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	20200924	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	文档编号	保密级别
YFII v2 机枪池智能合约审计报告	V1.0	YFIIV2-ZNNY-20200924	项目组公开

声明

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

目录

1. 综述

本次报告有效测试时间是从 2020 年 9 月 22 日开始到 2020 年 9 月 24 日结束，在此期间针对 **YFII v2 机枪池智能合约代码**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，发现访问控制问题；存在交易顺序依赖风险，该漏洞利用难度较大；存在增发代币问题，但由于该问题需根据交易所要求而定，故综合评定为通过。

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	
项目名称	YFII V2	
合约地址	Controller	0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f
	StrategyFortubeUSDT	0x1a6eC8EB73bf404112475895d6C8814ad5A7bd96
	iVaultUSDT	0x72Cf258c852Dc485a853370171d46B9D29fD3184
	StrategyCurveYCRVVoter	0x898828957133d4c50030a5A2D55Ca370915E6A77
	iVaultYCRV	0x3E3db9cc5b540d2794DB3861BE5A4887cF77E48B

	StrategyDForceDAI	0xbDD4a57c5EE8558370bb661d29a979657D81258e
	iVaultDAI	0x1e0DC67aEa5aA74718822590294230162B5f2064
	StrategyTUSDCurve	0x30aE128ebCdec11F62cB3fa9C6a0E8269a9AF686
	iVaultTUSD	0x4243f5C8683089b65a9F588B1AE578d5D84bFBC9
	StrategyFortubeUSDC	0x17D5C3FFe2A7c7a1E4567c7501d166B0532C8826
	iVaultUSDC	0x23B4dB3a435517fd5f2661a9c5a16f78311201c1
	StrategyFortubeETH	0x0c3E69eF29cbD32e0732409B748ef317a5F4f0a5
	iVaultETH	0xa8EA49a9e242fFfBdECc4583551c3BcB111456E6
	StrategyFortubeBUSD	0xB5639130ce84dE9684dA10B5E6d6Ec49828E0987
	iVaultBUSD	0xc46d2fC00554f1f874F37e6e3E828A0AdFEFfbcB
	StrategyFortubeHBTC	0xfe6A5A0efb399E2987bEe4d5DB89B925583d144b
	iVaultHBTC	0x26AEdD2205FF8a87AEF2eC9691d77Ce3f40CE6E9
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	e7123ac6b8307cd18b44feada0208b92
StrategyFortubeUSDT.sol	d3efeef6716466721df7d68d2ff7c6be
iVaultUSDT.sol	79966c7995a225db4b4214f7e46f732c
StrategyCurveYCRVVoter.sol	6ae2000311be6ad132fdad3ec772c286
iVaultYCRV.sol	c37b75d4917ac3f6fbedd9eb8ca21632
StrategyDForceDAI.sol	13278008b4e283c68672a5c886783d78
iVaultDAI.sol	c37b75d4917ac3f6fbedd9eb8ca21632
StrategyTUSDCurve.sol	0e727c2f694dd188759acc4044de1dfa
iVaultTUSD.sol	fa9889be789995c1bb04827685ea36f0
StrategyFortubeUSDC.sol	cf7f6f38119d83092a4dc63038b06cf3
iVaultUSDC.sol	fa9889be789995c1bb04827685ea36f0
StrategyFortubeETH.sol	6c0e3beaec6d68c3944dcbd95dafafe2
iVaultETH.sol	ec17ce67a7ab25d89aa2db37a92f31f1
StrategyFortubeBUSD.sol	cdd84bf2247a0699ec608f32a43eede8
iVaultBUSD.sol	c9004162c67c973ed9f868117b422db2
StrategyFortubeHBTC.sol	2e5d292556b938630b0d61f919075859
iVaultHBTC.sol	c9004162c67c973ed9f868117b422db2

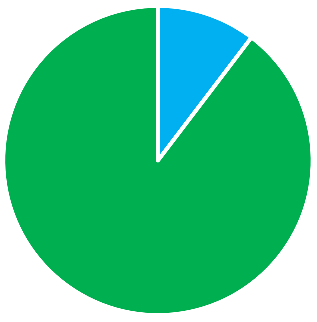
2. 代码漏洞分析

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	3	26

风险等级分布图



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
智能合约	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。
	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。
	访问控制缺陷检测	低危（通过）	经检测，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。
	call 注入攻击	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。
	未初始化的存储指针	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。
	交易顺序依赖检测	低危（通过）	经检测，代码中存在交易顺序依赖风险，但由于利用难度过大，故综合评定为通过。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	逻辑设计缺陷检测	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。
	增发代币漏洞检测	低危（通过）	经检测，代码中存在增发代币功能，但由于需视交易所要求而定，故综合评定为通过。
	冻结账户绕过检测	通过	经检测，不存在该安全问题。
	编译器版本安全	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。
	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。
	require/assert 的使用	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。
	fallback 函数安全	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。
	低级函数安全	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。
	时间戳依赖攻击	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。

3. 代码审计结果分析

3.1 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。

3.2 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。

3.3 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。

3.4 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。

3.5 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。

3.6 访问控制检测【低危】

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，智能合约代码中存在该安全问题。

Controller.sol

```
    // Only allows to withdraw non-core strategy tokens ~ this is over and above normal yield
    function yearn(address _strategy, address _token, uint parts) public {
        //knownsec// 无权限验证,添加以下语句校验
        //knownsec// require(msg.sender == governance, "!governance");
        // This contract should never have value in it, but just incase since this is a public call
        uint _before = IERC20(_token).balanceOf(address(this));
        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));
        if (_after > _before) {
            uint _amount = _after.sub(_before);//knownsec// 提现的实际值
            address _want = Strategy(_strategy).want();
            uint[] memory _distribution;
            uint _expected;
            _before = IERC20(_want).balanceOf(address(this));//knownsec// 本合约的指定策略所需代币量
            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);//knownsec// _token 转换为_want
                _after = IERC20(_want).balanceOf(address(this));//knownsec// 转换后本合约的指定策略所需代币量
                if (_after > _before) {
                    _amount = _after.sub(_before);//knownsec// 转换前后实际差值
                    uint _reward = _amount.mul(split).div(max);//knownsec// 奖励 = 实际差值 * split / max
                    earn(_want, _amount.sub(_reward));//knownsec// 差值 - 奖励 用于策略投资
                    IERC20(_want).safeTransfer(rewards, _reward);//knownsec// 奖励转给 rewards 地址
                }
            }
        }
```

安全建议：添加 require 语句校验调用者为治理地址。

3.7 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。StrategyTUSDCurve.sol 与 StrategyCurveYCRVVoter.sol 这两个策略合约的治理地址采用 tx.origin 而其他策略合约采用 msg.sender，部署时需注意。

```
StrategyTUSDCurve.sol
constructor() public {
    governance = tx.origin;//knownsec// 治理地址为最初的调用者地址,部署时需注意
    controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
}
```

```
StrategyCurveYCRVVoter.sol
constructor() public {
    governance = tx.origin;//knownsec// 治理地址为最初的调用者地址,部署时需注意
    controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
    getName = string(
        abi.encodePacked("yfii:Strategy:",
            abi.encodePacked(IERC20(want).name(),
                abi.encodePacked(":",IERC20(output).name())
            )
        ));
    swap2YFIIRouting = [output,weth,yfii];//knownsec//  df <> weth <> yfii
    swap2TokenRouting = [output,weth,dai];//knownsec//  df <> weth <> DAI
    doApprove();
    strategyDev = tx.origin;
}
```

安全建议：无。

3.8 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。

3.9 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。

3.10 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。

3.11 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。

3.12 交易顺序依赖【低危】

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，智能合约代码中存在该安全问题。

```
    function deposit() public {//knownsec// 流动性挖矿
        uint _want = IERC20(want).balanceOf(address(this));
        address _controller = For(fortube).controller();
        if (_want > 0) {
            //knownsec// 存在交易顺序依赖风险且不符合 safeApprove 函数正确用法,不应注释下行语句
            // IERC20(want).safeApprove(_controller, 0);
            IERC20(want).safeApprove(_controller, _want);
            For(fortube).deposit(want,_want);
        }
    }
```

可能存在的安全风险描述如下：

1. 用户 A 通过调用 approve 函数允许用户 B 代其转账的数量为 N（N>0）；
2. 经过一段时间后，用户 A 决定将 N 改为 M（M>0），所以再次调用 approve

函数；

3. 用户 B 在第二次调用被矿工处理之前迅速调用 transferFrom 函数转账 N 数量的 token；
4. 用户 A 对 approve 的第二次调用成功后，用户 B 便可再次获得 M 的转账额度，即用户 B 通过交易顺序攻击获得了 N+M 的转账额度。

安全建议：删去 IERC20(want).safeApprove(_controller, 0);前的注释。

3.13 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。

3.14 逻辑设计缺陷【通过】

检查智能合约代码中与业务设计相关的安全问题。

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。

3.15 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。

3.16 增发代币漏洞【低危】

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该问题。

```
function _mint(address account, uint amount) internal {//knownsec// 增发代币
        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);
}
```

安全建议：该问题不属于安全问题，但部分交易所会限制增发函数的使用，具体情况需根据交易所的要求而定。

3.17 冻结账户绕过【通过】

检查代币合约中在转移代币时，是否存在未校验代币来源账户、发起账户、目标账户是否被冻结的操作。

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。

3.18 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 0.5.8 以上，不存在该安全问题。

安全建议：无。

3.19 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。

3.20 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问题。

安全建议：无。

3.21 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。

3.22 require/assert 的使用【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。

3.23 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。

3.24 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。

3.25 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。

3.26 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。

3.27 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。

3.28 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。

3.29 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. 附录 A：合约代码

本次测试代码来源：

Controller.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-04
*/

/**
 *Submitted for verification at Etherscan.io on 2020-07-26
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.15;//knownsec//  指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
	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 {//knownsec//  安全算数库
```

```
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 {//knownsec//  策略池接口
        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;
        address public onesplit;
        address public rewards;
        address public factory;
        mapping(address => address) public vaults;
        mapping(address => address) public strategies;
        mapping(address => mapping(address => address)) public converters;//knownsec//  转换器

        uint public split = 5000;
        uint public constant max = 10000;


