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	20201015	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
BTF 智能合约审计报告	V1.0	BTF-ZNNY-20201015	项目组公开

声明

目录

1. 综述

本次报告有效测试时间是从 2020 年 10 月 14 日开始到 2020 年 10 月 15 日结束，在此期间针对 **BTF 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时时漏洞（见第三章节）进行了全面的分析，发现 Owner 权限过大问题，在部署时注意迁移到时间锁定（Timelock）合约即可解决；存在增发代币问题，但由于该问题需根据交易所要求而定，因此综合评定为**通过**。

本次智能合约安全审计结果： 通过

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

本次测试的目标信息：

条目	描述
Token 名称	BTF
合约地址	https://github.com/btf-finance/btf-contract
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	E8B8423CCC94CB068CD405C05E573697
Converter.sol	FCFD90F035BABD46690AF81B13E2687F
Gauge.sol	23228A3B0ED0B65017B34E59B1D6E1D4

IBTFReferral.sol	3518D80DC2C87178E465A9B6CB703B2D
IController.sol	11DC2129CBD48BCE49F4D576ED5AAFD8
IMigratorChef.sol	45E5B260BFD97CED7872DF8E9B1DFFF6
IStakingRewards.sol	3D193E086B262608FBFDF004F29857FD
IStrategy.sol	2FB2972AFC648E89FAEC2D24248137EA
IStrategyConverter.sol	21014D4CE13C1E44DE8082553BC44611
ISwerveFi.sol	C5CDCF63C5DCB3FFEF098ED4A7DF52B6
IUniswapV2Factory.sol	B66E188E5C62E7C0364632347109B6C8
IVault.sol	C3A5BFA8623AD868A41089D3FC771801
IYfvRewards.sol	7472729512D2D23A4A1AD89ACEFED504
OneSplitAudit.sol	A9153F106A980DF38CA67874149AAF52
UniswapRouterV2.sol	6C600BB869C8F21F0F4184310A17FC7B
USDT.sol	5FCA0C8CF94C05071F5D5A0E10C37F23
StrategySwerve.sol	D8773E77AF699B6FA999305E7286BD12
StrategyUniEthDaiLp.sol	485A761B65DCEA1EDDE46A63E873BCF0
StrategyUniEthUsdcLp.sol	CDADAB02586FC6B2D2F619C6E6E5F5B4
StrategyUniEthUsdtLp.sol	0839333AA32712DAB733CA7F4C207D7A
StrategyUniEthWbtcLp.	25B5B775B9DA1BA541F1E6DE11BB56BC

sol	
StrategyYfvDai.sol	C8A7D2FD9683D318C486515D86D96F45
StrategyYfvTusd.sol	42BFC6DA19E50989013920A776713567
StrategyYfvUsdc.sol	28844FF81ABED106975CF2819E75F25B
StrategyYfvUsdt.sol	AE369C3BB3713BA85F83882B44BE340C
Vault.sol	8821E16F07A00F25F08202D9D301A954
BTFReferral.sol	A6CA80338CA56D46D8D14B8DCF3F62AB
BTFToken.sol	F7B002C03C6BB5DA99EA246478699527
MasterChef.sol	4BF6C672B3AA6CC80D04A7BE54067E47
Migrations.sol	CA8D6CA8A6EDF34F149A5095A8B074C9
MockBtfToken.sol	19754700A2CAF400BD86B2D3A5C3C756
MockERC20.sol	5801424F9432ACD5B9CAEC7EA3A15F2E
Timelock.sol	7FBD9498672695C34393331F21A51B94

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计：

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	34

2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
业务安全性检测	控制器合约变量及构造函数	通过	经检测，不存在安全问题。
	控制器合约 yearn 功能	通过	经检测，不存在安全问题。
	Vault 合约构造函数	通过	经检测，不存在安全问题。
	Vault 合约 deposit 函数	通过	经检测，不存在安全问题。
	Vault 合约 withdraw 函数	通过	经检测，不存在安全问题。
	Strategy 合约 harvest 函数	通过	经检测，不存在安全问题。
	Strategy 合约 withdraw 函数	通过	经检测，不存在安全问题。
代码基本漏洞检测	编译器版本安全	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.orgin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	低危（通过）	经检测，代码中存在 owner 权限过高问题，建议在部署时注意迁移到时间锁定（Timelock）合约即不存在此问题，故综合评定为通过。
	gas 消耗检测	通过	经检测，不存在该安全问题。

