知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20201027	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档版本号	文档编号	保密级别
farmland 智能合约审计报告	V1.0	FARM-ZNHY-20201027	项目组公开

声明

目录

1. 综述

本次报告有效测试时间是从 2020 年 10 月 23 日开始到 2020 年 10 月 27 日结束，在此期间针对 **farmland 智能合约代码**的安全性和规范性进行审计并以此作为报告统计依据。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，综合评定为**通过**。

本次智能合约安全审计结果：通过

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次测试的目标信息：

条目	描述
Token 名称	farmland（farm）
代码类型	代币代码、DeFi 协议代码、以太坊智能合约代码
代码语言	solidity

合约文件及哈希：

合约文件	MD5
Controller.sol	c8e8acacf2ac4cc0b87828bd29699bdb
Migrations.sol	ca8d6ca8a6edf34f149a5095a8b074c9
StrategyDForceDAI.sol	ff36c68f3bd61573495eb4226e439863
VaultDai.sol	3fb87c174f9e0e85771c484683bae7d0
Farmland.sol	17dd5d2a1eeb569a7ce6ad3473324bb1
RewardsRenbtc.sol	6c19a375aa98af5e5903911f0f0c079f

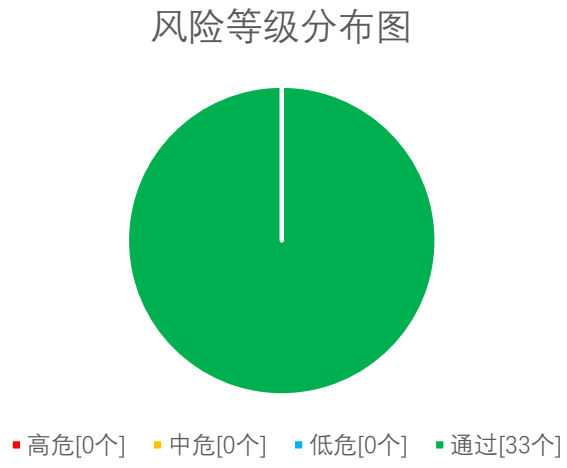
StrategyCRV.sol	439be44d1dc7b053667b0cd07aa0fdf7
VaultRenbtc.sol	775ac820e9bab777c19d649cad7eaa68
StrategyFortubeUsdc.sol	52cea8816fff13555a9974e609ebc1ba
VaultUsdc.sol	2c14fadef90bfe3138c68651e1574d01
StrategyFortube.sol	a0bfc51fce41d6fc1be1cf0050d82509
VaultUsdt.sol	505684a20a4d844657345fadbcf2ad7a

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计:

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	33



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
业务安全性检测	控制器合约变量及构造函数	通过	经检测，不存在安全问题。
	控制器合约 yearn 功能	通过	经检测，不存在安全问题。
	金库合约 deposit 函数	通过	经检测，不存在安全问题。
	金库合约 withdraw 函数	通过	经检测，不存在安全问题。
	策略合约 harvest 函数	通过	经检测，不存在安全问题。
	策略合约 withdraw 函数	通过	经检测，不存在安全问题。
代码基本漏洞检测	编译器版本安全	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.orgin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	不安全的接口使用	通过	经检测，不存在该安全问题。

	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

3. 业务安全性检测

3.1. 控制器合约变量及构造函数【通过】

审计分析：控制器 Controller.sol 合约变量定义及构造函数设计合理。

```
contract Controller {
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;

    address public governance;//knownsec// 治理地址
    address public onesplit;//knownsec// onesplit 交易所地址
    address public rewards;//knownsec// 奖励提取地址
    address public factory;//knownsec// 工厂合约地址
    mapping(address => address) public vaults;//knownsec// 各代币金库合约地址映射
    mapping(address => address) public strategies;//knownsec// 各代币策略合约地址映射
    mapping(address => mapping(address => address)) public converters;//knownsec// 转换器

    uint public split = 5000;//knownsec// 奖励抽成 split/max 50%
    uint public constant max = 10000;

    event NewVault(address indexed _token, address indexed _vault);

    constructor() public {
        governance = tx.origin;
        onesplit = address(0x50FDA034C0Ce7a8f7EFDAebDA7Aa7cA21CC1267e);
        rewards = 0x0EbcA310383A3f67784e7A573dD9499513e36a94;
    }
}
```

安全建议：无。

3.2. 控制器合约 yearn 功能【通过】

审计分析：控制器 Controller.sol 合约的 yearn 功能是收取指定策略的指定代币，从收益中抽成提取奖励后剩余再投入该策略中赚取利息。

```
function yearn(address _strategy, address _token, uint parts) public {
    // This contract should never have value in it, but just incase since this is a public call
    uint _before = IERC20(_token).balanceOf(address(this));//knownsec// 提取指定策略代币前的余额
    Strategy(_strategy).withdraw(_token);//knownsec// 将策略合约的所有 token 提取至本合约
    uint _after = IERC20(_token).balanceOf(address(this));//knownsec// 提取指定策略代币后的余额
    if (_after > _before) {
        uint _amount = _after.sub(_before);//knownsec// 提取先后的差额
        address _want = Strategy(_strategy).want();
        uint[] memory _distribution;
        uint _expected;
        _before = IERC20(_want).balanceOf(address(this));
        IERC20(_token).safeApprove(onesplit, 0);
        IERC20(_token).safeApprove(onesplit, _amount);//knownsec// 授权 onesplit 差值额度
        (_expected, _distribution) = OneSplitAudit(onesplit).getExpectedReturn(_token, _want, _amount, parts, 0);
        OneSplitAudit(onesplit).swap(_token, _want, _amount, _expected, _distribution, 0);
        _after = IERC20(_want).balanceOf(address(this));
        if (_after > _before) {
            _amount = _after.sub(_before);//knownsec// 实际提取额
            uint _reward = _amount.mul(split).div(max);//knownsec// 奖励 = 实际差值 * split / max
            earn(_want, _amount.sub(_reward));
            IERC20(_want).safeTransfer(rewards, _reward);
```

```
        }
    }
}
```

安全建议：无。

3.3. 金库合约 deposit 函数【通过】

审计分析：以 VaultDai.sol 为例，despoit 函数用于存入指定代币获得相应的流动性生息代币 farm 以获取收益利息。

```
function deposit(uint _amount) public {//knownsec// 存入 DAI
    uint _pool = balance();
    uint _before = token.balanceOf(address(this));
    token.safeTransferFrom(msg.sender, address(this), _amount);
    uint _after = token.balanceOf(address(this));
    _amount = _after.sub(_before); // Additional check for deflationary tokens
    uint shares = 0;
    if (totalSupply() == 0) {
        shares = _amount;
    } else {
        shares = (_amount.mul(totalSupply())).div(_pool);//knownsec// 相应 farm 量
    }
    _mint(msg.sender, shares);//knownsec// 存款者获取 shares 量的 farm 代币
    userList.pushAddress(msg.sender);
    if (token.balanceOf(address(this))>earnLowerlimit){//knownsec// 超过设定线自动 earn
        earn();
    }
}
```

安全建议：无。

3.4. 金库合约 withdraw 函数【通过】

审计分析：以 VaultDai.sol 为例，withdraw 函数用于提现策略合约中的流动性收益。

```
function withdraw(uint _shares) public {//knownsec// 提现为DAI
    uint r = (balance().mul(_shares)).div(totalSupply());
    _burn(msg.sender, _shares);

    // Check balance
    uint b = token.balanceOf(address(this));
    if (b < r) {
        uint _withdraw = r.sub(b);//knownsec// 理论差值
        Controller(controller).withdraw(address(token), _withdraw);//knownsec// 从控制器转入理论差值的DAI
        uint _after = token.balanceOf(address(this));
        uint _diff = _after.sub(b);//knownsec// 转入本合约前后的DAI 量差值,即控制器实际转入 DAI 量
        if (_diff < _withdraw) {//knownsec// 若实际差值 < 理论差值
            r = b.add(_diff);//knownsec// 则实际提现量 = 转入前本合约DAI 量 + 实际差值
        }
    }

    uint _max = balanceOf(msg.sender);
    if (_shares == _max) {//knownsec// 若提取完则移除用户
        userList.removeAddress(msg.sender);
    }
    token.safeTransfer(msg.sender, r);//knownsec// 转出
}
```

安全建议：无。

3.5. 策略合约 harvest 函数【通过】

审计分析：以 StrategyDForceDAI.sol 为例，harvest 函数用于对指定策略进行收取利息并分红以及循环生息。

```
function harvest() public {
        _checkHarvest();//knownsec//  校验时间
        require(!Address.isContract(msg.sender),"!contract");
        dRewards(pool).getReward();//knownsec//  收取利息

        doswap();
        dosplit();
        // deposit();
}

function doswap() internal {//knownsec//  将收益 DF 换为 DAI
        uint256 _2token = IERC20(output).balanceOf(address(this));
        UniswapRouter(unirouter).swapExactTokensForTokens(_2token,    0,    swap2TokenRouting,
address(this), now.add(1800));
}
function dosplit() internal{
        uint b = IERC20(want).balanceOf(address(this)).mul(10).div(100);
        uint _fee = b.mul(fee).div(max);
        uint _callfee = b.mul(callfee).div(max);
        IERC20(want).safeTransfer(Controller(controller).rewards(), _fee); //6%    team
        IERC20(want).safeTransfer(msg.sender, _callfee); //call fee 1%

        // other => sent to all users
        address _vault = Controller(controller).vaults(address(want));
        uint other = IERC20(want).balanceOf(address(this));
        address[] memory users = Vault(_vault).getUsers();
        for(uint i=0;i < users.length;i++) {//knownsec//  将剩余 DAI 按比例分红给金库所有用户
```

```
            address _user = users[i];
            uint reward = other.mul(
                Vault(_vault).balanceOf(_user)
            ).div(
                Vault(_vault).totalSupply()
            );
            if (reward > 0) {
                IERC20(want).safeTransfer(_user, reward);
            }
        }
    }
```