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


        constructor() public {
            governance = tx.origin;
            onesplit = address(0x50FDA034C0Ce7a8f7EFDAebDA7Aa7cA21CC1267e);
            rewards = 0x887F507EaAc58adD20263C6918538A9BdC882d47;
        }

        function setFactory(address _factory) public {//knownsec//  设置工厂合约,仅治理地址调用
            require(msg.sender == governance, "!governance");
            factory = _factory;
        }

        function setSplit(uint _split) public {//knownsec//  设置 split 值,仅治理地址调用
            require(msg.sender == governance, "!governance");
            split = _split;
        }

        function setOneSplit(address _onesplit) public {//knownsec//  设置 onesplit 地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        onesplit = _onesplit;
    }

    function setGovernance(address _governance) public {//knownsec// 设置治理地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setVault(address _token, address _vault) public {//knownsec// 添加新代币及策略,仅治理地址调用
        //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 {//knownsec// 设置转换器
地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        converters[_input][_output] = _converter;
    }

    function setStrategy(address _token, address _strategy) public {//knownsec// 添加新策略地址,仅治理地址调
用
        //某个币对应一个策略,比如现在的 ycrv 就是挖 yfii
        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 {//knownsec// 提取指定代币所有余额,仅治理地址调用
        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);//knownsec// 将指定代币转移至治理地址账户
    }

    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 {
        //knownsec// 无权限验证,添加以下语句校验
        //knownsec// require(msg.sender == governance, "!governance");
        // This contract should never have value in it, but just incase since this is a public call
        uint _before = IERC20(_token).balanceOf(address(this));
        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));
        if (_after > _before) {
            uint _amount = _after.sub(_before);//knownsec// 提现的实际值
            address _want = Strategy(_strategy).want();
            uint[] memory _distribution;
            uint _expected;
            _before = IERC20(_want).balanceOf(address(this));//knownsec// 本合约的指定策略所需代币量
            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);//knownsec// _token 转换为_want
            _after = IERC20(_want).balanceOf(address(this));//knownsec// 转换后本合约的指定策略所需代币量
            if (_after > _before) {
                _amount = _after.sub(_before);//knownsec// 转换前后实际差值
                uint _reward = _amount.mul(split).div(max);//knownsec// 奖励 = 实际差值 * split / max
                earn(_want, _amount.sub(_reward));//knownsec// 差值 - 奖励 用于策略投资
                IERC20(_want).safeTransfer(rewards, _reward);//knownsec// 奖励转给 rewards 地址
            }
        }
    }

    function withdraw(address _token, uint _amount) public {//knownsec// 提取指定代币一定额度,仅代币库地址调用
        require(msg.sender == vaults[_token], "!vault");
        Strategy(strategies[_token]).withdraw(_amount);
    }
}
```

StrategyFortubeUSDT.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-13
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-13
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.17;//knownsec// 指定编译器版本

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 {//knownsec// 安全算数库
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 {//knownsec// 控制器接口
    function vaults(address) external view returns (address);
    function rewards() external view returns (address);
}

/*

 A strategy must implement the following calls;

 - deposit()
 - withdraw(address) must exclude any tokens used in the yield - Controller role - withdraw should return to Controller
 - withdraw(uint) - Controller | Vault role - withdraw should always return to vault
 - withdrawAll() - Controller | Vault role - withdraw should always return to vault
 - balanceOf()

 Where possible, strategies must remain as immutable as possible, instead of updating variables, we update the contract by linking it in the controller

*/



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); //usdc
//knownsec// 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 yfii = address(0xa1d0E215a23d7030842FC67cE582a6aFa3CCaB83);


    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 strategyfee = 100;//knownsec// 10% strategyfee/max
    uint public fee = 300;//knownsec// 30% fee/max
    uint public burnfee = 500;//knownsec// 50% burnfee/max
    uint public callfee = 100;//knownsec// 10% callfee/max
    uint constant public max = 1000;

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

    address public governance;
    address public strategyDev;
    address public controller;
    address public burnAddress = 0xB6af2DabCEBC7d30E440714A33E5BD45CEEd103a;

    string public getName;

    address[] public swap2YFIIRouting;
    address[] public swap2TokenRouting;


    constructor() public {
        governance = msg.sender;//knownsec// 治理地址为合约部署者
        controller = 0xcDCf1f9Ac816Fed665B09a00f60c885dd8848b02;
        getName = string(
            abi.encodePacked("yfii:Strategy:",
                abi.encodePacked(IERC20(want).name(),"The Force Token"
                )
            ));
        swap2YFIIRouting = [output,weth,yfii];//knownsec// for <> weth <> yfii
        swap2TokenRouting = [output,weth,want];//knownsec// for <> weth <> USDT
        doApprove();
        strategyDev = tx.origin;
    }

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


    function deposit() public {//knownsec// 流动性挖矿
        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) {//knownsec// 提现某资产 for/weth
        require(msg.sender == controller, "!controller");//knownsec// 仅控制器调用
        require(want != address(_asset), "want");//knownsec// 校验不为 USDT
        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){//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 || 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);//knownsec// 本合约所有 USDT 转给对应 vault
    }

    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");
        ForReward(fortube_reward).claimReward();
        doswap();
        dosplit();//分 yfii
        deposit();
    }

    function doswap() internal {
        //knownsec// 建议添加校验本合约 output 余额大于 0,再执行以下 swap
        uint256 _2token = IERC20(output).balanceOf(address(this)).mul(90).div(100); //90%
        uint256 _2yfii = IERC20(output).balanceOf(address(this)).mul(10).div(100);  //10%
        UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this),
now.add(1800));
        UniswapRouter(unirouter).swapExactTokensForTokens(_2yfii, 0, swap2YFIIRouting, address(this),
now.add(1800));
    }
    function dosplit() internal{//knownsec// 分发 yfii
        uint b = IERC20(yfii).balanceOf(address(this));
        uint _fee = b.mul(fee).div(max);//knownsec// 本合约 YFII 量 * fee / max
        uint _callfee = b.mul(callfee).div(max);//knownsec// 本合约 YFII 量 * callfee / max
        uint _burnfee = b.mul(burnfee).div(max);//knownsec// 本合约 YFII 量 * burnfee / max
        IERC20(yfii).safeTransfer(Controller(controller).rewards(), _fee); //3%  3% team
        IERC20(yfii).safeTransfer(msg.sender, _callfee); //call fee 1%
        IERC20(yfii).safeTransfer(burnAddress, _burnfee); //burn fee 5%

        if (strategyfee >0){
            uint _strategyfee = b.mul(strategyfee).div(max); //1%
            IERC20(yfii).safeTransfer(strategyDev, _strategyfee);
        }
    }

    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 setStrategyFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        strategyfee = _fee;
    }
    function setCallFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        callfee = _fee;
    }
    function setBurnFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        burnfee = _fee;
    }
    function setBurnAddress(address _burnAddress) public{
        require(msg.sender == governance, "!governance");
        burnAddress = _burnAddress;
    }

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

iVaultUSDT.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-03
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-13
*/

pragma solidity ^0.5.16;//knownsec// 指定编译器版本

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 {//knownsec// 增发代币
            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;
}

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

    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;

    constructor (address _token,uint _earnLowerlimit) public ERC20Detailed(
        string(abi.encodePacked("yfii ", ERC20Detailed(_token).name())),
        string(abi.encodePacked("i", ERC20Detailed(_token).symbol())),//knownsec// iUSDT
        ERC20Detailed(_token).decimals()
    ) {
        token = IERC20(_token);
        governance = tx.origin;
        controller = 0xe14e60d0F7fb15b1A98FDE88A3415C17b023bf36;
        earnLowerlimit = _earnLowerlimit;
    }

    function balance() public view returns (uint) {//knownsec// 本合约 USDT 量 + 控制器 USDT 量
        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) {//knownsec// 可用量
        return token.balanceOf(address(this)).mul(min).div(max);
    }

    function earn() public {//knownsec// 赚取利息
        uint _bal = available();
        token.safeTransfer(controller, _bal);
        Controller(controller).earn(address(token), _bal);
    }

    function depositAll() external {//knownsec// 存入所有
        deposit(token.balanceOf(msg.sender));
    }

    function deposit(uint _amount) public {//knownsec// 存入 USDT
        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 {//knownsec// 提现所有
        withdraw(balanceOf(msg.sender));
    }


    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {//knownsec// iUSDT 提现为 USDT
        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);
    }

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

StrategyCurveYCRVVoter.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-06
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-28
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.17;//knownsec// 指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
    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 {//knownsec// 安全算数库
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 {//knownsec//  控制器接口
    function vaults(address) external view returns (address);
    function rewards() external view returns (address);
}

/*

 A strategy must implement the following calls;

 - deposit()
 - withdraw(address) must exclude any tokens used in the yield - Controller role - withdraw should return to Controller
 - withdraw(uint) - Controller | Vault role - withdraw should always return to vault
 - withdrawAll() - Controller | Vault role - withdraw should always return to vault
 - balanceOf()

 Where possible, strategies must remain as immutable as possible, instead of updating variables, we update the contract by linking it in the controller

*/

interface Gauge {
    function deposit(uint) external;
    function balanceOf(address) external view returns (uint);
    function withdraw(uint) external;
}

interface Mintr {
        function mint(address) external;
}

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

interface yERC20 {
    function deposit(uint256 _amount) external;
    function withdraw(uint256 _amount) external;
}

interface ICurveFi {

    function get_virtual_price() external view returns (uint);
    function add_liquidity(
        uint256[4] calldata amounts,
        uint256 min_mint_amount
    ) external;
    function remove_liquidity_imbalance(
        uint256[4] calldata amounts,
        uint256 max_burn_amount
    ) external;
    function remove_liquidity(
        uint256 _amount,
        uint256[4] calldata amounts
    ) external;
    function exchange(
        int128 from, int128 to, uint256 _from_amount, uint256 _min_to_amount
    ) external;
}

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

        address constant public want = address(0xdF5e0e81Dff6FAF3A7e52BA697820c5e32D806A8);//knownsec//
yCRV
        address constant public pool = address(0xFA712EE4788C042e2B7BB55E6cb8ec569C4530c1);//knownsec//
yCRV/CRV pool
        address constant public mintr = address(0xd061D61a4d941c39E5453435B6345Dc261C2fcE0);//knownsec//
矿工
        address constant public crv = address(0xD533a949740bb3306d119CC777fa900bA034cd52);//knownsec//
CRV
        address constant public output = address(0xD533a949740bb3306d119CC777fa900bA034cd52);//knownsec//
CRV
        address constant public unirouter =
address(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);//knownsec// UniswapV2Router02
        address constant public weth = address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2); // used for
crv <> weth <> dai route

        address constant public dai = address(0x6B175474E89094C44Da98b954EedeAC495271d0F);//knownsec//
DAI
        address constant public ydai = address(0x16de59092dAE5CcF4A1E6439D611fd0653f0Bd01);//knownsec//
yDAI
        address constant public curve = address(0x45F783CCE6B7FF23B2ab2D70e416cdb7D6055f51);//knownsec//
y Swap

        address constant public yfii = address(0xa1d0E215a23d7030842FC67cE582a6aFa3CCaB83);


        uint public strategyfee = 0;
        uint public fee = 400;//knownsec// 40% fee/max
        uint public burnfee = 500;//knownsec// 50% burnfee/max
        uint public callfee = 100;//knownsec// 10% callfee/max
        uint constant public max = 1000;

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

        address public governance;
        address public controller;
        address public strategyDev;
        address public burnAddress = 0xB6af2DabCEBC7d30E440714A33E5BD45CEEd103a;

        string public getName;

        address[] public swap2YFIIRouting;
        address[] public swap2TokenRouting;


        constructor() public {
                governance = tx.origin;//knownsec// 治理地址为最初的调用者地址,部署时需注意
                controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
                getName = string(
                        abi.encodePacked("yfii:Strategy:",
                abi.encodePacked(IERC20(want).name(),
                    abi.encodePacked(":",IERC20(output).name())
                )
            ));
        swap2YFIIRouting = [output,weth,yfii];//knownsec//   df <> weth <> yfii
        swap2TokenRouting = [output,weth,dai];//knownsec//   df <> weth <> DAI
        doApprove();
        strategyDev = tx.origin;

    }

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


    function deposit() public {
        uint _want = IERC20(want).balanceOf(address(this));//knownsec//  本合约yToken 量
        if (_want > 0) {
            IERC20(want).safeApprove(pool, 0);
            IERC20(want).safeApprove(pool, _want);
            Gauge(pool).deposit(_want);//knownsec//  质押挖矿
        }

    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint balance) {
        require(msg.sender == controller, "!controller");//knownsec//  仅控制器调用
        require(want != address(_asset), "want");
        require(crv != address(_asset), "crv");
        require(ydai != address(_asset), "ydai");
        require(dai != address(_asset), "dai");
        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){//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);//knownsec//  本合约所有 yCRV 转给对应 vault
    }

    function _withdrawAll() internal {
        Gauge(pool).withdraw(Gauge(pool).balanceOf(address(this)));
    }

    function harvest() public {
        require(!Address.isContract(msg.sender),"!contract");
        Mintr(mintr).mint(pool);

        doswap();
        dosplit();
        deposit();
    }
    function doswap() internal {
        //knownsec// 建议添加校验本合约 output 余额大于 0,再执行以下 swap
        uint256 _2token = IERC20(output).balanceOf(address(this)).mul(90).div(100); //90%
        uint256 _2yfii = IERC20(output).balanceOf(address(this)).mul(10).div(100);   //10%
        UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this),
now.add(1800));
        UniswapRouter(unirouter).swapExactTokensForTokens(_2yfii, 0, swap2YFIIRouting, address(this),
now.add(1800));

        uint _dai = IERC20(dai).balanceOf(address(this));
        if (_dai > 0) {
            yERC20(ydai).deposit(_dai);//knownsec// DAI 转换为 yDAI
        }
        uint _ydai = IERC20(ydai).balanceOf(address(this));
        if (_ydai > 0) {
            ICurveFi(curve).add_liquidity([_ydai,0,0,0],0);//knownsec// 为 curve 添加流动性 yDAI
        }

    }
    function dosplit() internal{//knownsec// 分发 yfii
        uint b = IERC20(yfii).balanceOf(address(this));
        uint _fee = b.mul(fee).div(max);//knownsec// 本合约 YFII 量 * fee / max
        uint _callfee = b.mul(callfee).div(max);//knownsec// 本合约 YFII 量 * callfee / max
        uint _burnfee = b.mul(burnfee).div(max);//knownsec// 本合约 YFII 量 * burnfee / max
        IERC20(yfii).safeTransfer(Controller(controller).rewards(), _fee); //4%   3% team +1% insurance
        IERC20(yfii).safeTransfer(msg.sender, _callfee); //call fee 1%
        IERC20(yfii).safeTransfer(burnAddress, _burnfee); //burn fee 5%

        if (strategyfee >0){
            uint _strategyfee = b.mul(strategyfee).div(max);
            IERC20(yfii).safeTransfer(strategyDev, _strategyfee);
        }
    }

    function _withdrawSome(uint256 _amount) internal returns (uint) {
        Gauge(pool).withdraw(_amount);
        return _amount;
    }

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

    function balanceOfPool() public view returns (uint) {
        return Gauge(pool).balanceOf(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 setStrategyFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        strategyfee = _fee;
    }
    function setCallFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        callfee = _fee;
    }
    function setBurnFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        burnfee = _fee;
    }
    function setBurnAddress(address _burnAddress) public{
        require(msg.sender == governance, "!governance");
        burnAddress = _burnAddress;
    }

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