	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	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。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	不安全的接口使用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

3. 业务安全性检测

3.1. 控制器合约变量及构造函数【通过】

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

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

    address public constant burn = 0x000000000000000000000000000000000000dEaD;
    address public onesplit = 0xC586BeF4a0992C495Cf22e1aeEE4E446CECDee0E;

    address public governance;//knownsec// 治理地址
    address public strategist;
    address public timelock;

    // community fund
    address public comAddr;//knownsec// 社区地址
    // development fund
    address public devAddr;//knownsec// 开发者地址
    // burn or repurchase
    address public burnAddr;//knownsec// 烧币地址
    mapping(address => address) public vaults;//knownsec// 各代币 vault 合约地址映射
    mapping(address => address) public strategies;//knownsec// 各代币策略合约地址映射
    mapping(address => mapping(address => address)) public converters;//knownsec// 转换器
    mapping(address => mapping(address => address)) public strategyConverters;
    mapping(address => mapping(address => bool)) public approvedStrategies;

    uint256 public split = 500;//knownsec// 奖励抽成 split/max 5%
    uint256 public constant max = 10000;
```

```
    constructor(
        address _governance,//knownsec//  初始化传入治理地址
        address _strategist,//knownsec//  初始化传入策略合约地址
        address _comAddr, // should be the multisig     //knownsec//  初始化传入社区地址
        address _devAddr,
        address _burnAddr, //should be the multisig
        address _timelock
    ) public {
        governance = _governance;
        strategist = _strategist;
        comAddr = _comAddr;
        devAddr = _devAddr;
        burnAddr = _burnAddr;
        timelock = _timelock;
    }
```

安全建议：无。

3.2. 控制器合约 yearn 功能【通过】

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

```
function yearn(
        address _strategy,
        address _token,
        uint256 parts
    ) public {
        require(
            msg.sender == strategist || msg.sender == governance,
            "!governance"
        );
        // This contract should never have value in it, but just incase since this is a public call
```

```
        uint256 _before = IERC20(_token).balanceOf(address(this));
        IStrategy(_strategy).withdraw(_token);//knownsec// 将策略合约的所有 token 提取至本合约
        uint256 _after = IERC20(_token).balanceOf(address(this));
        if (_after > _before) {
            uint256 _amount = _after.sub(_before);//knownsec// 提现的实际值
            address _want = IStrategy(_strategy).want();
            uint256[] memory _distribution;
            uint256 _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_want 代币量
            if (_after > _before) {
                _amount = _after.sub(_before);//knownsec// 转换前后实际差值
                uint256 _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(comAddr, _reward);//knownsec// 奖励转给
```

```
rewards 地址
                }
            }
        }
```

安全建议：无。

3.3. Vault 合约构造函数【通过】

审计分析：以 Vault 为例，Vault 类合约提供其他代币的 bToken 流动性挖矿的功能。

```
constructor(address _token, address _governance, address _controller)
public
ERC20(
    string(abi.encodePacked("btf ", ERC20(_token).name())),
    string(abi.encodePacked("b", ERC20(_token).symbol()))//knownsec// bToken
)
{
    _setupDecimals(ERC20(_token).decimals());
    token = IERC20(_token);
    governance = _governance;//knownsec// 初始化控制器合约地址
    controller = _controller;
}
```

安全建议：无。

3.4. Vualt 合约 deposit 函数【通过】

审计分析：以 Vault.sol 为例，despoit 函数用于存入指定代币转化为相应的流动性生息代币 bToken 以获取收益利息。