安全建议：无。

3.6. 策略合约 withdraw 函数【通过】

审计分析：以 StrategyDForceDAI.sol 为例，withdraw 函数用于提现流动性代币至金库合约。

```
function withdraw(uint _amount) external {
    require(msg.sender == controller, "!controller");
    uint _balance = IERC20(want).balanceOf(address(this));
    if (_balance < _amount) {//knownsec// 若本合约余额不足提现,则从 DAI/dDAI 赎回差额的 DAI
        _amount = _withdrawSome(_amount.sub(_balance));
        _amount = _amount.add(_balance);
    }

    uint _fee = 0;
    if (withdrawalFee>0){//knownsec// 若有提现费,收取 withdrawalFee/withdrawalMax
        _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
        IERC20(want).safeTransfer(Controller(controller).rewards(), _fee);
    }
```

```
        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds
        IERC20(want).safeTransfer(_vault, _amount.sub(_fee));
    }
    function _withdrawSome(uint256 _amount) internal returns (uint) {
        uint _d = _amount.mul(1e18).div(dERC20(d).getExchangeRate());
        uint _before = IERC20(d).balanceOf(address(this));
        dRewards(pool).withdraw(_d);//knownsec//  赎回 dDAI
        uint _after = IERC20(d).balanceOf(address(this));
        uint _withdrew = _after.sub(_before);
        _before = IERC20(want).balanceOf(address(this));
        dERC20(d).redeem(address(this), _withdrew);//knownsec//  赎回 DAI
        _after = IERC20(want).balanceOf(address(this));
        _withdrew = _after.sub(_before);
        return _withdrew;
    }
```

安全建议：无。

4. 代码基本漏洞检测

4.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.5.16 以上，不存在该安全问题。

安全建议：无。

4.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

检测结果：经检测，智能合约代码中已使用 SafeMath 安全算数库，不存在该安全问题。

安全建议：无。

4.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7. tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8. owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他账户余额等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.10. call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.11. 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞

call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.12. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中存在增发代币的功能，但由于流动性挖矿需要增发代币，故通过。

安全建议：无。

4.13. 访问控制缺陷检测【通过】

合约中不同函数应设置合理的权限

检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.14. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.15. 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、函数、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20，而 5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.16. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.17. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.18. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.19. 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

4.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；

传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于以太坊区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.22. 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

在以太坊的世界中，拒绝服务是致命的，遭受该类型攻击的智能合约可能永远无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25. 重入攻击检测【通过】

重入漏洞是最著名的以太坊智能合约漏洞，曾导致了以太坊的分叉（The DAO hack）。

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.27. 重排攻击检测【通过】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表(list)或映射(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果:经检测，智能合约代码中不存在相关漏洞。

安全建议:无。

5. 附录 A：合约代码

本次测试代码来源：

Controller.sol

```
pragma solidity ^0.5.16;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
```

```
}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

interface Strategy {
    function want() external view returns (address);
    function deposit() external;
    function withdraw(address) external;
    function withdraw(uint) external;
    function withdrawAll() external returns (uint);
    function balanceOf() external view returns (uint);
}

interface Converter {
    function convert(address) external returns (uint);
}

interface OneSplitAudit {
    function swap(
        address fromToken,
        address destToken,
        uint256 amount,
        uint256 minReturn,
        uint256[] calldata distribution,
        uint256 flags
    )
        external
        payable
        returns(uint256 returnAmount);

    function getExpectedReturn(
        address fromToken,
        address destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags // See constants in IOneSplit.sol
    )
        external
        view
        returns(
            uint256 returnAmount,
            uint256[] memory distribution
        );
}

contract Controller {
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;

    address public governance;//knownsec// 治理地址
    address public onesplit;//knownsec// onesplit 交易所地址
    address public rewards;//knownsec// 奖励提取地址
    address public factory;//knownsec// 工厂合约地址
    mapping(address => address) public vaults;//knownsec// 各代币金库合约地址映射
    mapping(address => address) public strategies;//knownsec// 各代币策略合约地址映射
    mapping(address => mapping(address => address)) public converters;//knownsec// 转换器
```

```
    uint public split = 5000;//knownsec//  奖励抽成 split/max 50%
    uint public constant max = 10000;


    event NewVault(address indexed _token, address indexed _vault);


    constructor() public {
        governance = tx.origin;
        onesplit = address(0x50FDA034C0Ce7a8f7EFDAebDA7Aa7cA21CC1267e);
        rewards = 0x0EbcA310383A3f67784e7A573dD9499513e36a94;
    }

    function setFactory(address _factory) public {
        require(msg.sender == governance, "!governance");
        factory = _factory;
    }

    function setSplit(uint _split) public {
        require(msg.sender == governance, "!governance");
        split = _split;
    }

    function setOneSplit(address _onesplit) public {
        require(msg.sender == governance, "!governance");
        onesplit = _onesplit;
    }

    function setGovernance(address _governance) public {
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setVault(address _token, address _vault) public {
        //TODO:加个 Event 添加新的策略了.
        require(msg.sender == governance, "!governance");
        vaults[_token] = _vault;
        emit NewVault(_token, _vault);
    }

    function setConverter(address _input, address _output, address _converter) public {
        require(msg.sender == governance, "!governance");
        converters[_input][_output] = _converter;
    }

    function setStrategy(address _token, address _strategy) public {
        //某个币对应一个策略
        require(msg.sender == governance, "!governance");
        address _current = strategies[_token];
        if (_current != address(0)) {//之前的策略存在的话,那么就先提取所有资金
            Strategy(_current).withdrawAll();
        }
        strategies[_token] = _strategy;
    }

    //
    function earn(address _token, uint _amount) public {
        address _strategy = strategies[_token]; //获取策略的合约地址
        address _want = Strategy(_strategy).want();//策略需要的 token 地址
        if (_want != _token) {//如果策略需要的和输入的不一样,需要先转换
            address converter = converters[_token][_want];//转换器合约地址
            IERC20(_token).safeTransfer(converter, _amount);//给转换器打钱
            _amount = Converter(converter).convert(_strategy);//执行转换...
            IERC20(_want).safeTransfer(_strategy, _amount);
        } else {
            IERC20(_token).safeTransfer(_strategy, _amount);
        }
        Strategy(_strategy).deposit();//存钱
    }

    function balanceOf(address _token) external view returns (uint) {
        return Strategy(strategies[_token]).balanceOf();
    }

    function withdrawAll(address _token) public {
        require(msg.sender == governance, "!governance");
        Strategy(strategies[_token]).withdrawAll();
    }

    function inCaseTokensGetStuck(address _token, uint _amount) public {//转任意 erc20
        require(msg.sender == governance, "!governance");
        IERC20(_token).safeTransfer(governance, _amount);
    }

    function getExpectedReturn(address _strategy, address _token, uint parts) public view returns (uint expected)
{
```

```
            uint _balance = IERC20(_token).balanceOf(_strategy);//获取策略器 某个代币的余额
            address _want = Strategy(_strategy).want();//策略器需要的代币.
            (expected,) = OneSplitAudit(onesplit).getExpectedReturn(_token, _want, _balance, parts, 0);
        }

        // Only allows to withdraw non-core strategy tokens ~ this is over and above normal yield
        function yearn(address _strategy, address _token, uint parts) public {
            // This contract should never have value in it, but just incase since this is a public call
            uint _before = IERC20(_token).balanceOf(address(this));//knownsec// 提取指定策略代币前的余额
            Strategy(_strategy).withdraw(_token);//knownsec// 将策略合约的所有 token 提取至本合约
            uint _after =  IERC20(_token).balanceOf(address(this));//knownsec// 提取指定策略代币后的余额
            if (_after > _before) {
                uint _amount = _after.sub(_before);//knownsec// 提取先后的差额
                address _want = Strategy(_strategy).want();
                uint[] memory _distribution;
                uint _expected;
                _before = IERC20(_want).balanceOf(address(this));
                IERC20(_token).safeApprove(onesplit, 0);
                IERC20(_token).safeApprove(onesplit, _amount);//knownsec// 授权 onesplit 差值额度
                (_expected, _distribution) = OneSplitAudit(onesplit).getExpectedReturn(_token, _want, _amount,
parts, 0);
                OneSplitAudit(onesplit).swap(_token, _want, _amount, _expected, _distribution, 0);
                _after = IERC20(_want).balanceOf(address(this));
                if (_after > _before) {
                    _amount = _after.sub(_before);//knownsec// 实际提取额
                    uint _reward = _amount.mul(split).div(max);//knownsec// 奖励 = 实际差值 * split / max
                    earn(_want, _amount.sub(_reward));
                    IERC20(_want).safeTransfer(rewards, _reward);
                }
            }
        }

        function withdraw(address _token, uint _amount) public {
            require(msg.sender == vaults[_token], "!vault");
            Strategy(strategies[_token]).withdraw(_amount);
        }
}
```

Migrations.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.8.0;

contract Migrations {
  address public owner = msg.sender;
  uint public last_completed_migration;

  modifier restricted() {
    require(
      msg.sender == owner,
      "This function is restricted to the contract's owner"
    );
    _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }
}
```

StrategyDForceDAI.sol

```
pragma solidity ^0.5.16;

interface IERC20 {
        function totalSupply() external view returns (uint256);
        function balanceOf(address account) external view returns (uint256);
        function transfer(address recipient, uint256 amount) external returns (bool);
        function allowance(address owner, address spender) external view returns (uint256);
        function decimals() external view returns (uint);
        function name() external view returns (string memory);
        function approve(address spender, uint256 amount) external returns (bool);
        function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
         event Transfer(address indexed from, address indexed to, uint256 value);
         event Approval(address indexed owner, address indexed spender, uint256 value);
}

library SafeMath {
        function add(uint256 a, uint256 b) internal pure returns (uint256) {
                uint256 c = a + b;
                require(c >= a, "SafeMath: addition overflow");

                return c;
        }
        function sub(uint256 a, uint256 b) internal pure returns (uint256) {
                return sub(a, b, "SafeMath: subtraction overflow");
```