```
            withdrawalFee = _withdrawalFee;
        }
}
```

iVaultYCRV.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-04
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-13
*/

pragma solidity ^0.5.16;//knownsec// 指定编译器版本

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 {//knownsec// 增发代币
            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;
}

contract iVault is ERC20, ERC20Detailed {//knownsec// iVualt 合约,继承自 ERC20、ERC20Detailed
        using SafeERC20 for IERC20;
        using Address for address;
        using SafeMath for uint256;

        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;

        constructor (address _token,uint _earnLowerlimit) public ERC20Detailed(
            string(abi.encodePacked("yfii ", ERC20Detailed(_token).name())),
            string(abi.encodePacked("i", ERC20Detailed(_token).symbol())),//knownsec// iyCRV
            ERC20Detailed(_token).decimals()
        ) {
            token = IERC20(_token);
            governance = tx.origin;
            controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
            earnLowerlimit = _earnLowerlimit;
        }

        function balance() public view returns (uint) {//knownsec//  本合约yCRV 量 +  控制器yCRV 量
            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) {//knownsec//  可用量
            return token.balanceOf(address(this)).mul(min).div(max);
        }

        function earn() public {//knownsec//  赚取利息
            uint _bal = available();
            token.safeTransfer(controller, _bal);
            Controller(controller).earn(address(token), _bal);
        }

        function depositAll() external {//knownsec//  存入所有
        deposit(token.balanceOf(msg.sender));
    }

    function deposit(uint _amount) public {//knownsec// 存入 yCRV
        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 {//knownsec// 提现所有
        withdraw(balanceOf(msg.sender));
    }



    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {//knownsec// iyCRV 提现为 yCRV
        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);
    }

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

StrategyDForceDAI.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-06
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-13
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.17;//knownsec// 指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
    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 {//knownsec// 安全算数库
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 {//knownsec// 控制器接口
        function vaults(address) external view returns (address);
        function rewards() external view returns (address);
}

/*

  A strategy must implement the following calls;

  - deposit()
  - withdraw(address) must exclude any tokens used in the yield - Controller role - withdraw should return to
Controller
  - withdraw(uint) - Controller | Vault role - withdraw should always return to vault
  - withdrawAll() - Controller | Vault role - withdraw should always return to vault
  - balanceOf()

  Where possible, strategies must remain as immutable as possible, instead of updating variables, we update the
contract by linking it in the controller

*/

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//
unipool dDAI/dForce
        address constant public df = address(0x431ad2ff6a9C365805eBaD47Ee021148d6f7DBe0);//knownsec//
dForce
        address constant public output = address(0x431ad2ff6a9C365805eBaD47Ee021148d6f7DBe0);//knownsec//
dForce
        address constant public unirouter =
address(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);//knownsec// UniswapV2Router02
        address constant public weth = address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2); // used for
df <> weth <> usdc route

        address constant public yfii = address(0xa1d0E215a23d7030842FC67cE582a6aFa3CCaB83);


        uint public strategyfee = 0;//knownsec// 0%
        uint public fee = 400;//knownsec// 40% fee/max
        uint public burnfee = 500;//knownsec// 50% burnfee/max
        uint public callfee = 100;//knownsec// 10% callfee/max
        uint constant public max = 1000;

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

        address public governance;
        address public strategyDev;
        address public controller;
        address public burnAddress = 0xB6af2DabCEBC7d30E440714A33E5BD45CEEd103a;

        string public getName;

        address[] public swap2YFIIRouting;
        address[] public swap2TokenRouting;


        constructor() public {
                governance = msg.sender;//knownsec// 设置部署者为治理地址
        controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
        getName = string(
            abi.encodePacked("yfii:Strategy:",
                abi.encodePacked(IERC20(want).name(),"DF Token"
                )
            ));
        swap2YFIIRouting = [output,weth,yfii];//knownsec//  df <> weth <> yfii
        swap2TokenRouting = [output,weth,want];//knownsec//  df <> weth <> DAI
        doApprove();
        strategyDev = tx.origin;
    }

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


    function deposit() public {//knownsec// 流动性挖矿 DAI->dDAI->质押
        uint _want = IERC20(want).balanceOf(address(this));//knownsec// 本合约 DAI 量
        if (_want > 0) {
            IERC20(want).safeApprove(d, 0);
            IERC20(want).safeApprove(d, _want);
            dERC20(d).mint(address(this), _want);//knownsec// 转为等量的 dDAI
        }

        uint _d = IERC20(d).balanceOf(address(this));//knownsec// 本合约 dDAI 量
        if (_d > 0) {
            IERC20(d).safeApprove(pool, 0);
            IERC20(d).safeApprove(pool, _d);
            dRewards(pool).stake(_d);//knownsec// 将 dDAI 质押进 unipool 池中
        }

    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint balance) {//knownsec// 提现某资产 df/yfii/weth
        require(msg.sender == controller, "!controller");//knownsec// 仅控制器调用
        require(want != address(_asset), "want");//knownsec// 校验不为 DAI
        require(d != address(_asset), "d");//knownsec// 校验不为 dDAI
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);//knownsec// 提现至控制器
    }

    // 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){//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);//knownsec// 本合约所有 DAI 转给对应 vault
    }

    function _withdrawAll() internal {
        dRewards(pool).exit();//knownsec// 退出流动池
        uint _d = IERC20(d).balanceOf(address(this));
        if (_d > 0) {
            dERC20(d).redeem(address(this),_d);//knownsec// dDAI 赎回 DAI
        }
    }

    function harvest() public {
        require(!Address.isContract(msg.sender),"!contract");
        dRewards(pool).getReward();//knownsec// 提取流动池奖励 df

        doswap();
        dosplit();//分 yfii
        deposit();//knownsec// 收益 df 转为 DAI 后继续流动性挖矿
    }

    function doswap() internal {
        //knownsec// 建议添加校验本合约 output 余额大于 0,再执行以下 swap
        uint256 _2token = IERC20(output).balanceOf(address(this)).mul(90).div(100); //90%
        uint256 _2yfii = IERC20(output).balanceOf(address(this)).mul(10).div(100);   //10%
        UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this), 
now.add(1800));//knownsec// df 收益 90%转为 DAI
        UniswapRouter(unirouter).swapExactTokensForTokens(_2yfii, 0, swap2YFIIRouting, address(this), 
now.add(1800));//knownsec// df 收益 10%转为 yfii 用于分发奖励
    }
    function dosplit() internal{//knownsec// 分发 yfii
        uint b = IERC20(yfii).balanceOf(address(this));//knownsec// 本合约 YFII 量
        uint _fee = b.mul(fee).div(max);//knownsec// 本合约 YFII 量 * fee / max
        uint _callfee = b.mul(callfee).div(max);//knownsec// 本合约 YFII 量 * callfee / max
        uint _burnfee = b.mul(burnfee).div(max);//knownsec// 本合约 YFII 量 * burnfee / max
        IERC20(yfii).safeTransfer(Controller(controller).rewards(), _fee); //4%   3% team +1% insurance
        IERC20(yfii).safeTransfer(msg.sender, _callfee); //call fee 1%
        IERC20(yfii).safeTransfer(burnAddress, _burnfee); //burn fee 5%

        if (strategyfee >0){
            uint _strategyfee = b.mul(strategyfee).div(max);
            IERC20(yfii).safeTransfer(strategyDev, _strategyfee);
        }
    }

    function _withdrawSome(uint256 _amount) internal returns (uint) {//knownsec// 从 unipool 提现 dDAI,仅内
部调用
        uint _d = _amount.mul(1e18).div(dERC20(d).getExchangeRate());//knownsec// _d =  提现数 * 1e18 / 
dDAI 汇率
        uint _before = IERC20(d).balanceOf(address(this));//knownsec// 提现前本合约 dDAI 量
        dRewards(pool).withdraw(_d);//knownsec// unipool 中 _d 数量的 dDAI 提现到本合约
        uint _after = IERC20(d).balanceOf(address(this));//knownsec// 提现后本合约 dDAI 量
        uint _withdrew = _after.sub(_before);//knownsec// 从 unipool 提现前后的差值
        _before = IERC20(want).balanceOf(address(this));//knownsec// 赎回前本合约 DAI 量
        dERC20(d).redeem(address(this), _withdrew);//knownsec// dDAI 赎回 DAI 到本合约
        _after = IERC20(want).balanceOf(address(this));//knownsec// 赎回后本合约 DAI 量
        _withdrew = _after.sub(_before);//knownsec// 赎回前后的差值为提现额
        return _withdrew;
    }

    function balanceOfWant() public view returns (uint) {//knownsec// 本合约 DAI 量
        return IERC20(want).balanceOf(address(this));
    }

    function balanceOfPool() public view returns (uint) {//knownsec// 本合约在 unipool 的 dDAI 质押量 * dDAI 
汇率 / 1e18
        return (dRewards(pool).balanceOf(address(this))).mul(dERC20(d).getExchangeRate()).div(1e18);
    }

    function getExchangeRate() public view returns (uint) {//knownsec// dDAI 汇率
        return dERC20(d).getExchangeRate();
    }

    function balanceOfD() public view returns (uint) {//knownsec// 本合约 dDAI 量
        return dERC20(d).getTokenBalance(address(this));
    }

    function balanceOf() public view returns (uint) {//knownsec// 本合约 DAI 量 + 本合约 dDAI 量 + unipool 
质押量
        return balanceOfWant()
               .add(balanceOfD())
               .add(balanceOfPool());
    }

    function setGovernance(address _governance) external {//knownsec// 设置治理地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setController(address _controller) external {//knownsec// 设置控制器,仅治理地址调用
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }
    function setFee(uint256 _fee) external{//knownsec// 设置手续费,仅治理地址调用
        require(msg.sender == governance, "!governance");
        fee = _fee;
    }
    function setStrategyFee(uint256 _fee) external{//knownsec// 设置策略手续费,仅治理地址调用
        require(msg.sender == governance, "!governance");
        strategyfee = _fee;
    }
    function setCallFee(uint256 _fee) external{//knownsec// 设置调用手续费,仅治理地址调用
        require(msg.sender == governance, "!governance");
        callfee = _fee;
    }
    function setBurnFee(uint256 _fee) external{//knownsec// 设置销毁手续费,仅治理地址调用
        require(msg.sender == governance, "!governance");
        burnfee = _fee;
    }
    function setBurnAddress(address _burnAddress) public{//knownsec// 设置销毁地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        burnAddress = _burnAddress;
    }

    function setWithdrawalFee(uint _withdrawalFee) external {//knownsec// 设置提现手续费,仅治理地址调用
        require(msg.sender == governance, "!governance");
        require(_withdrawalFee <=100,"fee >= 1%"); //max:1%
        withdrawalFee = _withdrawalFee;
    }
}
```