```
function deposit(uint256 _amount) public {//knownsec// 存款
        uint256 tokenBalanceWithoutBooster = balance().sub(balanceOf(profitBooster));
```

```
        uint256 totalSupplyWithoutBooster = totalSupply().sub(balanceOf(profitBooster));

        uint256 _before = token.balanceOf(address(this));
        token.safeTransferFrom(msg.sender, address(this), _amount);
        uint256 _after = token.balanceOf(address(this));//knownsec// 计算余额变化
        _amount = _after.sub(_before);
        // Additional check for deflationary tokens
        uint256 shares = 0;
        if (totalSupplyWithoutBooster == 0 || msg.sender == profitBooster) {
            shares = _amount;
        } else {
            shares = _amount.mul(totalSupplyWithoutBooster).div(tokenBalanceWithoutBooster);//knownsec// 计算流动池占比
        }
        _mint(msg.sender, shares);//knownsec// 给调用者铸造凭证
    }
```

安全建议：无。

3.5. Vualt 合约 withdraw 函数【通过】

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

```
    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint256 _shares) public {
        uint256 tokenBalanceWithoutBooster = balance().sub(balanceOf(profitBooster));
        uint256 totalSupplyWithoutBooster = totalSupply().sub(balanceOf(profitBooster));

        uint256 r = 0;
        if (profitBooster == msg.sender) {
```

```
            r = _shares;
        } else {
            r = tokenBalanceWithoutBooster.mul(_shares).div(totalSupplyWithoutBooster);
        }
        _burn(msg.sender, _shares);//knownsec// 烧币 减少供应

        // Check balance
        uint256 b = token.balanceOf(address(this));//knownsec// 余额检查
        if (b < r) {
            uint256 _withdraw = r.sub(b);
            IController(controller).withdraw(address(token), _withdraw);
            uint256 _after = token.balanceOf(address(this));
            uint256 _diff = _after.sub(b);
            if (_diff < _withdraw) {
                r = b.add(_diff);
            }
        }

        token.safeTransfer(msg.sender, r);//knownsec// 给流动性提供者发送资产
    }
```

安全建议：无。

3.6. Strategy 合约 harvest 函数【通过】

审计分析：以 StrategySwerve.sol 为例，harvest 函数用于按指定策略进行投资生息。

```
function harvest() public {//knownsec//赚取流动性利息
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
        // i.e. will be be heavily frontrunned?
```

```
        //          if so, a new strategy will be deployed.

        // stablecoin we want to convert to
        (address to, uint256 toIndex) = getMostPremiumStablecoin();//knownsec//获取尽量多的稳定币

        // Collects SWRV tokens
        Mintr(mintr).mint(rewards);
        uint256 _swrv = IERC20(swrv).balanceOf(address(this));//knownsec//计算本合约 swrv 余额
        if (_swrv > 0) {
            // 10% is locked up for future gov
            if (keepSWRV > 0) {//knownsec//  治理费用扣除
                uint256 _keepSWRV = _swrv.mul(keepSWRV).div(keepSWRVMax);
                IERC20(swrv).safeTransfer(
                    IController(controller).devAddr(),
                    _keepSWRV
                );//knownsec//  治理费用转移到 controller
                _swrv = _swrv.sub(_keepSWRV);
            }

            _swap(swrv, to, _swrv);
        }

        uint256 _to = IERC20(to).balanceOf(address(this));
        if (_to > 0) {
            // Burn some btfs first
            if (burnFee > 0) {//knownsec// Burn 费用
                uint256 _burnFee = _to.mul(burnFee).div(burnMax);
                _swap(to, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
```

```
                );
                _to = _to.sub(_burnFee);
            }
        }

        // Adds in liquidity for swusdv2 pool to get back want (swusd)
        if (_to > 0) {
            IERC20(to).safeApprove(curve, 0);
            IERC20(to).safeApprove(curve, _to);
            uint256[4] memory liquidity;
            liquidity[toIndex] = _to;
            ISwerveFi(curve).add_liquidity(liquidity, 0);
        }