```
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
```

```
interface Controller {
        function vaults(address) external view returns (address);
        function rewards() external view returns (address);
}

interface Vault {
        function getUsers() external view returns (address[] memory);
        function balanceOf(address account) external view returns (uint256);
        function totalSupply() external view returns (uint256);
}

interface dRewards {
        function withdraw(uint) external;
        function getReward() external;
        function stake(uint) external;
        function balanceOf(address) external view returns (uint);
        function exit() external;
}

interface dERC20 {
    function mint(address, uint256) external;
    function redeem(address, uint) external;
    function getTokenBalance(address) external view returns (uint);
    function getExchangeRate() external view returns (uint);
}

interface UniswapRouter {
        function swapExactTokensForTokens(uint, uint, address[] calldata, address, uint) external;
}

contract StrategyDForceDAI {
        using SafeERC20 for IERC20;
        using Address for address;
        using SafeMath for uint256;

        address constant public want = address(0x6B175474E89094C44Da98b954EedeAC495271d0F);//knownsec//
DAI
        address constant public d = address(0x02285AcaafEB533e03A7306C55EC031297df9224);//knownsec// dDAI
        address constant public pool = address(0xD2fA07cD6Cd4A5A96aa86BacfA6E50bB3aaDBA8B);//knownsec//
dDAI->DF unipool
        address constant public df = address(0x431ad2ff6a9C365805eBaD47Ee021148d6f7DBe0);//knownsec// DF
        address constant public output = address(0x431ad2ff6a9C365805eBaD47Ee021148d6f7DBe0);//knownsec//
DF
        address constant public unirouter =
address(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);//knownsec// UniswapV2Router
        address constant public weth = address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2); // used for
df <> weth <> usdc route

        address constant public farmland = address(0xa1d0E215a23d7030842FC67cE582a6aFa3CCaB83);


        uint public strategyfee = 0;
        uint public fee = 600;
        uint public callfee = 100;
        uint constant public max = 1000;

        uint public withdrawalFee = 0;
        uint constant public withdrawalMax = 10000;

        address public governance;
        address public controller;

        string public getName;

        address[] public swap2TokenRouting;

        uint256 public minHarvestTimeIntervalSec;
        uint256 public lastHarvestTimestampSec;
        uint256 public harvestWindowLengthSec;
        uint256 public epoch;

        constructor() public {
                governance = msg.sender;
                controller = 0x67b199B87a1bA9948CC73e946dca7c2bac2d6C3F;
                getName = string(
                        abi.encodePacked("farmland:Strategy:",
                                abi.encodePacked(IERC20(want).name(),"DF Token"
                                )
                        ));
                swap2TokenRouting = [output,weth,want];//knownsec// DF <> weth <> DAI
                doApprove();
                lastHarvestTimestampSec = 1603281600; // 2020-10-21 20:00:00 utc+8
                minHarvestTimeIntervalSec = 24 hours;
                harvestWindowLengthSec = 30 * 60;
        }
```

```
    function doApprove () public{
        IERC20(output).safeApprove(unirouter, 0);
        IERC20(output).safeApprove(unirouter, uint(-1));
    }


    function deposit() public {
        uint _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            IERC20(want).safeApprove(d, 0);
            IERC20(want).safeApprove(d, _want);
            dERC20(d).mint(address(this), _want);//knownsec// DAI 转为 dDAI
        }

        uint _d = IERC20(d).balanceOf(address(this));
        if (_d > 0) {
            IERC20(d).safeApprove(pool, 0);
            IERC20(d).safeApprove(pool, _d);
            dRewards(pool).stake(_d);//knownsec// 质押 dDAI
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        require(d != address(_asset), "d");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint _amount) external {
        require(msg.sender == controller, "!controller");
        uint _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {//knownsec// 若本合约余额不足提现,则从 DAI/dDAI 赎回差额的 DAI
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        uint _fee = 0;
        if (withdrawalFee>0){//knownsec// 若有提现费,收取 withdrawalFee/withdrawalMax
            _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(Controller(controller).rewards(), _fee);
        }


        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds
        IERC20(want).safeTransfer(_vault, _amount.sub(_fee));
    }

    // Withdraw all funds, normally used when migrating strategies
    function withdrawAll() external returns (uint balance) {
        require(msg.sender == controller, "!controller");
        _withdrawAll();


        balance = IERC20(want).balanceOf(address(this));

        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds
        IERC20(want).safeTransfer(_vault, balance);
    }

    function _withdrawAll() internal {
        dRewards(pool).exit();
        uint _d = IERC20(d).balanceOf(address(this));
        if (_d > 0) {
            dERC20(d).redeem(address(this),_d);
        }
    }

    function harvest() public {
        _checkHarvest();//knownsec// 校验时间
        require(!Address.isContract(msg.sender),"!contract");
        dRewards(pool).getReward();//knownsec// 收取利息

        doswap();
        dosplit();
        // deposit();
    }

    function doswap() internal {//knownsec// 将收益 DF 换为 DAI
        uint256 _2token = IERC20(output).balanceOf(address(this));
        UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this),
now.add(1800));
```

```
}
function dosplit() internal{
        uint b = IERC20(want).balanceOf(address(this)).mul(10).div(100);
        uint _fee = b.mul(fee).div(max);
        uint _callfee = b.mul(callfee).div(max);
        IERC20(want).safeTransfer(Controller(controller).rewards(), _fee); //6%    team
        IERC20(want).safeTransfer(msg.sender, _callfee); //call fee 1%

        // other => sent to all users
        address _vault = Controller(controller).vaults(address(want));
        uint other = IERC20(want).balanceOf(address(this));
        address[] memory users = Vault(_vault).getUsers();
        for(uint i=0;i < users.length;i++) {//knownsec//  将剩余 DAI 按比例分红给金库所有用户
                address _user = users[i];
                uint reward = other.mul(
                        Vault(_vault).balanceOf(_user)
                ).div(
                        Vault(_vault).totalSupply()
                );
                if (reward > 0) {
                        IERC20(want).safeTransfer(_user, reward);
                }
        }
}

function _withdrawSome(uint256 _amount) internal returns (uint) {
        uint _d = _amount.mul(1e18).div(dERC20(d).getExchangeRate());
        uint _before = IERC20(d).balanceOf(address(this));
        dRewards(pool).withdraw(_d);//knownsec//  赎回 dDAI
        uint _after = IERC20(d).balanceOf(address(this));
        uint _withdrew =  _after.sub(_before);
        _before = IERC20(want).balanceOf(address(this));
        dERC20(d).redeem(address(this), _withdrew);//knownsec//  赎回 DAI
        _after = IERC20(want).balanceOf(address(this));
        _withdrew = _after.sub(_before);
        return _withdrew;
}

function balanceOfWant() public view returns (uint) {
        return IERC20(want).balanceOf(address(this));
}

function balanceOfPool() public view returns (uint) {
        return (dRewards(pool).balanceOf(address(this))).mul(dERC20(d).getExchangeRate()).div(1e18);
}

function getExchangeRate() public view returns (uint) {
        return dERC20(d).getExchangeRate();
}

function balanceOfD() public view returns (uint) {
        return dERC20(d).getTokenBalance(address(this));
}

function balanceOf() public view returns (uint) {
        return balanceOfWant()
                        .add(balanceOfD())
                        .add(balanceOfPool());
}

function setGovernance(address _governance) external {
        require(msg.sender == governance, "!governance");
        governance = _governance;
}

function setController(address _controller) external {
        require(msg.sender == governance, "!governance");
        controller = _controller;
}
function setFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        fee = _fee;
}
function setCallFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        callfee = _fee;
}

function setWithdrawalFee(uint _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        require(_withdrawalFee <=100,"fee >= 1%"); //max:1%
        withdrawalFee = _withdrawalFee;
}

function _checkHarvest() internal {
        // ensure harvest at correct time
        require(now.sub(lastHarvestTimestampSec) >= minHarvestTimeIntervalSec, "too early");
```

```
            require(now.sub(lastHarvestTimestampSec) <
(minHarvestTimeIntervalSec.add(harvestWindowLengthSec)), "too late");

            lastHarvestTimestampSec = lastHarvestTimestampSec.add(minHarvestTimeIntervalSec);

            epoch = epoch.add(1);
        }
    }
```

VaultDai.sol

```
pragma solidity ^0.5.16;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract Context {
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
    constructor () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }
    function owner() public view returns (address) {
        return _owner;
    }
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(_msgSender(), recipient, amount);
```

```
        return true;
    }
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
    function transferFrom(address sender, address recipient, uint256 amount) public returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
        return true;
    }
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "ERC20: burn amount exceeds allowance"));
    }
}

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }
    function name() public view returns (string memory) {
        return _name;
    }
    function symbol() public view returns (string memory) {
        return _symbol;
    }
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
```

```
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

library AddrArrayLib {
    using AddrArrayLib for Addresses;

    struct Addresses {
       address[]  _items;
    }

    /**
     * @notice push an address to the array
     * @dev if the address already exists, it will not be added again
     * @param self Storage array containing address type variables
     * @param element the element to add in the array
     */
    function pushAddress(Addresses storage self, address element) internal {
        if (!exists(self, element)) {
            self._items.push(element);
        }
    }

    /**
     * @notice remove an address from the array
     * @dev finds the element, swaps it with the last element, and then deletes it;
     *          returns a boolean whether the element was found and deleted
     * @param self Storage array containing address type variables
     * @param element the element to remove from the array
     */
    function removeAddress(Addresses storage self, address element) internal returns (bool) {
        for (uint i = 0; i < self.size(); i++) {
```

```
                if (self._items[i] == element) {
                    self._items[i] = self._items[self.size() - 1];
                    self._items.pop();
                    return true;
                }
            }
            return false;
        }

        /**
         * @notice get the address at a specific index from array
         * @dev revert if the index is out of bounds
         * @param self Storage array containing address type variables
         * @param index the index in the array
         */
        function getAddressAtIndex(Addresses storage self, uint256 index) internal view returns (address) {
            require(index < size(self), "the index is out of bounds");
            return self._items[index];
        }

        /**
         * @notice get the size of the array
         * @param self Storage array containing address type variables
         */
        function size(Addresses storage self) internal view returns (uint256) {
           return self._items.length;
        }

        /**
         * @notice check if an element exist in the array
         * @param self Storage array containing address type variables
         * @param element the element to check if it exists in the array
         */
        function exists(Addresses storage self, address element) internal view returns (bool) {
            for (uint i = 0; i < self.size(); i++) {
                if (self._items[i] == element) {
                    return true;
                }
            }
            return false;
        }

        /**
         * @notice get the array
         * @param self Storage array containing address type variables
         */
        function getAllAddresses(Addresses storage self) internal view returns(address[] memory) {
            return self._items;
        }

}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");
```

```
            if (returndata.length > 0) { // Return data is optional
                // solhint-disable-next-line max-line-length
                require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
            }
        }
}

interface Controller {
        function withdraw(address, uint) external;
        function balanceOf(address) external view returns (uint);
        function earn(address, uint) external;
}

contract VaultDai is ERC20, ERC20Detailed {
        using SafeERC20 for IERC20;
        using Address for address;
        using SafeMath for uint256;
        using AddrArrayLib for AddrArrayLib.Addresses;