iVaultDAI.sol

```
/**
  *Submitted for verification at Etherscan.io on 2020-09-04
*/

/**
  *Submitted for verification at Etherscan.io on 2020-08-13
*/

pragma solidity ^0.5.16;//knownsec// 指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
    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 {//knownsec// 上下文属性
    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 {//knownsec// 所有权合约,继承自 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 {//knownsec// ERC20 代币标准实现,继承自 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 {//knownsec// 增发代币
        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 {//knownsec// ERC20 代币补充信息,继承自 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 {//knownsec// 安全算数库
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 {//knownsec// 控制器接口
    function withdraw(address, uint) external;
    function balanceOf(address) external view returns (uint);
    function earn(address, uint) external;
}

contract iVault is ERC20, ERC20Detailed {//knownsec// iVualt 合约,继承自 ERC20、ERC20Detailed
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;

    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;

    constructor (address _token,uint _earnLowerlimit) public ERC20Detailed(
        string(abi.encodePacked("yfii ", ERC20Detailed(_token).name())),
        string(abi.encodePacked("i", ERC20Detailed(_token).symbol())),//knownsec// iDAI
        ERC20Detailed(_token).decimals()
    ) {
        token = IERC20(_token);
        governance = tx.origin;
        controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
        earnLowerlimit = _earnLowerlimit;
    }

    function balance() public view returns (uint) {//knownsec// 本合约 DAI 量 + 控制器 DAI 量
        return token.balanceOf(address(this))
                .add(Controller(controller).balanceOf(address(token)));
    }

    function setMin(uint _min) external {//knownsec// 设置可借贷量比率分子,仅治理地址调用
        require(msg.sender == governance, "!governance");
        min = _min;
    }

    function setGovernance(address _governance) public {//knownsec// 设置治理地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setController(address _controller) public {//knownsec// 设置控制器,仅治理地址调用
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }
    function setEarnLowerlimit(uint256 _earnLowerlimit) public{//knownsec// 设置自动 earn 线,仅治理地址调
用
        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) {//knownsec// 可用量
        return token.balanceOf(address(this)).mul(min).div(max);//knownsec// 本合约 DAI 量 * min / max
    }

    function earn() public {//knownsec// 赚取利息
        uint _bal = available();
        token.safeTransfer(controller, _bal);//knownsec// 将可借贷量的 DAI 转至控制器
        Controller(controller).earn(address(token), _bal);//knownsec// 调用控制器 earn 方法
    }

    function depositAll() external {//knownsec// 存入调用者所有 DAI
        deposit(token.balanceOf(msg.sender));
    }

    function deposit(uint _amount) public {//knownsec// 存入 DAI
        uint _pool = balance();//knownsec// DAI 总量
        uint _before = token.balanceOf(address(this));//knownsec// 本合约 DAI 量
        token.safeTransferFrom(msg.sender, address(this), _amount);//knownsec// 将调用者 DAI 转入本合约
        uint _after = token.balanceOf(address(this));//knownsec// 存款转入后本合约 DAI 量
        _amount = _after.sub(_before); // Additional check for deflationary tokens //knownsec// 更新存入额为
本合约转账前后差值
        uint shares = 0;//knownsec// 相应的 iDAI 量
        if (totalSupply() == 0) {//knownsec// 若 iDAI 量为 0
            shares = _amount;
        } else {
            shares = (_amount.mul(totalSupply())).div(_pool);//knownsec// 存款 DAI 量_amount * iDAI 总量
/ DAI 总量
        }
        _mint(msg.sender, shares);//knownsec// 存款者获取 shares 量的 iDAI 代币
        if (token.balanceOf(address(this))>earnLowerlimit){//knownsec// 若本合约 DAI 量超过最低线,调用
earn
            earn();
        }
    }

    function withdrawAll() external {//knownsec// 提现所有 iDAI 为 DAI
        withdraw(balanceOf(msg.sender));
    }



    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {//knownsec// iDAI 提现为 DAI
        uint r = (balance().mul(_shares)).div(totalSupply());//knownsec// r = DAI 总量 * iDAI 提取额 / iDAI 总
量
        _burn(msg.sender, _shares);//knownsec// 销毁相应 iDAI 量

        // Check balance
        uint b = token.balanceOf(address(this));//knownsec// 本合约 DAI 量
        if (b < r) {//knownsec// 提现量 > 本合约 DAI 量
            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));//knownsec// 转入后的本合约 DAI 量
            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
            }
        }

        token.safeTransfer(msg.sender, r);//knownsec// 转出
    }

    function getPricePerFullShare() public view returns (uint) {//knownsec// DAI/iDAI 汇率
        return balance().mul(1e18).div(totalSupply());//knownsec// DAI 总量 * 1e18 / iDAI 总量
    }
}
```