        // We want to get back swrv
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // 4.5% rewards gets sent to treasury
            IERC20(want).safeTransfer(
                IController(controller).comAddr(),
                _want.mul(performanceFee).div(performanceMax)
            );

            deposit();// knownsec //收益转入
        }
    }
```

安全建议：无。

3.7. Strategy 合约 withdraw 函数【通过】

审计分析：以 StrategySwerve.sol 为例，withdraw 函数用于提现流动性代币

至 Vault 合约。

```
// Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {//knownsec// 提现资产 量 _amount
        require(msg.sender == controller, "!controller");//knownsec// 仅控制器调用
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {//knownsec// 处理余额不够的情况
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {//knownsec// 处理提取手续费
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }

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

        IERC20(want).safeTransfer(_vault, _amount);//knownsec// 转账
    }

    // Withdraw all funds, normally used when migrating strategies
    function withdrawAll() external returns (uint256 balance) {//knownsec// 提现所有资产,
controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

        address _vault = IController(controller).vaults(address(want));
```

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }

    function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec// 部分提现  内部使用
        Gauge(rewards).withdraw(_amount);
        return _amount;
    }
```

安全建议：无。

4. 代码基本漏洞检测

4.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.5.15 以上，不存在该安全问题。

安全建议：无。

4.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

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

安全建议：无。

4.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7. tx.origin 身份验证【通过】

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

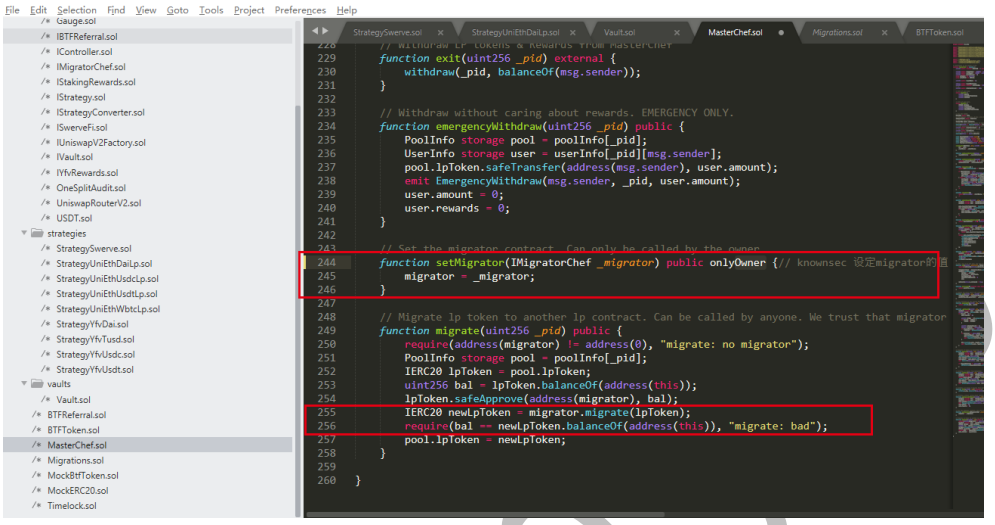
检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8. owner 权限控制【低危】

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 MasterChef.sol 在 244 行 Owner 有权限来设定下图中 245 行 migrator 变量的值，该值的设定可以决定由哪一个 migrator 合约的代码来进行后面的操作。



当 migrator 的值被确定之后（如上图中 255 行代码显示），migrator.migrate(lpToken)也就可以被随之确定。由 migrate 的方法是通过 IMigratorChef 的接口来进行调用的，因此在调用的时候，migrate 的方法中的逻辑代码会根据 migrator 值的不同而变化。