        IERC20 public token;

        uint public min = 9500;
        uint public constant max = 10000;
        uint public earnLowerlimit; //池内空余资金到这个值就自动 earn

        address public governance;
        address public controller;
        AddrArrayLib.Addresses userList;

        constructor (address _token,uint _earnLowerlimit) public ERC20Detailed(
                string(abi.encodePacked("Farmland ", ERC20Detailed(_token).name())),
                string(abi.encodePacked("farm", ERC20Detailed(_token).symbol())),
                ERC20Detailed(_token).decimals()
        ) {
                token = IERC20(_token);
                governance = tx.origin;
                controller = 0x67b199B87a1bA9948CC73e946dca7c2bac2d6C3F;
                earnLowerlimit = _earnLowerlimit;
        }

        function balance() public view returns (uint) {
                return token.balanceOf(address(this))
                                .add(Controller(controller).balanceOf(address(token)));
        }

        function setMin(uint _min) external {
                require(msg.sender == governance, "!governance");
                min = _min;
        }

        function setGovernance(address _governance) public {
                require(msg.sender == governance, "!governance");
                governance = _governance;
        }

        function setController(address _controller) public {
                require(msg.sender == governance, "!governance");
                controller = _controller;
        }
        function setEarnLowerlimit(uint256 _earnLowerlimit) public{
           require(msg.sender == governance, "!governance");
           earnLowerlimit = _earnLowerlimit;
      }

        // Custom logic in here for how much the vault allows to be borrowed
        // Sets minimum required on-hand to keep small withdrawals cheap
        function available() public view returns (uint) {
                return token.balanceOf(address(this)).mul(min).div(max);
        }

        function earn() public {
                uint _bal = available();
                token.safeTransfer(controller, _bal);
                Controller(controller).earn(address(token), _bal);
        }

        function depositAll() external {
                deposit(token.balanceOf(msg.sender));
        }

        function getUsers() public view returns (address[] memory) {
                return userList.getAllAddresses();
        }

        function deposit(uint _amount) public {//knownsec//  存入 DAI
                uint _pool = balance();
```

```
            uint _before = token.balanceOf(address(this));
            token.safeTransferFrom(msg.sender, address(this), _amount);
            uint _after = token.balanceOf(address(this));
            _amount = _after.sub(_before); // Additional check for deflationary tokens
            uint shares = 0;
            if (totalSupply() == 0) {
                shares = _amount;
            } else {
                shares = (_amount.mul(totalSupply())).div(_pool);//knownsec// 相应farm 量
            }
            _mint(msg.sender, shares);//knownsec// 存款者获取 shares 量的farm 代币
            userList.pushAddress(msg.sender);
            if (token.balanceOf(address(this))>earnLowerlimit){//knownsec// 超过设定线自动 earn
               earn();
            }
        }

        function withdrawAll() external {
            withdraw(balanceOf(msg.sender));
        }

        // No rebalance implementation for lower fees and faster swaps
        function withdraw(uint _shares) public {//knownsec// 提现为 DAI
            uint r = (balance().mul(_shares)).div(totalSupply());
            _burn(msg.sender, _shares);

            // Check balance
            uint b = token.balanceOf(address(this));
            if (b < r) {
                uint _withdraw = r.sub(b);//knownsec// 理论差值
                Controller(controller).withdraw(address(token), _withdraw);//knownsec// 从控制器转入理论差值
的 DAI
                uint _after = token.balanceOf(address(this));
                uint _diff = _after.sub(b);//knownsec// 转入本合约前后的 DAI 量差值,即控制器实际转入 DAI 量
                if (_diff < _withdraw) {//knownsec// 若实际差值 < 理论差值
                    r = b.add(_diff);//knownsec// 则实际提现量 = 转入前本合约 DAI 量 + 实际差值
                }
            }

            uint _max = balanceOf(msg.sender);
            if (_shares == _max) {//knownsec// 若提取完则移除用户
                userList.removeAddress(msg.sender);
            }
            token.safeTransfer(msg.sender, r);//knownsec// 转出
        }

        function getPricePerFullShare() public view returns (uint) {
            return balance().mul(1e18).div(totalSupply());
        }
    }
```

Farmland.sol

```
pragma solidity ^0.5.16;

interface IERC20 {
    function totalSupply() external view returns (uint);
    function balanceOf(address account) external view returns (uint);
    function transfer(address recipient, uint amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint);
    function approve(address spender, uint amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint value);
    event Approval(address indexed owner, address indexed spender, uint value);
}

contract Context {
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
}

contract ERC20 is Context, IERC20 {
    using SafeMath for uint;

    mapping (address => uint) private _balances;

    mapping (address => mapping (address => uint)) private _allowances;

    uint private _totalSupply;
    function totalSupply() public view returns (uint) {
        return _totalSupply;
    }
    function balanceOf(address account) public view returns (uint) {
```

```
        return _balances[account];
    }
    function transfer(address recipient, uint amount) public returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
    function allowance(address owner, address spender) public view returns (uint) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
    function transferFrom(address sender, address recipient, uint amount) public returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
amount exceeds allowance"));
        return true;
    }
    function increaseAllowance(address spender, uint addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20:
decreased allowance below zero"));
        return true;
    }
    function _transfer(address sender, address recipient, uint amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
    function _mint(address account, uint amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
    function _burn(address account, uint amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(address owner, address spender, uint amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
}

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }
    function name() public view returns (string memory) {
        return _name;
    }
    function symbol() public view returns (string memory) {
        return _symbol;
    }
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

library SafeMath {
    function add(uint a, uint b) internal pure returns (uint) {
        uint c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
```

```
    function sub(uint a, uint b) internal pure returns (uint) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
        require(b <= a, errorMessage);
        uint c = a - b;

        return c;
    }
    function mul(uint a, uint b) internal pure returns (uint) {
        if (a == 0) {
            return 0;
        }

        uint c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint a, uint b) internal pure returns (uint) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint c = a / b;

        return c;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
}

library SafeERC20 {
    using SafeMath for uint;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

contract Farmland is ERC20, ERC20Detailed {
  using SafeERC20 for IERC20;
  using Address for address;
  using SafeMath for uint;


  address public governance;
  mapping (address => bool) public minters;

  constructor () public ERC20Detailed("farm.land", "FARM", 18) {
      governance = msg.sender;
  }

  function mint(address account, uint amount) public {
```

```
        require(minters[msg.sender], "!minter");
        _mint(account, amount);
    }

    function setGovernance(address _governance) public {
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function addMinter(address _minter) public {
        require(msg.sender == governance, "!governance");
        minters[_minter] = true;
    }

    function removeMinter(address _minter) public {
        require(msg.sender == governance, "!governance");
        minters[_minter] = false;
    }
}
```

RewardsRenbtc.sol

```
/*
   ____          __             __        __  _
  / __/__ __ ___ / /_ ___  ___ / /_ __ __ \ \/ /
 _\ \ / // /(_-</ __// -_)/ _ \\__// // /  \  /
/___/ \_, //___/\__/ \__//_//_/   /_/ \_,_/ /_/
     /___/

*
* Docs: https://docs.synthetix.io/
*
*
* MIT License
* ===========
*
* Copyright (c) 2020 Synthetix
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in all
* copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
*/

// File: @openzeppelin/contracts/math/Math.sol

pragma solidity ^0.5.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    /**
     * @dev Returns the largest of two numbers.
     */
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    /**
     * @dev Returns the smallest of two numbers.
     */
    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }

    /**
     * @dev Returns the average of two numbers. The result is rounded towards
     * zero.
     */
    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
```

```
}

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
```

```
        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
```

```
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
```

```
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    function mint(address account, uint amount) external;
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
```

```
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     *
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0;



/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
```

```
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //   1. The target address is checked to verify it contains contract code
        //   2. The call itself is made, and success asserted
        //   3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0;


contract IRewardDistributionRecipient is Ownable {
    function notifyRewardAmount(uint256 reward) external;
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0;

contract LPTokenWrapper {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 public renbtc = IERC20(0x0A9ADD98C076448CBcFAcf5E457DA12ddbEF4A8f);

    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;
```

```
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function deposit(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        renbtc.safeTransferFrom(msg.sender, address(this), amount);
    }

    function withdraw(uint256 amount) public {
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        renbtc.safeTransfer(msg.sender, amount);
    }
}

contract RewardsRenbtc is LPTokenWrapper, IRewardDistributionRecipient {
    IERC20 public farm = IERC20(0xe1AB5C9b806F8898f8194CD369adDc66997a9907);

    uint256 public constant DURATION = 7 days;
    uint256 public starttime = 1602864000; //utc+8 2020-10-17 00:00:00
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;

    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);

    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = claimable_tokens(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }

    function rewardPerToken() public view returns (uint256) {
        if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime)
                    .mul(rewardRate)
                    .mul(1e18)
                    .div(totalSupply())
            );
    }

    function claimable_tokens(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
                .div(1e18)
                .add(rewards[account]);
    }

    // stake visibility is public as overriding LPTokenWrapper's stake() function
    function deposit(uint256 amount) public updateReward(msg.sender) checkStart {
        require(amount > 0, "Cannot stake 0");
        super.deposit(amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) public updateReward(msg.sender) checkStart {
        require(amount > 0, "Cannot withdraw 0");
        super.withdraw(amount);
        emit Withdrawn(msg.sender, amount);
    }
```

```
    function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
    }

    function getReward() public updateReward(msg.sender) checkStart {
        uint256 reward = claimable_tokens(msg.sender);
        if (reward > 0) {
            rewards[msg.sender] = 0;
            farm.safeTransfer(msg.sender, reward);
            emit RewardPaid(msg.sender, reward);
        }
    }

    modifier checkStart() {
        require(block.timestamp > starttime, "not start");
        _;
    }

    function notifyRewardAmount(uint256 reward)
        external
        onlyOwner
        updateReward(address(0))
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        farm.mint(address(this), reward);
        emit RewardAdded(reward);
    }
}
```

StrategyCRV.sol

```
pragma solidity ^0.5.15;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function decimals() external view returns (uint);
    function name() external view returns (string memory);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
```

```
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

interface Controller {
    function vaults(address) external view returns (address);
    function rewards() external view returns (address);
}

interface CurveDeposit{
    function deposit(uint256) external;
    function withdraw(uint256) external;
    function balanceOf(address) external view returns (uint256);
    function claimable_tokens(address) external view returns (uint256);
}

interface CurveMinter{
    function mint(address) external;
}

interface ICurveFi {
  function get_virtual_price() external view returns (uint);
  function add_liquidity(
    uint256[2] calldata amounts,
```

```
        uint256 min_mint_amount
    ) external;
    function remove_liquidity_imbalance(
        uint256[2] calldata amounts,
        uint256 max_burn_amount
    ) external;
    function remove_liquidity(
        uint256 _amount,
        uint256[2] calldata amounts
    ) external;
    function remove_liquidity_one_coin(
        uint256 _amount,
        int128 i,
        uint256 min
    ) external;
    function exchange(
        int128 from, int128 to, uint256 _from_amount, uint256 _min_to_amount
    ) external;
}

interface UniswapRouter {
    function swapExactTokensForTokens(
            uint amountIn,
            uint amountOutMin,
            address[] calldata path,
            address to,
            uint deadline
        ) external returns (uint[] memory amounts);
        function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory amounts);