StrategyTUSDCurve.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-07
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-29
*/

// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.5.17;

interface IERC20 {//knownsec// 指定编译器版本
	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 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 {//knownsec// ERC20 代币标准接口
	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 {//knownsec// OpenZeppelin Address 库
	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 {//knownsec// OpenZeppelin SafeERC20 库
	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 {//knownsec// 控制器接口
    function vaults(address) external view returns (address);
    function rewards() external view returns (address);
}

/*

 A strategy must implement the following calls;

 - deposit()
 - withdraw(address) must exclude any tokens used in the yield - Controller role - withdraw should return to Controller
 - withdraw(uint) - Controller | Vault role - withdraw should always return to vault
 - withdrawAll() - Controller | Vault role - withdraw should always return to vault
 - balanceOf()

 Where possible, strategies must remain as immutable as possible, instead of updating variables, we update the contract by linking it in the controller

*/

interface yERC20 {
  function deposit(uint) external;
  function withdraw(uint) external;
  function getPricePerFullShare() external view returns (uint);
}

interface ICurveFi {

  function get_virtual_price() external view returns (uint);
  function add_liquidity(
    uint256[4] calldata amounts,
    uint256 min_mint_amount
  ) external;
  function remove_liquidity_imbalance(
    uint256[4] calldata amounts,
    uint256 max_burn_amount
  ) external;
  function remove_liquidity(
    uint256 _amount,
    uint256[4] calldata amounts
  ) external;
  function exchange(
    int128 from, int128 to, uint256 _from_amount, uint256 _min_to_amount
  ) external;
}

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

    address constant public want = address(0x0000000000085d4780B73119b644AE5ecd22b376);//knownsec// TUSD
    address constant public y = address(0x73a052500105205d34Daf004eAb301916DA8190f);//knownsec// yTUSD
    address constant public ycrv = address(0xdF5e0e81Dff6FAF3A7e52BA697820c5e32D806A8);//knownsec// yCRV
    address constant public iycrv = address(0x3E3db9cc5b540d2794DB3861BE5A4887cF77E48B);//knownsec// iyCRV
    address constant public curve = address(0x45F783CCE6B7FF23B2ab2D70e416cdb7D6055f51);//knownsec// y Swap
    address constant public dai = address(0x6B175474E89094C44Da98b954EedeAC495271d0F);//knownsec//
DAI
    address constant public ydai = address(0x16de59092dAE5CcF4A1E6439D611fd0653f0Bd01);//knownsec//
yDAI

    address constant public usdc = address(0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48);//knownsec//
USDC
    address constant public yusdc = address(0xd6aD7a6750A7593E092a9B218d66C0A814a3436e);//knownsec//
yUSDC

    address constant public usdt = address(0xdAC17F958D2ee523a2206206994597C13D831ec7);//knownsec//
USDT
    address constant public yusdt = address(0x83f798e925BcD4017Eb265844FDDAbb448f1707D);//knownsec//
yUSDT

    address constant public tusd = address(0x0000000000085d4780B73119b644AE5ecd22b376);//knownsec//
TUSD
    address constant public ytusd = address(0x73a052500105205d34Daf004eAb301916DA8190f);//knownsec//
yTUSD


    address public governance;
    address public controller;

    constructor() public {
        governance = tx.origin;//knownsec// 治理地址为最初的调用者地址,部署时需注意
        controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
    }

    function getName() external pure returns (string memory) {
        return "yfii:StrategyTUSD:Curve";
    }

    function deposit() public {
        uint _want = IERC20(want).balanceOf(address(this));//knownsec// 本合约 TUSD 量
        if (_want > 0) {
            IERC20(want).safeApprove(y, 0);
            IERC20(want).safeApprove(y, _want);//knownsec// 授权给 yTUSD
            yERC20(y).deposit(_want);//knownsec// 将 TUSD 转换为 yTUSD
        }
        uint _y = IERC20(y).balanceOf(address(this);//knownsec// 本合约的 yTUSD 量
        if (_y > 0) {
            IERC20(y).safeApprove(curve, 0);
            IERC20(y).safeApprove(curve, _y);//knownsec// 授权给 curve
            ICurveFi(curve).add_liquidity([0,0,0,_y],0);//knownsec// 添加流动性,质押挖矿
        }
        uint _ycrv = IERC20(ycrv).balanceOf(address(this));//knownsec// 本合约 yCRV 量
        if (_ycrv > 0) {
            IERC20(ycrv).safeApprove(iycrv, 0);
            IERC20(ycrv).safeApprove(iycrv, _ycrv);//knownsec// 授权给 iyCRV
            yERC20(iycrv).deposit(_ycrv);//knownsec// 将 yCRV 转换为 iyCRV,yCRV 将再次投入策略池
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint balance) {
        require(msg.sender == controller, "!controller");//knownsec// 仅控制器调用
        require(want != address(_asset), "want");
        require(y != address(_asset), "y");
        require(ycrv != address(_asset), "ycrv");
        require(iycrv != address(_asset), "iycrv");
        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);
        }

        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);
    }

    // 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 withdrawTUSD(uint256 _amount) internal returns (uint) {
        IERC20(ycrv).safeApprove(curve, 0);
        IERC20(ycrv).safeApprove(curve, _amount);//knownsec// 授权给 curve
        ICurveFi(curve).remove_liquidity(_amount, [uint256(0),0,0,0]);

        uint256 _ydai = IERC20(ydai).balanceOf(address(this));
        uint256 _yusdc = IERC20(yusdc).balanceOf(address(this));
        uint256 _yusdt = IERC20(yusdt).balanceOf(address(this));

        if (_ydai > 0) {
            IERC20(ydai).safeApprove(curve, 0);
            IERC20(ydai).safeApprove(curve, _ydai);
            ICurveFi(curve).exchange(0, 3, _ydai, 0);
        }
        if (_yusdc > 0) {
            IERC20(yusdc).safeApprove(curve, 0);
            IERC20(yusdc).safeApprove(curve, _yusdc);
            ICurveFi(curve).exchange(1, 3, _yusdc, 0);
        }
        if (_yusdt > 0) {
            IERC20(yusdt).safeApprove(curve, 0);
            IERC20(yusdt).safeApprove(curve, _yusdt);
            ICurveFi(curve).exchange(2, 3, _yusdt, 0);
        }

        uint _before = IERC20(want).balanceOf(address(this));//knownsec// 提现前本合约 TUSD 量
        yERC20(ytusd).withdraw(IERC20(ytusd).balanceOf(address(this)));//knownsec// 提现 yTUSD 为 TUSD
        uint _after = IERC20(want).balanceOf(address(this));//knownsec// 提现后本合约 TUSD 量

        return _after.sub(_before);//knownsec// 实际差值
    }

    function _withdrawAll() internal {
        uint _yycrv = IERC20(iycrv).balanceOf(address(this));
        if (_yycrv > 0) {
            yERC20(iycrv).withdraw(_yycrv);//knownesc// 提现所有 iyCRV 为 yCRV
            withdrawTUSD(IERC20(ycrv).balanceOf(address(this)));//knownsec// 提现 yCRV 数量的 TUSD
        }
    }

    function _withdrawSome(uint256 _amount) internal returns (uint) {
        // calculate amount of ycrv to withdraw for amount of _want_
        uint _ycrv = _amount.mul(1e18).div(ICurveFi(curve).get_virtual_price());
        // calculate amount of iycrv to withdraw for amount of _ycrv_
        uint _yycrv = _ycrv.mul(1e18).div(yERC20(iycrv).getPricePerFullShare());
        uint _before = IERC20(ycrv).balanceOf(address(this));
        yERC20(iycrv).withdraw(_yycrv);
        uint _after = IERC20(ycrv).balanceOf(address(this));
        return withdrawTUSD(_after.sub(_before));
    }

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

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

    function balanceOfYYCRVinYCRV() public view returns (uint) {
        return balanceOfYYCRV().mul(yERC20(iycrv).getPricePerFullShare()).div(1e18);
    }

    function balanceOfYYCRVinyTUSD() public view returns (uint) {
        return balanceOfYYCRVinYCRV().mul(ICurveFi(curve).get_virtual_price()).div(1e18);
    }

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

    function balanceOfYCRVyTUSD() public view returns (uint) {
        return balanceOfYCRV().mul(ICurveFi(curve).get_virtual_price()).div(1e18);
    }

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

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

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

iVaultTUSD.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-07
*/

/**
 *Submitted for verification at Etherscan.io on 2020-09-04
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-13
*/

pragma solidity ^0.5.16;//knownsec// 指定编译器版本

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 {//knownsec// 增发代币
        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;
}

contract iVault is ERC20, ERC20Detailed {//knownsec// iVualt 合约,继承自 ERC20、ERC20Detailed
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;