安全建议：将该项目迁移到时间锁定（Timelock）合约，即任意 BTFSwap 项目智能合约拥有者的操作会有 48 小时的延迟锁定。

4.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.10. call 注入攻击【通过】

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

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.11. 低级函数安全【通过】

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

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.12. 增发代币漏洞【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,在 BTFToken.sol 的 15 行。

```
function mint(address _to, uint256 _amount) public onlyOwner {//   knownsec//增发代币
        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _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。

4.13. 访问控制缺陷检测【通过】

合约中不同函数应设置合理的权限

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.14. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

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

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.15. 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、函数、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整

数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20，而 5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.16. 错误使用随机数【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.17. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.18. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.19. 未初始化的储存指针【通过】

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

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

检测结果：经检测，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

4.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继

续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.22. 时间戳依赖攻击【通过】

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

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24. 假充值漏洞【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25. 重入攻击检测【通过】

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

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

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

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.27. 重排攻击检测【通过】

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

检测结果:经检测，智能合约代码中不存在相关漏洞。

安全建议:无。

5. 附录 A：合约代码

本次测试代码来源：

Controller.sol

```
pragma solidity ^0.6.7;//knownsec//  指定编译器版本

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";

import "../interface/IStrategy.sol";
import "../interface/IVault.sol";
import "../interface/Converter.sol";
import "../interface/IStrategyConverter.sol";
import "../interface/OneSplitAudit.sol";


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

    address public constant burn = 0x000000000000000000000000000000000000dEaD;
    address public onesplit = 0xC586BeF4a0992C495Cf22e1aeEE4E446CECDee0E;

    address public governance;//knownsec//  治理地址
    address public strategist;
    address public timelock;

    // community fund
    address public comAddr;//knownsec//  社区地址
    // development fund
    address public devAddr;//knownsec//  开发者地址
    // burn or repurchase
    address public burnAddr;//knownsec//  烧币地址
    mapping(address => address) public vaults;//knownsec//  各代币 vault 合约地址映射
    mapping(address => address) public strategies;//knownsec//  各代币策略合约地址映射
    mapping(address => mapping(address => address)) public converters;//knownsec//  转换器
    mapping(address => mapping(address => address)) public strategyConverters;
    mapping(address => mapping(address => bool)) public approvedStrategies;

    uint256 public split = 500;//knownsec//  奖励抽成 split/max 5%
    uint256 public constant max = 10000;

    constructor(
        address _governance,//knownsec//  初始化传入治理地址
        address _strategist,//knownsec//  初始化传入策略合约地址
        address _comAddr, // should be the multisig     //knownsec//  初始化传入社区地址
        address _devAddr,
        address _burnAddr, //should be the multisig
        address _timelock
    ) public {
        governance = _governance;
        strategist = _strategist;
        comAddr = _comAddr;
        devAddr = _devAddr;
        burnAddr = _burnAddr;
        timelock = _timelock;
    }

    function setComAddr(address _comAddr) public {//knownsec//  改变社区地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        comAddr = _comAddr;
    }

    function setDevAddr(address _devAddr) public {//knownsec//  改变开发者地址,仅治理地址调用
        require(msg.sender == governance || msg.sender == devAddr, "!governance");
        devAddr = _devAddr;
    }

    function setBurnAddr(address _burnAddr) public {//knownsec//  改变烧币地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        burnAddr = _burnAddr;
    }
```

```
    function setStrategist(address _strategist) public {//knownsec// 改变策略地址,仅治理地址调用
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
    }

    function setSplit(uint256 _split) public {
        require(msg.sender == governance, "!governance");//knownsec// 改变分成比例,仅治理地址调用
        split = _split;
    }

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

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

    function setTimelock(address _timelock) public {//knownsec// 设置 timelock 管理地址,仅 timelock 管理地址调用
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
    }

    function setVault(address _token, address _vault) public {//knownsec// 添加新代币及策略,仅治理地址和策略管理地址调用
        require(
            msg.sender == strategist || msg.sender == governance,
            "!strategist"
        );
        require(vaults[_token] == address(0), "vault");
        vaults[_token] = _vault;
    }

    function approveStrategy(address _token, address _strategy) public {//knownsec// 启用策略,仅 timelock 管理地址调用
        require(msg.sender == timelock, "!timelock");
        approvedStrategies[_token][_strategy] = true;
    }