}

contract StrategyCRV  {
        using SafeERC20 for IERC20;
        using Address for address;
        using SafeMath for uint256;

        address public constant want = address(0xEB4C2781e4ebA804CE9a9803C67d0893436bB27D); // renbtc
        address public constant curveminter = address(0xd061D61a4d941c39E5453435B6345Dc261C2fcE0); // Token minter
        address constant public unirouter = address(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
        address constant public weth = address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);

        address constant public curve = address(0x93054188d876f558f4a66B2EF1d97d16eDf0895B); // Curve.fi:REN Swap
        address public constant curvedeposit = address(0xB1F2cdeC61db658F091671F5f199635aEF202CAC); // Curve.fi: renCrv Gauge
        address public constant rencrv = address(0x49849C98ae39Fff122806C06791Fa73784FB3675); // Curve.fi: renCrv Token
        address constant public output = address(0xD533a949740bb3306d119CC777fa900bA034cd52); // crv

        uint constant public DENOMINATION = 10 ** 10;

        uint public fee = 600;
        uint public callfee = 100;
        uint constant public max = 1000;

        uint public withdrawalFee = 0;
        uint constant public withdrawalMax = 10000;

        address public governance;
        address public controller;

        string public getName;

        address[] public swap2TokenRouting;

        constructor() public {
                governance = tx.origin;
                controller = 0x67D320cf7148D69058477B2b86991D2C1dE60E86;
                getName = string(
                        abi.encodePacked("farmland:Strategy:",
                                abi.encodePacked(IERC20(want).name(),
                                        abi.encodePacked(":",IERC20(output).name())
                                )
                        ));
                doApprove();
                swap2TokenRouting = [output,weth,want]; //knownsec// crv <> weth <> renbtc
        }

        function deposit() public {
                // renbtc -> ren
                uint _renbtc = IERC20(want).balanceOf(address(this));
                if (_renbtc > 0) {
                        IERC20(want).safeApprove(curve, 0);
                        IERC20(want).safeApprove(curve, _renbtc);
```

```
            ICurveFi(curve).add_liquidity([_renbtc, 0], 0);
        }
        uint _rencrv = IERC20(rencrv).balanceOf(address(this));
        if (_rencrv > 0) {
            IERC20(rencrv).safeApprove(curvedeposit, 0);
            IERC20(rencrv).safeApprove(curvedeposit, _rencrv);
            CurveDeposit(curvedeposit).deposit(_rencrv);
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        require(rencrv != address(_asset), "rencrv");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint _amount) external {
        require(msg.sender == controller, "!controller");
        uint _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            uint _diff = _amount.sub(_balance);
            // calculate amount of rencrv lp to withdraw for amount of _want
            uint _rencrv = _diff.mul(1e18).div(ICurveFi(curve).get_virtual_price());
            _amount = _withdrawSome(_rencrv.mul(DENOMINATION));
            _amount = _amount.add(_balance);
        }
        uint _fee = 0;
        if (withdrawalFee > 0) {
            _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(Controller(controller).rewards(), _fee);
        }

        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds
        IERC20(want).safeTransfer(_vault, _amount.sub(_fee));
    }

    // Withdraw all funds, normally used when migrating strategies
    function withdrawAll() external returns (uint balance) {
        require(msg.sender == controller, "!controller");
        _withdrawAll();
        balance = IERC20(want).balanceOf(address(this));

        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds
        IERC20(want).safeTransfer(_vault, balance);
    }

    function _withdrawAll() internal {
        uint256 b = CurveDeposit(curvedeposit).balanceOf(address(this));
        if (b > 0) {
            _withdrawSome(b);
        }
    }

    function _withdrawSome(uint256 _rencrv) internal returns(uint256) {
        uint _before = IERC20(rencrv).balanceOf(address(this));
        CurveDeposit(curvedeposit).withdraw(_rencrv); // get rencrv
        uint _after = IERC20(rencrv).balanceOf(address(this));

        return withdrawUnderlying(_after.sub(_before));
    }

    function withdrawUnderlying(uint256 _amount) internal returns (uint) {
        IERC20(rencrv).safeApprove(curve, 0);
        IERC20(rencrv).safeApprove(curve, _amount);

        uint _before = IERC20(want).balanceOf(address(this));
        ICurveFi(curve).remove_liquidity_one_coin(_amount, 0, 0);
        uint _after = IERC20(want).balanceOf(address(this));

        return _after.sub(_before);
    }

    function doApprove () public {
        IERC20(output).safeApprove(unirouter, 0);
        IERC20(output).safeApprove(unirouter, uint(-1));
    }

    function harvest() public {
        require(!Address.isContract(msg.sender),"!contract");
        CurveMinter(curveminter).mint(curvedeposit);//get crv
```

```
        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds

        doswap();

        deposit(); //循环生息

        // fee of want
        uint b = IERC20(want).balanceOf(address(this));
        uint _fee = b.mul(fee).div(max);
        uint _callfee = b.mul(callfee).div(max);
        IERC20(want).safeTransfer(Controller(controller).rewards(), _fee); //6% team
        IERC20(want).safeTransfer(msg.sender, _callfee); //call fee 1%

    }
    function doswap() internal {
            uint256 _2token = IERC20(output).balanceOf(address(this)); //100%
            UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting,
address(this), now.add(1800));

            // want -> ren
            uint _renbtc = IERC20(want).balanceOf(address(this)).mul(90).div(100);
            if (_renbtc > 0) {
                IERC20(want).safeApprove(curve, 0);
                IERC20(want).safeApprove(curve, _renbtc);
                ICurveFi(curve).add_liquidity([_renbtc, 0], 0);
            }
    }

    function balanceOf() public view returns (uint) {
        uint _rencrv = CurveDeposit(curvedeposit).balanceOf(address(this)); // amount of rencrv
        uint _amount = (_rencrv.mul(ICurveFi(curve).get_virtual_price()).div(1e18);
        return _amount.div(DENOMINATION);
    }

    function balanceOfPendingReward() public view returns(uint){ //还没有领取的收益有多少...
        return CurveDeposit(curvedeposit).claimable_tokens(address(this));
    }

    function setGovernance(address _governance) external {
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setController(address _controller) external {
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }

    function setFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        fee = _fee;
    }
    function setCallFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        callfee = _fee;
    }
    function setWithdrawalFee(uint _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        require(_withdrawalFee <=100,"fee >= 1%"); //max:1%
        withdrawalFee = _withdrawalFee;
    }
}
```

VaultRenbtc.sol

```
pragma solidity ^0.5.16;
pragma experimental ABIEncoderV2;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract Context {
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
```

```
    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
    constructor () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }
    function owner() public view returns (address) {
        return _owner;
    }
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
    function transferFrom(address sender, address recipient, uint256 amount) public returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
        return true;
    }
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
```

```
    }
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "ERC20: burn
amount exceeds allowance"));
    }
}

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }
    function name() public view returns (string memory) {
        return _name;
    }
    function symbol() public view returns (string memory) {
        return _symbol;
    }
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

interface Controller {
    function withdraw(address, uint) external;
    function balanceOf(address) external view returns (uint);
    function earn(address, uint) external;
}

interface IGateway {
    function mint(bytes32 _pHash, uint256 _amount, bytes32 _nHash, bytes calldata _sig) external returns
(uint256);
    function burn(bytes calldata _to, uint256 _amount) external returns (uint256);
}

interface IGatewayRegistry {
    function getGatewayBySymbol(string calldata _tokenSymbol) external view returns (IGateway);
    function getTokenBySymbol(string calldata _tokenSymbol) external view returns (IERC20);
}
```

```
contract VaultRenbtc is ERC20, ERC20Detailed {
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;

    IERC20 public token;
    IGatewayRegistry public registry;

    event BtcDeposit(uint256 _amount, bytes _msg);
    event BtcWithdrawal(bytes _to, uint256 _amount, bytes _msg);

    uint public min = 9500;
    uint public constant max = 10000;
    uint public earnLowerlimit; //池内空余资金到这个值就自动 earn

    address public governance;
    address public controller;

    constructor (address _token, uint _earnLowerlimit, IGatewayRegistry _registry) public ERC20Detailed(
        string(abi.encodePacked("Farmland ", ERC20Detailed(_token).name())),
        string(abi.encodePacked("farm", ERC20Detailed(_token).symbol())),
        ERC20Detailed(_token).decimals()
    ) {
        token = IERC20(_token);
        governance = tx.origin;
        controller = 0x67D320cf7148D69058477B2b86991D2C1dE60E86;
        earnLowerlimit = _earnLowerlimit;
        registry = _registry;
    }

    function balance() public view returns (uint) {
        return token.balanceOf(address(this))
                .add(Controller(controller).balanceOf(address(token)));
    }

    function setMin(uint _min) external {
        require(msg.sender == governance, "!governance");
        min = _min;
    }

    function setGovernance(address _governance) public {
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setController(address _controller) public {
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }
    function setEarnLowerlimit(uint256 _earnLowerlimit) public{
       require(msg.sender == governance, "!governance");
       earnLowerlimit = _earnLowerlimit;
  }

    // Custom logic in here for how much the vault allows to be borrowed
    // Sets minimum required on-hand to keep small withdrawals cheap
    function available() public view returns (uint) {
        return token.balanceOf(address(this)).mul(min).div(max);
    }

    function earn() public {
        uint _bal = available();
        token.safeTransfer(controller, _bal);
        Controller(controller).earn(address(token), _bal);
    }

    function depositAll() external {
        deposit(token.balanceOf(msg.sender));
    }

    function deposit(uint _amount) public {
        uint _pool = balance();
        uint _before = token.balanceOf(address(this));
        token.safeTransferFrom(msg.sender, address(this), _amount);
        uint _after = token.balanceOf(address(this));
        _amount = _after.sub(_before); // Additional check for deflationary tokens
        uint shares = 0;
        if (totalSupply() == 0) {
            shares = _amount;
        } else {
            shares = (_amount.mul(totalSupply())).div(_pool);
        }
        _mint(msg.sender, shares);
        if (token.balanceOf(address(this)) > earnLowerlimit){
           earn();
        }
```

```
    }

    function withdrawAll() external {
        withdraw(balanceOf(msg.sender));
    }

    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public returns (uint){
        uint r = (balance().mul(_shares)).div(totalSupply());
        _burn(msg.sender, _shares);