    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;

    constructor (address _token,uint _earnLowerlimit) public ERC20Detailed(
        string(abi.encodePacked("yfii ", ERC20Detailed(_token).name())),
        string(abi.encodePacked("i", ERC20Detailed(_token).symbol())),//knownsec// iTUSD
        ERC20Detailed(_token).decimals()
    ) {
        token = IERC20(_token);
        governance = tx.origin;
        controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
        earnLowerlimit = _earnLowerlimit;
    }

    function balance() public view returns (uint) {//knownsec// 本合约 TUSD 量 + 控制器 TUSD 量
        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) {//knownsec// 可用量
```

```
        return token.balanceOf(address(this)).mul(min).div(max);
    }

    function earn() public {//knownsec// 赚取利息
        uint _bal = available();
        token.safeTransfer(controller, _bal);
        Controller(controller).earn(address(token), _bal);
    }

    function depositAll() external {//knownsec// 存入调用者所有 DAI
        deposit(token.balanceOf(msg.sender));
    }

    function deposit(uint _amount) public {//knownsec// 存入 TUSD
        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 {//knownsec// 提现所有 iTUSD 为 TUSD
        withdraw(balanceOf(msg.sender));
    }



    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {//knownsec// iTUSD 提现为 TUSD
        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);
    }

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

StrategyFortubeUSDC.sol

```
/**
  *Submitted for verification at Etherscan.io on 2020-09-11
*/

/**
  *Submitted for verification at Etherscan.io on 2020-08-13
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.17;//knownsec// 指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
    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 {//knownsec// 安全算数库
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 {//knownsec// 控制器接口
      function vaults(address) external view returns (address);
      function rewards() external view returns (address);
}

/*

 A strategy must implement the following calls;

 - deposit()
 - withdraw(address) must exclude any tokens used in the yield - Controller role - withdraw should return to
Controller
 - withdraw(uint) - Controller | Vault role - withdraw should always return to vault
 - withdrawAll() - Controller | Vault role - withdraw should always return to vault
 - balanceOf()

 Where possible, strategies must remain as immutable as possible, instead of updating variables, we update the
contract by linking it in the controller

*/



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(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 yfii = address(0xa1d0E215a23d7030842FC67cE582a6aFa3CCaB83);


      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 strategyfee = 100;//knownsec// 10% strategyfee/max
      uint public fee = 300;//knownsec// 30% fee/max
      uint public burnfee = 500;//knownsec// 50% burnfee/max
      uint public callfee = 100;//knownsec// 10% callfee/max
      uint constant public max = 1000;

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

      address public governance;
      address public strategyDev;
      address public controller;
      address public burnAddress = 0xB6af2DabCEBC7d30E440714A33E5BD45CEEd103a;

      string public getName;

      address[] public swap2YFIIRouting;
      address[] public swap2TokenRouting;


      constructor() public {
              governance = msg.sender;//knownsec//  设置部署者为治理地址
              controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
              getName = string(
                      abi.encodePacked("yfii:Strategy:",
                              abi.encodePacked(IERC20(want).name(),"The Force Token"
                              )
                      ));
              swap2YFIIRouting = [output,weth,yfii];//knownsec// for <> weth <> yfii
              swap2TokenRouting = [output,weth,want];//knownsec// for <> weth <> USDC
              doApprove();
              strategyDev = tx.origin;
      }

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


      function deposit() public {//knownsec//  流动性挖矿
              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) {//knownsec//  提现某资产 for/weth
              require(msg.sender == controller, "!controller");//knownsec//  仅控制器调用
              require(want != address(_asset), "want");//knownsec//  校验不为 USDC
              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){//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);//knownsec//  本合约所有 USDC 转给对应 vault
        }

        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");
            ForReward(fortube_reward).claimReward();
            doswap();
            dosplit();//分 yfii
            deposit();
        }

        function doswap() internal {
            //knownsec//  建议添加校验本合约 output 余额大于 0,再执行以下 swap
            uint256 _2token = IERC20(output).balanceOf(address(this)).mul(90).div(100); //90%
            uint256 _2yfii = IERC20(output).balanceOf(address(this)).mul(10).div(100);   //10%
            UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this),
      now.add(1800));
            UniswapRouter(unirouter).swapExactTokensForTokens(_2yfii, 0, swap2YFIIRouting, address(this),
      now.add(1800));
        }
        function dosplit() internal{//knownsec//  分发 yfii
            uint b = IERC20(yfii).balanceOf(address(this));
            uint _fee = b.mul(fee).div(max);//knownsec//  本合约 YFII 量 * fee / max
            uint _callfee = b.mul(callfee).div(max);//knownsec//  本合约 YFII 量 * callfee / max
            uint _burnfee = b.mul(burnfee).div(max);//knownsec//  本合约 YFII 量 * burnfee / max
            IERC20(yfii).safeTransfer(Controller(controller).rewards(), _fee); //3%   3% team
            IERC20(yfii).safeTransfer(msg.sender, _callfee); //call fee 1%
            IERC20(yfii).safeTransfer(burnAddress, _burnfee); //burn fee 5%

            if (strategyfee >0){
                uint _strategyfee = b.mul(strategyfee).div(max); //1%
                IERC20(yfii).safeTransfer(strategyDev, _strategyfee);
            }
        }

        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 setStrategyFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            strategyfee = _fee;
        }
        function setCallFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            callfee = _fee;
        }
        function setBurnFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            burnfee = _fee;
        }
        function setBurnAddress(address _burnAddress) public{
            require(msg.sender == governance, "!governance");
            burnAddress = _burnAddress;
        }

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

iVaultUSDC.sol

```
/**
  *Submitted for verification at Etherscan.io on 2020-09-07
*/

/**
  *Submitted for verification at Etherscan.io on 2020-09-04
*/

/**
  *Submitted for verification at Etherscan.io on 2020-08-13
*/

pragma solidity ^0.5.16;//knownsec// 指定编译器版本

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 {//knownsec// 增发代币
                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;
}

contract iVault is ERC20, ERC20Detailed {//knownsec// iVualt 合约,继承自 ERC20、ERC20Detailed
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;

    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;

    constructor (address _token,uint _earnLowerlimit) public ERC20Detailed(
        string(abi.encodePacked("yfii ", ERC20Detailed(_token).name())),
        string(abi.encodePacked("i", ERC20Detailed(_token).symbol())),//knownsec// iUSDC
        ERC20Detailed(_token).decimals()
    ) {
        token = IERC20(_token);
        governance = tx.origin;
        controller = 0x8C2a19108d8F6aEC72867E9cfb1bF517601b515f;
        earnLowerlimit = _earnLowerlimit;
    }

    function balance() public view returns (uint) {//knownsec//  本合约 USDC +  控制器 USDC 量
        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) {//knownsec// 可用量
        return token.balanceOf(address(this)).mul(min).div(max);
    }

    function earn() public {//knownsec// 赚取利息
        uint _bal = available();
        token.safeTransfer(controller, _bal);
        Controller(controller).earn(address(token), _bal);
    }

    function depositAll() external {//knownsec// 存入调用者所有 USDC
        deposit(token.balanceOf(msg.sender));
    }

    function deposit(uint _amount) public {//knownsec// 存入 USDC
        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 {//knownsec// 提取所有
        withdraw(balanceOf(msg.sender));
    }



    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {//knownsec// iUSDC 提取为 USDC
        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);
    }