    function revokeStrategy(address _token, address _strategy) public {//knownsec// 停止策略,仅 timelock 管理地址调用
        require(msg.sender == governance, "!governance");
        approvedStrategies[_token][_strategy] = false;
    }

    function setConverter(//knownsec// 设置转换器地址,仅治理地址和策略管理地址调用
        address _input,
        address _output,
        address _converter
    ) public {
        require(
            msg.sender == strategist || msg.sender == governance,
            "!strategist"
        );
        converters[_input][_output] = _converter;
    }

    function setStrategyConverter(
        address[] memory stratFrom,
        address[] memory stratTo,
        address _stratConverter
    ) public {//knownsec// 设置策略转换地址,仅治理地址和策略管理地址调用
        require(
            msg.sender == strategist || msg.sender == governance,
            "!strategist"
        );

        for (uint256 i = 0; i < stratFrom.length; i++) {
            for (uint256 j = 0; j < stratTo.length; j++) {
                strategyConverters[stratFrom[i]][stratTo[j]] = _stratConverter;
            }
        }
    }

    function setStrategy(address _token, address _strategy) public {//knownsec// 添加新策略地址,仅治理地址和策略管理地址调用
        require(
            msg.sender == strategist || msg.sender == governance,
            "!strategist"
        );
        require(approvedStrategies[_token][_strategy] == true, "!approved");

        address _current = strategies[_token];
```

```
        if (_current != address(0)) {//knownsec//之前的策略存在的话,那么就先提取所有资金
            IStrategy(_current).withdrawAll();
        }
        strategies[_token] = _strategy;
    }

    function earn(address _token, uint256 _amount) public {//knownsec//  存币
        address _strategy = strategies[_token];//knownsec //获取策略的合约地址
        address _want = IStrategy(_strategy).want();//knownsec //策略需要的 token 地址
        if (_want != _token) {//knownsec //如果策略需要的和输入的不一样,需要先转换
            address converter = converters[_token][_want];//knownsec //转换器合约地址
            IERC20(_token).safeTransfer(converter, _amount);//knownsec //给转换器打钱
            _amount = Converter(converter).convert(_strategy);//knownsec //执行转换...
            IERC20(_want).safeTransfer(_strategy, _amount);
        } else {
            IERC20(_token).safeTransfer(_strategy, _amount);
        }
        IStrategy(_strategy).deposit();//knownsec //存钱
    }

    function balanceOf(address _token) external view returns (uint256) {//knownsec//  获取策略中 token 的余额
        return IStrategy(strategies[_token]).balanceOf();
    }

    function withdrawAll(address _token) public {//knownsec//  提取指定代币所有余额,仅治理地址和策略管理地址调用
        require(
            msg.sender == strategist || msg.sender == governance,
            "!strategist"
        );
        IStrategy(strategies[_token]).withdrawAll();
    }

    function inCaseTokensGetStuck(address _token, uint256 _amount) public {//knownsec //转任意 erc20
        require(
            msg.sender == strategist || msg.sender == governance,
            "!governance"
        );
        IERC20(_token).safeTransfer(msg.sender, _amount);//knownsec//  将指定代币转移至治理地址账户
    }

    function inCaseStrategyTokenGetStuck(address _strategy, address _token)
    public
    {
        require(
            msg.sender == strategist || msg.sender == governance,
            "!governance"
        );
        IStrategy(_strategy).withdraw(_token);
    }

    function getExpectedReturn(
        address _strategy,
        address _token,
        uint256 parts
    ) public view returns (uint256 expected) {
        uint256 _balance = IERC20(_token).balanceOf(_strategy);//knownec //获取策略器_某个代币的余额
        address _want = IStrategy(_strategy).want();//knownec //获取策略器需要的代币地址
        (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,
        uint256 parts
    ) public {
        require(
            msg.sender == strategist || msg.sender == governance,
            "!governance"
        );
        // This contract should never have value in it, but just incase since this is a public call
        uint256 _before = IERC20(_token).balanceOf(address(this));
        IStrategy(_strategy).withdraw(_token);//knownsec//  将策略合约的所有 token 提取至本合约
        uint256 _after = IERC20(_token).balanceOf(address(this));
        if (_after > _before) {
            uint256 _amount = _after.sub(_before);//knownsec//  提现的实际值
            address _want = IStrategy(_strategy).want();
            uint256[] memory _distribution;
            uint256 _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_want 代币量
                if (_after > _before) {
                    _amount = _after.sub(_before);//knownsec// 转换前后实际差值
                    uint256 _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(comAddr, _reward);//knownsec// 奖励转给 rewards 地址
                }
            }
        }