        // Check balance
        uint b = token.balanceOf(address(this));
        if (b < r) {
            uint _withdraw = r.sub(b);
            Controller(controller).withdraw(address(token), _withdraw);
            uint _after = token.balanceOf(address(this));
            uint _diff = _after.sub(b);
            if (_diff < _withdraw) {
                r = b.add(_diff);
            }
        }

        token.safeTransfer(msg.sender, r);
        return r;
    }

    function getPricePerFullShare() public view returns (uint) {
        return balance().mul(1e18).div(totalSupply());
    }

    function depositbtc(address _user, bytes memory _msg, uint256 _amount, bytes32 _nHash, bytes memory _sig)
public {
        bytes32 pHash = keccak256(abi.encode(_user, _msg));
        uint256 mintedAmount = registry.getGatewayBySymbol("BTC").mint(pHash, _amount, _nHash, _sig);
        uint _pool = balance();
        uint shares = 0;
        if (totalSupply() == 0) {
            shares = _amount;
        } else {
            shares = (_amount.mul(totalSupply())).div(_pool);
        }
        _mint(_user, shares);
        // if (token.balanceOf(address(this)) > earnLowerlimit) {
        //     earn();
        // }
        emit BtcDeposit(mintedAmount, _msg);
    }

    function batchDepositbtc(address[] memory _users, bytes[] memory _msgs, uint256[] memory _amounts,
bytes32[] memory _nHashes, bytes[] memory _sigs) public {
        require(_users.length > 0, "length zero");
        require(
            _users.length == _msgs.length &&
            _msgs.length == _amounts.length &&
            _nHashes.length == _sigs.length &&
            _amounts.length == _nHashes.length,
            "!length mismatch"
        );
        for (uint index = 0;index < _msgs.length;index++) {
            address _user = _users[index];
            bytes memory _msg = _msgs[index];
            uint256 _amount = _amounts[index];
            bytes32 _nHash = _nHashes[index];
            bytes memory _sig = _sigs[index];
            depositbtc(_user, _msg, _amount, _nHash, _sig);
        }
    }

    function withdrawbtc(bytes calldata _msg, bytes calldata _to, uint256 _shares) external {
        uint _amount = _checkBtcWithdrawal(_shares);
        uint256 burnedAmount = registry.getGatewayBySymbol("BTC").burn(_to, _amount);
        emit BtcWithdrawal(_to, burnedAmount, _msg);
    }

    function _checkBtcWithdrawal(uint _shares) internal returns (uint) {
        uint r = (balance().mul(_shares)).div(totalSupply());
        _burn(msg.sender, _shares);

        // Check balance
        uint b = token.balanceOf(address(this));
        if (b < r) {
            uint _withdraw = r.sub(b);
            Controller(controller).withdraw(address(token), _withdraw);
            uint _after = token.balanceOf(address(this));
            uint _diff = _after.sub(b);
            if (_diff < _withdraw) {
```

```
                    r = b.add(_diff);
                }
            }

            return r;
        }

        function balanceOfBtc(address user) public view returns (uint256) {
            uint _shares = balanceOf(user);
            uint bal = (balance().mul(_shares)).div(totalSupply());
            return bal;
        }
}
```

StrategyFortubeUsdc.sol

```
pragma solidity ^0.5.16;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function decimals() external view returns (uint);
    function name() external view returns (string memory);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
     event Transfer(address indexed from, address indexed to, uint256 value);
     event Approval(address indexed owner, address indexed spender, uint256 value);
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
```

```
        function sendValue(address payable recipient, uint256 amount) internal {
            require(address(this).balance >= amount, "Address: insufficient balance");

            // solhint-disable-next-line avoid-call-value
            (bool success, ) = recipient.call.value(amount)("");
            require(success, "Address: unable to send value, recipient may have reverted");
        }
}

library SafeERC20 {
        using SafeMath for uint256;
        using Address for address;

        function safeTransfer(IERC20 token, address to, uint256 value) internal {
            callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
        }

        function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
            callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
        }

        function safeApprove(IERC20 token, address spender, uint256 value) internal {
            require((value == 0) || (token.allowance(address(this), spender) == 0),
                "SafeERC20: approve from non-zero to non-zero allowance"
            );
            callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
        }
        function callOptionalReturn(IERC20 token, bytes memory data) private {
            require(address(token).isContract(), "SafeERC20: call to non-contract");

            // solhint-disable-next-line avoid-low-level-calls
            (bool success, bytes memory returndata) = address(token).call(data);
            require(success, "SafeERC20: low-level call failed");

            if (returndata.length > 0) { // Return data is optional
                // solhint-disable-next-line max-line-length
                require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
            }
        }
}

interface Controller {
        function vaults(address) external view returns (address);
        function rewards() external view returns (address);
}

interface Vault {
        function getUsers() external view returns (address[] memory);
        function balanceOf(address account) external view returns (uint256);
        function totalSupply() external view returns (uint256);
}

interface UniswapRouter {
        function swapExactTokensForTokens(uint, uint, address[] calldata, address, uint) external;
}
interface For{
        function deposit(address token, uint256 amount) external payable;
        function withdraw(address underlying, uint256 withdrawTokens) external;
        function withdrawUnderlying(address underlying, uint256 amount) external;
        function controller() view external returns(address);

}
interface IFToken {
        function balanceOf(address account) external view returns (uint256);

        function calcBalanceOfUnderlying(address owner)
            external
            view
            returns (uint256);
}

interface IBankController {

        function getFTokeAddress(address underlying)
            external
            view
            returns (address);
}
interface ForReward{
        function claimReward() external;
}

contract StrategyFortubeUsdc {
        using SafeERC20 for IERC20;
        using Address for address;
        using SafeMath for uint256;
```

```
    address constant public want = address(0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48); //usdc
    address constant public output = address(0x1FCdcE58959f536621d76f5b7FfB955baa5A672F); //for
    address constant public unirouter = address(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
    address constant public weth = address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2); // used for for <> weth <> usdc route
    address constant public fortube = address(0xdE7B3b2Fe0E7b4925107615A5b199a4EB40D9ca9);//主合约
    address constant public fortube_reward = address(0xF8Df2E6E46AC00Cdf3616C4E35278b7704289d82); //领取奖励的合约

    uint public fee = 600;
    uint public callfee = 100;
    uint constant public max = 1000;

    uint public withdrawalFee = 0;
    uint constant public withdrawalMax = 10000;

    address public governance;
    address public controller;

    string public getName;

    address[] public swap2TokenRouting;

    uint256 public minHarvestTimeIntervalSec;
    uint256 public lastHarvestTimestampSec;
    uint256 public harvestWindowLengthSec;
    uint256 public epoch;

    constructor() public {
        governance = msg.sender;
        controller = 0x3a725fe399641a0965c30e72bA18aAE6948c97b1;
        getName = string(
            abi.encodePacked("farmland:Strategy:",
                abi.encodePacked(IERC20(want).name(),"The Force Token"
                )
            ));
        swap2TokenRouting = [output, weth, want];
        doApprove();
        lastHarvestTimestampSec = 1603281600; // 2020-10-21 20:00:00 utc+8
        minHarvestTimeIntervalSec = 24 hours;
        harvestWindowLengthSec = 30 * 60;
    }

    function doApprove () public{
        IERC20(output).safeApprove(unirouter, 0);
        IERC20(output).safeApprove(unirouter, uint(-1));
    }

    function deposit() public {
        uint _want = IERC20(want).balanceOf(address(this));
        address _controller = For(fortube).controller();
        if (_want > 0) {
            IERC20(want).safeApprove(_controller, 0);
            IERC20(want).safeApprove(_controller, _want);
            For(fortube).deposit(want, _want);
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint _amount) external {
        require(msg.sender == controller, "!controller");
        uint _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        uint _fee = 0;
        if (withdrawalFee>0){
            _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(Controller(controller).rewards(), _fee);
        }

        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds
        IERC20(want).safeTransfer(_vault, _amount.sub(_fee));
    }

    // Withdraw all funds, normally used when migrating strategies
```

```
    function withdrawAll() external returns (uint balance) {
        require(msg.sender == controller, "!controller");
        _withdrawAll();

        balance = IERC20(want).balanceOf(address(this));

        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds
        IERC20(want).safeTransfer(_vault, balance);
    }

    function _withdrawAll() internal {
        address _controller = For(fortube).controller();
        IFToken fToken = IFToken(IBankController(_controller).getFTokeAddress(want));
        uint b = fToken.balanceOf(address(this));
        For(fortube).withdraw(want,b);
    }

    function harvest() public {
        require(!Address.isContract(msg.sender),"!contract");
        _checkHarvest();
        ForReward(fortube_reward).claimReward();
        doswap();
        dosplit();
        // deposit();
    }

    function doswap() internal {
        uint256 _2token = IERC20(output).balanceOf(address(this));
        UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this),
now.add(1800));
    }
    function dosplit() internal{
        uint b = IERC20(want).balanceOf(address(this)).mul(10).div(100);
        uint _fee = b.mul(fee).div(max);
        uint _callfee = b.mul(callfee).div(max);
        IERC20(want).safeTransfer(Controller(controller).rewards(), _fee); //6% team
        IERC20(want).safeTransfer(msg.sender, _callfee); //call fee 1%

        // other => sent to all users
        address _vault = Controller(controller).vaults(address(want));
        uint other = IERC20(want).balanceOf(address(this));
        address[] memory users = Vault(_vault).getUsers();
        for(uint i=0;i < users.length;i++) {
            address _user = users[i];
            uint reward = other.mul(
                Vault(_vault).balanceOf(_user)
            ).div(
                Vault(_vault).totalSupply()
            );
            if (reward > 0) {
                IERC20(want).safeTransfer(_user, reward);
            }
        }
    }

    function _withdrawSome(uint256 _amount) internal returns (uint) {
        For(fortube).withdrawUnderlying(want,_amount);
        return _amount;
    }

    function balanceOfWant() public view returns (uint) {
        return IERC20(want).balanceOf(address(this));
    }

    function balanceOfPool() public view returns (uint) {
        address _controller = For(fortube).controller();
        IFToken fToken = IFToken(IBankController(_controller).getFTokeAddress(want));
        return fToken.calcBalanceOfUnderlying(address(this));
    }


    function balanceOf() public view returns (uint) {
        return balanceOfWant()
                .add(balanceOfPool());
    }

    function setGovernance(address _governance) external {
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setController(address _controller) external {
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }
    function setFee(uint256 _fee) external{
```