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

StrategyFortubeETH.sol

```
/**
  *Submitted for verification at Etherscan.io on 2020-09-12
*/

/**
  *Submitted for verification at Etherscan.io on 2020-08-13
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.17;//knownsec// 指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
    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 {//knownsec// 安全算数库
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 {//knownsec// 控制器接口
        function vaults(address) external view returns (address);
        function rewards() external view returns (address);
}

/*

  A strategy must implement the following calls;

  - deposit()
  - withdraw(address) must exclude any tokens used in the yield - Controller role - withdraw should return to Controller
  - withdraw(uint) - Controller | Vault role - withdraw should always return to vault
  - withdrawAll() - Controller | Vault role - withdraw should always return to vault
  - balanceOf()

  Where possible, strategies must remain as immutable as possible, instead of updating variables, we update the contract by linking it in the controller

*/



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;
}

interface WETH {
        function deposit() external payable;
        function withdraw(uint wad) external;
        event Deposit(address indexed dst, uint wad);
        event Withdrawal(address indexed src, uint wad);
}

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

        address constant public eth_address = address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
        address constant public want = address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2); //eth //knownsec// weth
        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 yfii = address(0xa1d0E215a23d7030842FC67cE582a6aFa3CCaB83);


    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 strategyfee = 100;//knownsec// 10% strategyfee/max
    uint public fee = 300;//knownsec// 30% fee/max
    uint public burnfee = 500;//knownsec// 50% burnfee/max
    uint public callfee = 100;//knownsec// 10% callfee/max
    uint constant public max = 1000;

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

    address public governance;
    address public strategyDev;
    address public controller;
    address public burnAddress = 0xB6af2DabCEBC7d30E440714A33E5BD45CEEd103a;

    string public getName;

    address[] public swap2YFIIRouting;
    address[] public swap2TokenRouting;


    constructor() public {
        governance = msg.sender;//knownsec//  设置部署者为治理地址
        controller = 0xcDCf1f9Ac816Fed665B09a00f60c885dd8848b02;
        getName = string(
            abi.encodePacked("yfii:Strategy:",
                abi.encodePacked(IERC20(want).name(),"The Force Token"
                )
            ));
        swap2YFIIRouting = [output,weth,yfii];//knownsec// for <> weth <> yfii
        swap2TokenRouting = [output,weth];//for->weth
        doApprove();
        strategyDev = tx.origin;
    }

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



    function () external payable {
    }

    function deposit() public {//knownsec//  流动性挖矿
        uint _want = IERC20(want).balanceOf(address(this));
        address _controller = For(fortube).controller();
        if (_want > 0) {
            WETH(address(weth)).withdraw(_want); //weth->eth
            For(fortube).deposit.value(_want)(eth_address,_want);
        }

    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint balance) {//knownsec//  提现某资产 for/weth
        require(msg.sender == controller, "!controller");//knownsec//  仅控制器调用
        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){//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 || 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);//knownsec//  本合约所有 WETH 转给对应 vault
    }

    function _withdrawAll() internal {
        address _controller = For(fortube).controller();
        IFToken fToken = IFToken(IBankController(_controller).getFTokeAddress(eth_address));
        uint b = fToken.calcBalanceOfUnderlying(address(this));
        _withdrawSome(b);
    }

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

    function doswap() internal {
        //knownsec//  建议添加校验本合约 output 余额大于 0,再执行以下 swap
        uint256 _2token = IERC20(output).balanceOf(address(this)).mul(90).div(100); //90%
        uint256 _2yfii = IERC20(output).balanceOf(address(this)).mul(10).div(100);    //10%
        UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this), 
now.add(1800));
        UniswapRouter(unirouter).swapExactTokensForTokens(_2yfii, 0, swap2YFIIRouting, address(this), 
now.add(1800));
    }
    function dosplit() internal{//knownsec//  分发 yfii
        uint b = IERC20(yfii).balanceOf(address(this));
        uint _fee = b.mul(fee).div(max);//knownsec//  本合约 YFII 量  * fee / max
        uint _callfee = b.mul(callfee).div(max);//knownsec//  本合约 YFII 量  * callfee / max
        uint _burnfee = b.mul(burnfee).div(max);//knownsec//  本合约 YFII 量  * burnfee / max
        IERC20(yfii).safeTransfer(Controller(controller).rewards(), _fee); //3%   3% team
        IERC20(yfii).safeTransfer(msg.sender, _callfee); //call fee 1%
        IERC20(yfii).safeTransfer(burnAddress, _burnfee); //burn fee 5%

        if (strategyfee >0){
            uint _strategyfee = b.mul(strategyfee).div(max); //1%
            IERC20(yfii).safeTransfer(strategyDev, _strategyfee);
        }
    }

    function _withdrawSome(uint256 _amount) internal returns (uint) {
        For(fortube).withdrawUnderlying(eth_address,_amount);
        WETH(address(weth)).deposit.value(address(this).balance)();
        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(eth_address));
        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 setStrategyFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            strategyfee = _fee;
        }
        function setCallFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            callfee = _fee;
        }
        function setBurnFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            burnfee = _fee;
        }
        function setBurnAddress(address _burnAddress) public{
            require(msg.sender == governance, "!governance");
            burnAddress = _burnAddress;
        }

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

iVaultETH.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-12
*/

/**
 *Submitted for verification at Etherscan.io on 2020-09-01
*/

pragma solidity ^0.5.16;//knownsec// 指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
        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 {//knownsec// 上下文属性
        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 ERC20 is Context, IERC20 {//knownsec// ERC20 代币标准实现,继承自 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 {//knownsec// 增发代币
        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 {//knownsec// ERC20 代币补充信息,继承自 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 {//knownsec// 安全算数库
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 WETH {//knownsec// WETH 接口
    function deposit() external payable;
    function withdraw(uint wad) external;
    event Deposit(address indexed dst, uint wad);
    event Withdrawal(address indexed src, uint wad);
}

interface Controller {//knownsec// 控制器接口
    function withdraw(address, uint) external;
    function balanceOf(address) external view returns (uint);
    function earn(address, uint) external;
}

contract iVault is ERC20, ERC20Detailed {//knownsec// iVualt 合约,继承自 ERC20、ERC20Detailed
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;

    IERC20 public token;

    uint public min = 9990;
    uint public constant max = 10000;
    uint public earnLowerlimit = 50 ether; //池内空余资金到这个值就自动 earn

    address public governance;
    address public controller;

    constructor (address _token) public ERC20Detailed(
        string(abi.encodePacked("yfii ", ERC20Detailed(_token).name())),
        string(abi.encodePacked("i", ERC20Detailed(_token).symbol())),//knownsec// iWETH
        ERC20Detailed(_token).decimals()
    ) {
        token = IERC20(_token);
        governance = msg.sender;
        controller = 0xcDCf1f9Ac816Fed665B09a00f60c885dd8848b02;
    }

    function balance() public view returns (uint) {//knownsec// 本合约 WETH 量 + 控制器 WETH 量
        return token.balanceOf(address(this))
                .add(Controller(controller).balanceOf(address(token)));
    }

    function setMin(uint _min) external {//knownsec// 设置可借贷量比率分子,仅治理地址调用
        require(msg.sender == governance, "!governance");
        min = _min;
    }

    function setGovernance(address _governance) public {//knownsec// 设置治理地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function setController(address _controller) public {//knownsec// 设置控制器,仅治理地址调用
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }
    function setEarnLowerlimit(uint256 _earnLowerlimit) public{//knownsec// 设置自动 earn 线,仅治理地址调
用
      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) {//knownsec// 可用量
        return token.balanceOf(address(this)).mul(min).div(max);//knownsec// 本合约 WETH 量 * min / max
    }

    function earn() public {//knownsec// 赚取利息
        uint _bal = available();
        token.safeTransfer(controller, _bal);//knownsec// 将可借贷量的 WETH 转至控制器
        Controller(controller).earn(address(token), _bal);//knownsec// 调用控制器 earn 方法
    }

    function depositAll() external {//knownsec// 存入调用者所有 WETH
        deposit(token.balanceOf(msg.sender));
    }

    function deposit(uint _amount) public {//knownsec// 存入 WETH
        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 depositETH() public payable {//knownsec// 直接通过 ETH 存入
        uint _pool = balance();
        uint _before = token.balanceOf(address(this));
        uint _amount = msg.value;
        WETH(address(token)).deposit.value(_amount)();//knownsec// ETH 转为 WETH
        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);
    }

    function withdrawAll() external {//knownsec// 提现所有 iWETH 为 WETH
        withdraw(balanceOf(msg.sender));
    }

    function withdrawAllETH() external {
        withdrawETH(balanceOf(msg.sender));
    }



    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {//knownsec// 提现 iWETH 为 WETH
        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);
    }

    // No rebalance implementation for lower fees and faster swaps
    function withdrawETH(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);
            }
        }
```

```
            WETH(address(token)).withdraw(r);
            address(msg.sender).transfer(r);
        }

        function getPricePerFullShare() public view returns (uint) {//knownsec// WETH/iWETH 汇率
            return balance().mul(1e18).div(totalSupply());
        }

        function () external payable {
            if (msg.sender != address(token)) {
                depositETH();
            }
        }
}
```

StrategyFortubeBUSD.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-13
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-13
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.17;//knownsec// 指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
        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 {//knownsec// 安全算数库
        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 {//knownsec// OpenZeppelin Address 库
	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 {//knownsec// OpenZeppelin SafeERC20 库
	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 {//knownsec// 控制器接口
	function vaults(address) external view returns (address);
	function rewards() external view returns (address);
}
/*

 A strategy must implement the following calls;

 - deposit()
 - withdraw(address) must exclude any tokens used in the yield - Controller role - withdraw should return to Controller
 - withdraw(uint) - Controller | Vault role - withdraw should always return to vault
 - withdrawAll() - Controller | Vault role - withdraw should always return to vault
 - balanceOf()

 Where possible, strategies must remain as immutable as possible, instead of updating variables, we update the contract by linking it in the controller

*/



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(0x4Fabb145d64652a948d72533023f6E7A623C7C53); //usdc
//knownsec// BUSD
    address constant public output = address(0x1FCdcE58959f536621d76f5b7FfB955baa5A672F); //for
    address constant public unirouter =
address(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);//knownsec// UniswapV2Router02
    address constant public weth = address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2); // used for
for <> weth <> usdc route

    address constant public yfii = address(0xa1d0E215a23d7030842FC67cE582a6aFa3CCaB83);


    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 strategyfee = 100;//knownsec// 10% strategyfee/max
    uint public fee = 300;//knownsec// 30% fee/max
    uint public burnfee = 500;//knownsec// 50% burnfee/max
    uint public callfee = 100;//knownsec// 10% callfee/max
    uint constant public max = 1000;

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

    address public governance;
    address public strategyDev;
    address public controller;
    address public burnAddress = 0xB6af2DabCEBC7d30E440714A33E5BD45CEEd103a;

    string public getName;

    address[] public swap2YFIIRouting;
    address[] public swap2TokenRouting;


    constructor() public {
        governance = msg.sender;//knownsec// 设置部署者为治理地址
        controller = 0xcDCf1f9Ac816Fed665B09a00f60c885dd8848b02;
        getName = string(
            abi.encodePacked("yfii:Strategy:",
                abi.encodePacked(IERC20(want).name(),"The Force Token"
                )
            ));
        swap2YFIIRouting = [output,weth,yfii];//knownsec// for <> weth <> yfii
        swap2TokenRouting = [output,weth,want];//knownsec// for <> weth <> BUSD
        doApprove();
        strategyDev = tx.origin;
    }

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


    function deposit() public {//knownsec// 流动性挖矿
        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) {//knownsec// 提现某资产 for/weth
        require(msg.sender == controller, "!controller");//knownsec// 仅控制器调用
        require(want != address(_asset), "want");//knownsec// 校验不为 BUSD
        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){//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 || 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);//knownsec// 本合约所有 BUSD 转给对应 vault
    }

    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");
        ForReward(fortube_reward).claimReward();
        doswap();
        dosplit();//分 yfii
        deposit();
    }

    function doswap() internal {
        //knownsec// 建议添加校验本合约 output 余额大于 0,再执行以下 swap
        uint256 _2token = IERC20(output).balanceOf(address(this)).mul(90).div(100); //90%
        uint256 _2yfii = IERC20(output).balanceOf(address(this)).mul(10).div(100);   //10%
        UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this),
now.add(1800));
        UniswapRouter(unirouter).swapExactTokensForTokens(_2yfii, 0, swap2YFIIRouting, address(this),
now.add(1800));
    }
    function dosplit() internal{//knownsec// 分发 yfii
        uint b = IERC20(yfii).balanceOf(address(this));
        uint _fee = b.mul(fee).div(max);//knownsec// 本合约 YFII 量 * fee / max
        uint _callfee = b.mul(callfee).div(max);//knownsec// 本合约 YFII 量 * callfee / max
        uint _burnfee = b.mul(burnfee).div(max);//knownsec// 本合约 YFII 量 * burnfee / max
        IERC20(yfii).safeTransfer(Controller(controller).rewards(), _fee); //3%   3% team
        IERC20(yfii).safeTransfer(msg.sender, _callfee); //call fee 1%
        IERC20(yfii).safeTransfer(burnAddress, _burnfee); //burn fee 5%

        if (strategyfee >0){
            uint _strategyfee = b.mul(strategyfee).div(max); //1%
            IERC20(yfii).safeTransfer(strategyDev, _strategyfee);
        }
    }

