI=0
      *                5、user A withdraw user B’  s 30 USDC
      it("check user A status success after first round",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("check user B status success after first round",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("check Pool status success after first round",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("user A transfer 10000 USDC to user B success",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("user A transfer 10000 PAX to user B success",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("user A transfer 10000 DAI to user B success",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("user A transfer 10000 TUSD to user B success",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("user B deposit   100 USDC success",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("user B deposit   100 PAX success",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("user B deposit   100 TUSD success",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("user B deposit   100 DAI success",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("check mint status",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("check Pool status after second mint",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("user A deposit 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("user A deposit 30 TUSD success",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("user clam 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("user B oneClickMinting",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 try to destory 30 USDx",async()=>{
        await
DFProtocol_contract.destroy(web3.utils.toBN(0),web3.utils.toBN(30*10**USDx_decimal),{from:accounts[2]});
    });
    //The script function is tested and the evil test begins.
    it("user A try to mint USDx below the limit but fail",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("user B deposit 0.99 TUSD success",async()=>{
        await
DFProtocol_contract.deposit(TUSD_contract.address,web3.utils.toBN(0),web3.utils.toBN(0.99*10**TUSD_decimal),{fro
m: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("user B deposit 0.99 PAX success",async()=>{
        await
DFProtocol_contract.deposit(PAX_contract.address,web3.utils.toBN(0),web3.utils.toBN(0.99*10**PAX_decimal),{from:a
ccounts[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("user B deposit 0.99 USDC success",async()=>{
        await
DFProtocol_contract.deposit(USDC_contract.address,web3.utils.toBN(0),web3.utils.toBN(0.99*10**USDC_decimal),{fro
m: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("user B deposit 0.99 DAI success",async()=>{
        await
DFProtocol_contract.deposit(DAI_contract.address,web3.utils.toBN(0),web3.utils.toBN(0.99*10**DAI_decimal),{from:ac
counts[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("because does not reach the threshold so not trigger mint",async()=>{
        let USDx_minted = DFStore_contract.getTotalMinted.call();
        assert(USDx_minted.toString(16),(530 * 10 ** USDx_decimal,"USDx minted amount incorrect"));
    });
    it("depkoy GUSD fasle top-up contract success",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("deploy xGUSD false top-up contract success",async()=>{
        xGUSDFalse = await
WrapToken.new(GUSDFalse_contract.address,DAI_decimal,web3.utils.stringToHex("xGUSDFalse"));
        //Token approve to contract
        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]})
        //approve Engine
        await xGUSDFalse.setAuthority(DFEngine_contract.address,{from:accounts[0]});
        //DFPool approve to Engine
        await DFPool_contract.approveToEngine(xGUSDFalse.address,DFEngine_contract.address,{from:accounts[0]});
        //col approve Engine
        await col_contract.approveToEngine(xGUSDFalse.address,DFEngine_contract.address,{from:accounts[0]});
    });

    it("Update Tokens",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("check status after updating tokens",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 colIDs = 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<colIDs.length;i++){

assert.equal(colIDs[i],[xGUSDFalse.address,xTUSD.address,xUSDC.address,xDAI.address,xPAX.address][i],"new colIDs 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");
        }
    });
    //Token true/false model false top-up test
    it("user A trying to exploit the GUSD false top-up contract vulnerability, but failed",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("user B try to withdraw 60 DAI,but his balance is 57.99， the events declare the actual withdraw amount is
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:acco
unts[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("after destroying the owner hacker can not be evil anymore",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');
        }
    });
})
```

慢雾科技
slow mist

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

WeChat Official Account