```
        require(msg.sender == governance, "!governance");
        fee = _fee;
    }
    function setCallFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        callfee = _fee;
    }

    function setWithdrawalFee(uint _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        require(_withdrawalFee <=100,"fee >= 1%"); //max:1%
        withdrawalFee = _withdrawalFee;
    }

    function _checkHarvest() internal {
        // ensure harvest at correct time
        require(now.sub(lastHarvestTimestampSec) >= minHarvestTimeIntervalSec, "too early");
        require(now.sub(lastHarvestTimestampSec)                                                    <
(minHarvestTimeIntervalSec.add(harvestWindowLengthSec)), "too late");

        lastHarvestTimestampSec = lastHarvestTimestampSec.add(minHarvestTimeIntervalSec);

        epoch = epoch.add(1);
    }
}
```

VaultUsdc.sol

```
pragma solidity ^0.5.16;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract Context {
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
    constructor () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }
    function owner() public view returns (address) {
        return _owner;
    }
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
```

```
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
    function transferFrom(address sender, address recipient, uint256 amount) public returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
amount exceeds allowance"));
        return true;
    }
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20:
decreased allowance below zero"));
        return true;
    }
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "ERC20: burn
amount exceeds allowance"));
    }
}

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }
```

```
    function name() public view returns (string memory) {
        return _name;
    }
    function symbol() public view returns (string memory) {
        return _symbol;
    }
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library AddrArrayLib {
    using AddrArrayLib for Addresses;

    struct Addresses {
      address[]  _items;
    }

    /**
     * @notice push an address to the array
     * @dev if the address already exists, it will not be added again
     * @param self Storage array containing address type variables
     * @param element the element to add in the array
     */
    function pushAddress(Addresses storage self, address element) internal {
       if (!exists(self, element)) {
          self._items.push(element);
       }
    }

    /**
     * @notice remove an address from the array
     * @dev finds the element, swaps it with the last element, and then deletes it;
     *          returns a boolean whether the element was found and deleted
     * @param self Storage array containing address type variables
     * @param element the element to remove from the array
     */
    function removeAddress(Addresses storage self, address element) internal returns (bool) {
        for (uint i = 0; i < self.size(); i++) {
            if (self._items[i] == element) {
                self._items[i] = self._items[self.size() - 1];
                self._items.pop();
```

```
                return true;
            }
        }
        return false;
    }

    /**
     * @notice get the address at a specific index from array
     * @dev revert if the index is out of bounds
     * @param self Storage array containing address type variables
     * @param index the index in the array
     */
    function getAddressAtIndex(Addresses storage self, uint256 index) internal view returns (address) {
        require(index < size(self), "the index is out of bounds");
        return self._items[index];
    }

    /**
     * @notice get the size of the array
     * @param self Storage array containing address type variables
     */
    function size(Addresses storage self) internal view returns (uint256) {
        return self._items.length;
    }

    /**
     * @notice check if an element exist in the array
     * @param self Storage array containing address type variables
     * @param element the element to check if it exists in the array
     */
    function exists(Addresses storage self, address element) internal view returns (bool) {
        for (uint i = 0; i < self.size(); i++) {
            if (self._items[i] == element) {
                return true;
            }
        }
        return false;
    }

    /**
     * @notice get the array
     * @param self Storage array containing address type variables
     */
    function getAllAddresses(Addresses storage self) internal view returns(address[] memory) {
        return self._items;
    }

}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }
```

```
    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

interface Controller {
    function withdraw(address, uint) external;
    function balanceOf(address) external view returns (uint);
    function earn(address, uint) external;
}

contract VaultUsdc is ERC20, ERC20Detailed {
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;
    using AddrArrayLib for AddrArrayLib.Addresses;

    IERC20 public token;

    uint public min = 9500;
    uint public constant max = 10000;
    uint public earnLowerlimit; //池内空余资金到这个值就自动 earn

    address public governance;
    address public controller;
    AddrArrayLib.Addresses userList;

    constructor (address _token, uint _earnLowerlimit) public ERC20Detailed(
        string(abi.encodePacked("Farmland ", ERC20Detailed(_token).name())),
        string(abi.encodePacked("farm", ERC20Detailed(_token).symbol())),
        ERC20Detailed(_token).decimals()
    ) {
        token = IERC20(_token);
        governance = tx.origin;
        controller = 0x3a725fe399641a0965c30e72bA18aAE6948c97b1;
        earnLowerlimit = _earnLowerlimit;
    }

    function balance() public view returns (uint) {
        return token.balanceOf(address(this))
                .add(Controller(controller).balanceOf(address(token)));
    }

    function setMin(uint _min) external {
        require(msg.sender == governance, "!governance");
        min = _min;
    }

    function setGovernance(address _governance) public {
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setController(address _controller) public {
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }
    function setEarnLowerlimit(uint256 _earnLowerlimit) public{
       require(msg.sender == governance, "!governance");
       earnLowerlimit = _earnLowerlimit;
   }

    // Custom logic in here for how much the vault allows to be borrowed
    // Sets minimum required on-hand to keep small withdrawals cheap
    function available() public view returns (uint) {
        return token.balanceOf(address(this)).mul(min).div(max);
    }
```

```
    function earn() public {
        uint _bal = available();
        token.safeTransfer(controller, _bal);
        Controller(controller).earn(address(token), _bal);
    }

    function depositAll() external {
        deposit(token.balanceOf(msg.sender));
    }

    function getUsers() public view returns (address[] memory) {
        return userList.getAllAddresses();
    }

    function deposit(uint _amount) public {
        uint _pool = balance();
        uint _before = token.balanceOf(address(this));
        token.safeTransferFrom(msg.sender, address(this), _amount);
        uint _after = token.balanceOf(address(this));
        _amount = _after.sub(_before); // Additional check for deflationary tokens
        uint shares = 0;
        if (totalSupply() == 0) {
            shares = _amount;
        } else {
            shares = (_amount.mul(totalSupply())).div(_pool);
        }
        _mint(msg.sender, shares);
        userList.pushAddress(msg.sender);
        if (token.balanceOf(address(this))>earnLowerlimit){
           earn();
        }
    }

    function withdrawAll() external {
        withdraw(balanceOf(msg.sender));
    }

    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {
        uint r = (balance().mul(_shares)).div(totalSupply());
        _burn(msg.sender, _shares);

        // Check balance
        uint b = token.balanceOf(address(this));
        if (b < r) {
            uint _withdraw = r.sub(b);
            Controller(controller).withdraw(address(token), _withdraw);
            uint _after = token.balanceOf(address(this));
            uint _diff = _after.sub(b);
            if (_diff < _withdraw) {
                r = b.add(_diff);
            }
        }

        uint _max = balanceOf(msg.sender);
        if (_shares == _max) {
            userList.removeAddress(msg.sender);
        }
        token.safeTransfer(msg.sender, r);
    }

    function getPricePerFullShare() public view returns (uint) {
        return balance().mul(1e18).div(totalSupply());
    }
}
```

StrategyFortube.sol

```
pragma solidity ^0.5.16;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function decimals() external view returns (uint);
    function name() external view returns (string memory);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
     event Transfer(address indexed from, address indexed to, uint256 value);
     event Approval(address indexed owner, address indexed spender, uint256 value);
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
```

```
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");
```

```
            if (returndata.length > 0) { // Return data is optional
                // solhint-disable-next-line max-line-length
                require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
            }
        }
}

interface Controller {
        function vaults(address) external view returns (address);
        function rewards() external view returns (address);
}

interface Vault {
        function getUsers() external view returns (address[] memory);
        function balanceOf(address account) external view returns (uint256);
        function totalSupply() external view returns (uint256);
}

interface UniswapRouter {
        function swapExactTokensForTokens(uint, uint, address[] calldata, address, uint) external;
}
interface For{
        function deposit(address token, uint256 amount) external payable;
        function withdraw(address underlying, uint256 withdrawTokens) external;
        function withdrawUnderlying(address underlying, uint256 amount) external;
        function controller() view external returns(address);

}
interface IFToken {
        function balanceOf(address account) external view returns (uint256);

        function calcBalanceOfUnderlying(address owner)
            external
            view
            returns (uint256);
}

interface IBankController {

        function getFTokeAddress(address underlying)
            external
            view
            returns (address);
}
interface ForReward{
        function claimReward() external;
}

contract StrategyFortube {
        using SafeERC20 for IERC20;
        using Address for address;
        using SafeMath for uint256;

        address constant public want = address(0xdAC17F958D2ee523a2206206994597C13D831ec7); //usdt
        address constant public output = address(0x1FCdcE58959f536621d76f5b7FfB955baa5A672F); //for
        address constant public unirouter = address(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
        address constant public weth = address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2); // used for
for <> weth <> usdc route
        address constant public fortube = address(0xdE7B3b2Fe0E7b4925107615A5b199a4EB40D9ca9);//主合约.
        address constant public fortube_reward = address(0xF8Df2E6E46AC00Cdf3616C4E35278b7704289d82); //
领取奖励的合约

        uint public fee = 600;
        uint public callfee = 100;
        uint constant public max = 1000;

        uint public withdrawalFee = 0;
        uint constant public withdrawalMax = 10000;

        address public governance;
        address public controller;

        string public getName;

        address[] public swap2TokenRouting;

        uint256 public minHarvestTimeIntervalSec;
        uint256 public lastHarvestTimestampSec;
        uint256 public harvestWindowLengthSec;
        uint256 public epoch;

        constructor() public {
            governance = msg.sender;
            controller = 0x3a725fe399641a0965c30e72bA18aAE6948c97b1;
            getName = string(
                abi.encodePacked("farmland:Strategy:",
                    abi.encodePacked(IERC20(want).name(),"The Force Token"
```

```
                )
            ));
        swap2TokenRouting = [output, want]; //for usdt
        doApprove();
        lastHarvestTimestampSec = 1603281600; // 2020-10-21 20:00:00 utc+8
        minHarvestTimeIntervalSec = 24 hours;
        harvestWindowLengthSec = 30 * 60;
    }

    function doApprove () public{
        IERC20(output).safeApprove(unirouter, 0);
        IERC20(output).safeApprove(unirouter, uint(-1));
    }

    function deposit() public {
        uint _want = IERC20(want).balanceOf(address(this));
        address _controller = For(fortube).controller();
        if (_want > 0) {
            IERC20(want).safeApprove(_controller, 0);
            IERC20(want).safeApprove(_controller, _want);
            For(fortube).deposit(want,_want);
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint _amount) external {
        require(msg.sender == controller, "!controller");
        uint _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        uint _fee = 0;
        if (withdrawalFee>0){
            _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(Controller(controller).rewards(), _fee);
        }