    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 setStrategyFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            strategyfee = _fee;
        }
        function setCallFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            callfee = _fee;
        }
        function setBurnFee(uint256 _fee) external{
            require(msg.sender == governance, "!governance");
            burnfee = _fee;
        }
        function setBurnAddress(address _burnAddress) public{
            require(msg.sender == governance, "!governance");
            burnAddress = _burnAddress;
        }

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

iVaultBUSD.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-13
*/

/**
 *Submitted for verification at Etherscan.io on 2020-09-04
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-13
*/

pragma solidity ^0.5.16;//knownsec// 指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
        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 {//knownsec// 上下文属性
        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 {//knownsec// 所有权合约,继承自 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 {//knownsec// ERC20 代币标准实现,继承自 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 {//knownsec// 增发代币
        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 {//knownsec// ERC20 代币补充信息,继承自 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 {//knownsec// 安全算数库
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 iVault is ERC20, ERC20Detailed {//knownsec// iVualt 合约,继承自 ERC20、ERC20Detailed
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;
    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;

    constructor (address _token,uint _earnLowerlimit) public ERC20Detailed(
        string(abi.encodePacked("yfii ", ERC20Detailed(_token).name())),
        string(abi.encodePacked("i", ERC20Detailed(_token).symbol())),//knownsec// iBUSD
        ERC20Detailed(_token).decimals()
    ) {
        token = IERC20(_token);
        governance = tx.origin;
        controller = 0xcDCf1f9Ac816Fed665B09a00f60c885dd8848b02;
        earnLowerlimit = _earnLowerlimit;
    }

    function balance() public view returns (uint) {//knownsec//  本合约 BUSD 量 + 控制器 iBUSD 量
        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) {//knownsec//  可用量
        return token.balanceOf(address(this)).mul(min).div(max);
    }

    function earn() public {//knownsec//  赚取利息
        uint _bal = available();
        token.safeTransfer(controller, _bal);
        Controller(controller).earn(address(token), _bal);
    }

    function depositAll() external {//knownsec//  存入调用者所有 DAI
        deposit(token.balanceOf(msg.sender));
    }

    function deposit(uint _amount) public {//knownsec//  存入 BUSD
        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 {//knownsec//  提现所有 iBUSD 为 BUSD
        withdraw(balanceOf(msg.sender));
    }



    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {//knownsec// iBUSD 提现为 BUSD
        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);
      }

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

StrategyFortubeHBTC.sol

```
/**
  *Submitted for verification at Etherscan.io on 2020-09-15
*/

/**
  *Submitted for verification at Etherscan.io on 2020-08-13
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.17;//knownsec// 指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口//knownsec// 安全算数库
      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 {//knownsec// 安全算数库
      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 {//knownsec// OpenZeppelin Address 库
        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 {//knownsec// OpenZeppelin SafeERC20 库
        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 {//knownsec//  控制器接口
        function vaults(address) external view returns (address);
        function rewards() external view returns (address);
}

/*

  A strategy must implement the following calls;

  - deposit()
  - withdraw(address) must exclude any tokens used in the yield - Controller role - withdraw should return to
Controller
  - withdraw(uint) - Controller | Vault role - withdraw should always return to vault
  - withdrawAll() - Controller | Vault role - withdraw should always return to vault
  - balanceOf()

  Where possible, strategies must remain as immutable as possible, instead of updating variables, we update the
contract by linking it in the controller

*/



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(0x0316EB71485b0Ab14103307bf65a021042c6d380); //hbtc
    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 yfii = address(0xa1d0E215a23d7030842FC67cE582a6aFa3CCaB83);


    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 strategyfee = 100;//knownsec// 10% strategyfee/max
    uint public fee = 300;//knownsec// 30% fee/max
    uint public burnfee = 500;//knownsec// 50% burnfee/max
    uint public callfee = 100;//knownsec// 10% callfee/max
    uint constant public max = 1000;

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

    address public governance;
    address public strategyDev;
    address public controller;
    address public burnAddress = 0xB6af2DabCEBC7d30E440714A33E5BD45CEEd103a;

    string public getName;

    address[] public swap2YFIIRouting;
    address[] public swap2TokenRouting;


    constructor() public {
        governance = msg.sender;//knownsec// 设置部署者为治理地址
        controller = 0xcDCf1f9Ac816Fed665B09a00f60c885dd8848b02;
        getName = string(
            abi.encodePacked("yfii:Strategy:",
                abi.encodePacked(IERC20(want).name(),"The Force Token"
                )
            ));
        swap2YFIIRouting = [output,weth,yfii];//knownsec// for <> weth <> yfii
        swap2TokenRouting = [output,weth,want];//knownsec// for <> weth <> HBTC
        doApprove();
        strategyDev = tx.origin;
    }

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


    function deposit() public {//knownsec// 流动性挖矿
        uint _want = IERC20(want).balanceOf(address(this));
```

```
            address _controller = For(fortube).controller();
            if (_want > 0) {
                //knownsec// 存在交易顺序依赖风险,不应注释以下语句
                // 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) {//knownsec// 提现某资产 for/weth
            require(msg.sender == controller, "!controller");//knownsec// 仅控制器调用
            require(want != address(_asset), "want");//knownsec// 校验不为 HBTC
            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){//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 || 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);//knownsec// 本合约所有 HBTC 转给对应 vault
        }

        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");
            ForReward(fortube_reward).claimReward();
            doswap();
            dosplit();//分 yfii
            deposit();
        }

        function doswap() internal {
            //knownsec// 建议添加校验本合约 output 余额大于 0,再执行以下 swap
            uint256 _2token = IERC20(output).balanceOf(address(this)).mul(90).div(100); //90%
            uint256 _2yfii = IERC20(output).balanceOf(address(this)).mul(10).div(100);   //10%
            UniswapRouter(unirouter).swapExactTokensForTokens(_2token, 0, swap2TokenRouting, address(this),
now.add(1800));
            UniswapRouter(unirouter).swapExactTokensForTokens(_2yfii,   0,   swap2YFIIRouting,   address(this),
now.add(1800));
        }
        function dosplit() internal{//knownsec// 分发 yfii
            uint b = IERC20(yfii).balanceOf(address(this));
            uint _fee = b.mul(fee).div(max);//knownsec// 本合约 YFII 量 * fee / max
            uint _callfee = b.mul(callfee).div(max);//knownsec// 本合约 YFII 量 * callfee / max
            uint _burnfee = b.mul(burnfee).div(max);//knownsec// 本合约 YFII 量 * burnfee / max
            IERC20(yfii).safeTransfer(Controller(controller).rewards(), _fee); //3%   3% team
            IERC20(yfii).safeTransfer(msg.sender, _callfee); //call fee 1%
            IERC20(yfii).safeTransfer(burnAddress, _burnfee); //burn fee 5%

            if (strategyfee >0){
                uint _strategyfee = b.mul(strategyfee).div(max); //1%
            IERC20(yfii).safeTransfer(strategyDev, _strategyfee);
        }
    }

    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 setStrategyFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        strategyfee = _fee;
    }
    function setCallFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        callfee = _fee;
    }
    function setBurnFee(uint256 _fee) external{
        require(msg.sender == governance, "!governance");
        burnfee = _fee;
    }
    function setBurnAddress(address _burnAddress) public{
        require(msg.sender == governance, "!governance");
        burnAddress = _burnAddress;
    }

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

iVaultHBTC.sol

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-13
*/

/**
 *Submitted for verification at Etherscan.io on 2020-09-04
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-13
*/

pragma solidity ^0.5.16;//knownsec//  指定编译器版本

interface IERC20 {//knownsec// ERC20 代币标准接口
    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 {//knownsec// 上下文属性
    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 {//knownsec// 所有权合约,继承自 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 {//knownsec// ERC20 代币标准实现,继承自 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 {//knownsec// 增发代币
        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 {//knownsec// ERC20 代币补充信息,继承自 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 {//knownsec// 安全算数库
    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 {//knownsec// OpenZeppelin Address 库
    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 {//knownsec// OpenZeppelin SafeERC20 库
    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 iVault is ERC20, ERC20Detailed {//knownsec// iVualt 合约,继承自 ERC20、ERC20Detailed
        using SafeERC20 for IERC20;
        using Address for address;
        using SafeMath for uint256;

        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;

        constructor (address _token,uint _earnLowerlimit) public ERC20Detailed(
                string(abi.encodePacked("yfii ", ERC20Detailed(_token).name())),
                string(abi.encodePacked("i", ERC20Detailed(_token).symbol())),//knownsec// iHBTC
                ERC20Detailed(_token).decimals()
        ) {
                token = IERC20(_token);
                governance = tx.origin;
                controller = 0xcDCf1f9Ac816Fed665B09a00f60c885dd8848b02;
                earnLowerlimit = _earnLowerlimit;
        }

        function balance() public view returns (uint) {//knownsec//  本合约 HBTC 量 + 控制器 iHBTC 量
                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) {//knownsec//  可用量
                return token.balanceOf(address(this)).mul(min).div(max);
        }

        function earn() public {//knownsec//  赚取利息
                uint _bal = available();
                token.safeTransfer(controller, _bal);
                Controller(controller).earn(address(token), _bal);
        }

        function depositAll() external {//knownsec//  存入调用者所有 HBTC
                deposit(token.balanceOf(msg.sender));
        }

        function deposit(uint _amount) public {//knownsec//  存入 HBTC
                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 {//knownsec//  提现所有 iHBTC 为 HBTC
                withdraw(balanceOf(msg.sender));
        }
    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint _shares) public {//knownsec// iHBTC 提现为 HBTC
        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);
    }

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

5. 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。

6. 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