        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds
        IERC20(want).safeTransfer(_vault, _amount.sub(_fee));
    }

    // Withdraw all funds, normally used when migrating strategies
    function withdrawAll() external returns (uint balance) {
        require(msg.sender == controller || msg.sender == governance,"!governance");
        _withdrawAll();

        balance = IERC20(want).balanceOf(address(this));

        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we don't burn the funds
        IERC20(want).safeTransfer(_vault, balance);
    }

    function _withdrawAll() internal {
        address _controller = For(fortube).controller();
        IFToken fToken = IFToken(IBankController(_controller).getFTokeAddress(want));
        uint b = fToken.balanceOf(address(this));
        For(fortube).withdraw(want,b);
    }

    function harvest() public {
        checkHarvest();
        ForReward(fortube_reward).claimReward();
        doswap();
        dosplit();
        // deposit();
    }

    function doswap() internal {
        uint256 _2token = IERC20(output).balanceOf(address(this));
        UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this),
now.add(1800));
    }
    function dosplit() internal{
        uint b = IERC20(want).balanceOf(address(this)).mul(10).div(100);
        uint _fee = b.mul(fee).div(max);
```

```
            uint _callfee = b.mul(callfee).div(max);
            IERC20(want).safeTransfer(Controller(controller).rewards(), _fee); //6% team
            IERC20(want).safeTransfer(msg.sender, _callfee); //call fee 1%

            // other => sent to all users
            address _vault = Controller(controller).vaults(address(want));
            uint other = IERC20(want).balanceOf(address(this));
            address[] memory users = Vault(_vault).getUsers();
            for(uint i=0;i < users.length;i++) {
                address _user = users[i];
                uint reward = other.mul(
                    Vault(_vault).balanceOf(_user)
                ).div(
                    Vault(_vault).totalSupply()
                );
                if (reward > 0) {
                    IERC20(want).safeTransfer(_user, reward);
                }
            }
        }

        function _withdrawSome(uint256 _amount) internal returns (uint) {
            For(fortube).withdrawUnderlying(want,_amount);
            return _amount;
        }

        function balanceOfWant() public view returns (uint) {
            return IERC20(want).balanceOf(address(this));
        }

        function balanceOfPool() public view returns (uint) {
            address _controller = For(fortube).controller();
            IFToken fToken = IFToken(IBankController(_controller).getFTokeAddress(want));
            return fToken.calcBalanceOfUnderlying(address(this));
        }


        function balanceOf() public view returns (uint) {
            return balanceOfWant()
                    .add(balanceOfPool());
        }

        function setGovernance(address _governance) external {
            require(msg.sender == governance, "!governance");
            governance = _governance;
        }

        function setController(address _controller) external {
            require(msg.sender == governance, "!governance");
            controller = _controller;
        }
        function setFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            fee = _fee;
        }
        function setCallFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            callfee = _fee;
        }

        function setWithdrawalFee(uint _withdrawalFee) external {
            require(msg.sender == governance, "!governance");
            require(_withdrawalFee <=100,"fee >= 1%"); //max:1%
            withdrawalFee = _withdrawalFee;
        }

        function setSwap2Token(address[] memory _path) public{
            require(msg.sender == governance, "!governance");
            swap2TokenRouting = _path;
        }

        function _checkHarvest() internal {
            // ensure harvest at correct time
            require(now.sub(lastHarvestTimestampSec) >= minHarvestTimeIntervalSec, "too early");
            require(now.sub(lastHarvestTimestampSec)                                                    <
(minHarvestTimeIntervalSec.add(harvestWindowLengthSec)), "too late");

            lastHarvestTimestampSec = lastHarvestTimestampSec.add(minHarvestTimeIntervalSec);

            epoch = epoch.add(1);
        }
}
```

VaultUsdt.sol

```
pragma solidity ^0.5.16;
```

```
interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract Context {
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
    constructor () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }
    function owner() public view returns (address) {
        return _owner;
    }
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
    function transferFrom(address sender, address recipient, uint256 amount) public returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
```

```
        return true;
    }
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20:
decreased allowance below zero"));
        return true;
    }
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "ERC20: burn
amount exceeds allowance"));
    }
}

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }
    function name() public view returns (string memory) {
        return _name;
    }
    function symbol() public view returns (string memory) {
        return _symbol;
    }
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
```

```
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;

        return c;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

library AddrArrayLib {
    using AddrArrayLib for Addresses;

    struct Addresses {
       address[]  _items;
    }

    /**
     * @notice push an address to the array
     * @dev if the address already exists, it will not be added again
     * @param self Storage array containing address type variables
     * @param element the element to add in the array
     */
    function pushAddress(Addresses storage self, address element) internal {
        if (!exists(self, element)) {
            self._items.push(element);
        }
    }

    /**
     * @notice remove an address from the array
     * @dev finds the element, swaps it with the last element, and then deletes it;
     *          returns a boolean whether the element was found and deleted
     * @param self Storage array containing address type variables
     * @param element the element to remove from the array
     */
    function removeAddress(Addresses storage self, address element) internal returns (bool) {
        for (uint i = 0; i < self.size(); i++) {
            if (self._items[i] == element) {
                self._items[i] = self._items[self.size() - 1];
                self._items.pop();
                return true;
            }
        }
        return false;
    }

    /**
     * @notice get the address at a specific index from array
     * @dev revert if the index is out of bounds
     * @param self Storage array containing address type variables
```

```
     * @param index the index in the array
     */
    function getAddressAtIndex(Addresses storage self, uint256 index) internal view returns (address) {
        require(index < size(self), "the index is out of bounds");
        return self._items[index];
    }

    /**
     * @notice get the size of the array
     * @param self Storage array containing address type variables
     */
    function size(Addresses storage self) internal view returns (uint256) {
      return self._items.length;
    }

    /**
     * @notice check if an element exist in the array
     * @param self Storage array containing address type variables
     * @param element the element to check if it exists in the array
     */
    function exists(Addresses storage self, address element) internal view returns (bool) {
        for (uint i = 0; i < self.size(); i++) {
            if (self._items[i] == element) {
                return true;
            }
        }
        return false;
    }

    /**
     * @notice get the array
     * @param self Storage array containing address type variables
     */
    function getAllAddresses(Addresses storage self) internal view returns(address[] memory) {
        return self._items;
    }

}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

interface Controller {
    function withdraw(address, uint) external;
    function balanceOf(address) external view returns (uint);
    function earn(address, uint) external;
}
```

```
contract VaultUsdt is ERC20, ERC20Detailed {
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;
    using AddrArrayLib for AddrArrayLib.Addresses;

    IERC20 public token;

    uint public min = 9500;
    uint public constant max = 10000;
    uint public earnLowerlimit; //池内空余资金到这个值就自动 earn

    address public governance;
    address public controller;
    AddrArrayLib.Addresses userList;

    constructor (address _token, uint _earnLowerlimit) public ERC20Detailed(
        string(abi.encodePacked("Farmland ", ERC20Detailed(_token).name())),
        string(abi.encodePacked("farm", ERC20Detailed(_token).symbol())),
        ERC20Detailed(_token).decimals()
    ) {
        token = IERC20(_token);
        governance = tx.origin;
        controller = 0x3a725fe399641a0965c30e72bA18aAE6948c97b1;
        earnLowerlimit = _earnLowerlimit;
    }

    function balance() public view returns (uint) {
        return token.balanceOf(address(this))
                .add(Controller(controller).balanceOf(address(token)));
    }

    function setMin(uint _min) external {
        require(msg.sender == governance, "!governance");
        min = _min;
    }

    function setGovernance(address _governance) public {
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setController(address _controller) public {
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }
    function setEarnLowerlimit(uint256 _earnLowerlimit) public{
       require(msg.sender == governance, "!governance");
       earnLowerlimit = _earnLowerlimit;
  }

    // Custom logic in here for how much the vault allows to be borrowed
    // Sets minimum required on-hand to keep small withdrawals cheap
    function available() public view returns (uint) {
        return token.balanceOf(address(this)).mul(min).div(max);
    }

    function earn() public {
        uint _bal = available();
        token.safeTransfer(controller, _bal);
        Controller(controller).earn(address(token), _bal);
    }

    function depositAll() external {
        deposit(token.balanceOf(msg.sender));
    }

    function getUsers() public view returns (address[] memory) {
        return userList.getAllAddresses();
    }

    function deposit(uint _amount) public {
        uint _pool = balance();
        uint _before = token.balanceOf(address(this));
        token.safeTransferFrom(msg.sender, address(this), _amount);
        uint _after = token.balanceOf(address(this));
        _amount = _after.sub(_before); // Additional check for deflationary tokens
        uint shares = 0;
        if (totalSupply() == 0) {
            shares = _amount;
        } else {
            shares = (_amount.mul(totalSupply())).div(_pool);
        }
        _mint(msg.sender, shares);
        userList.pushAddress(msg.sender);
        if (token.balanceOf(address(this))>earnLowerlimit){
```

```
            earn();
        }
    }

    function withdrawAll() external {
        withdraw(balanceOf(msg.sender));
    }

    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {
        uint r = (balance().mul(_shares)).div(totalSupply());
        _burn(msg.sender, _shares);

        // Check balance
        uint b = token.balanceOf(address(this));
        if (b < r) {
            uint _withdraw = r.sub(b);
            Controller(controller).withdraw(address(token), _withdraw);
            uint _after = token.balanceOf(address(this));
            uint _diff = _after.sub(b);
            if (_diff < _withdraw) {
                r = b.add(_diff);
            }
        }

        uint _max = balanceOf(msg.sender);
        if (_shares == _max) {
            userList.removeAddress(msg.sender);
        }
        token.safeTransfer(msg.sender, r);
    }

    function getPricePerFullShare() public view returns (uint) {
        return balance().mul(1e18).div(totalSupply());
    }
}
```

6. 附录 B：安全风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 ETH 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 ETH 导致的拒绝服务漏洞、因 gas 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 ETH 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 gas 触发的事务顺序依赖风险等。

7. 附录 C：智能合约安全审计工具简介

6.1 Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号以太坊虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。 与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

6.2 Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

6.3 securify.sh

Securify 可以验证以太坊智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

6.4 Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

6.5 MAIAN

MAIAN 是一个用于查找以太坊智能合约漏洞的自动化工具，Maian 处理合

约的字节码，并尝试建立一系列交易以找出并确认错误。

6.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7 ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

6.8 Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建以太坊合约并调试交易。

6.9 知道创宇区块链安全审计人员专用工具包

知道创宇渗透测试人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509