        function withdraw(address _token, uint256 _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");
            IStrategy(strategies[_token]).withdraw(_amount);
        }

        // Swaps between vaults
        // Note: This is supposed to be called
        //       by a user if they'd like to swap between vaults w/o the 0.5% fee
        function userSwapVault(
            address _fromToken,
            address _toToken,
            uint256 _pAmount // Pickling token amount to convert
        ) public nonReentrant returns (uint256) {//knownsec// 用户进行代币置换
            address _fromVault = vaults[_fromToken];
            address _toVault = vaults[_toToken];

            address _fromStrategy = strategies[_fromToken];
            address _toStrategy = strategies[_toToken];


            address _strategyConverter = strategyConverters[_fromStrategy][_toStrategy];

            require(_strategyConverter != address(0), "!strategy-converter");

            // 1. Transfers bVault tokens from msg.sender
            IVault(_fromVault).transferFrom(msg.sender, address(this), _pAmount);//knownsec// 收取置换者 token

            // 2. Get amount of tokens to transfer from strategy to burn
            // Note: this token amount is the LP token
            uint256 _fromTokenAmount = IVault(_fromVault).getRatio().mul(_pAmount).div(
                1e18
            );//knownsec// 计算 LP_token 比例

            // If we don't have enough funds in the strategy
            // We'll deposit funds from the vault to the strategy
            // Note: This assumes that no single person is responsible
            //       for 100% of the liquidity.
            // If this a single person is 100% responsible for the liquidity
            // we can simply set min = max in vaults
            if (IStrategy(_fromStrategy).balanceOf() < _fromTokenAmount) {//knownsec// 处理仓库 token 不够情况
                IVault(_fromVault).earn();
            }

            // 3. Withdraw tokens from strategy and burns pToken
            IVault(_fromVault).transfer(burn, _pAmount);//knownsec// 从仓库烧币
            IStrategy(_fromStrategy).freeWithdraw(_fromTokenAmount);

            // 4. Converts to Token
            IERC20(_fromToken).approve(_strategyConverter, _fromTokenAmount);
            IStrategyConverter(_strategyConverter).convert(
                msg.sender,
                _fromToken,
                _toToken,
                _fromTokenAmount
            );//knownsec// 转换 token

            // 5. Deposits into BFTVault
            uint256 _toTokenAmount = IERC20(_toToken).balanceOf(address(this));//knownsec// 获取结果货币余
额
            IERC20(_toToken).approve(_toVault, _toTokenAmount);
            IVault(_toVault).deposit(_toTokenAmount);//knownsec// 存款

            // 6. Sends msg.sender all the btf vault tokens
            uint256 _retPAmount = IVault(_toVault).balanceOf(address(this));
```

```
            IVault(_toVault).transfer(
                msg.sender,
                _retPAmount
            );//knownsec//  转给存币人凭据

            return _retPAmount;
        }
    }
```

Vault.sol

```
pragma solidity ^0.6.7;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";

contract Vault is ERC20 {
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;

    IERC20 public token;

    uint256 public min = 9500;
    uint256 public constant max = 10000;

    address public governance;
    address public controller;
    // todo
    address public constant profitBooster = address(0);

    constructor(address _token, address _governance, address _controller)
    public
    ERC20(
        string(abi.encodePacked("btf ", ERC20(_token).name())),
        string(abi.encodePacked("b", ERC20(_token).symbol()))
    )
    {
        _setupDecimals(ERC20(_token).decimals());
        token = IERC20(_token);
        governance = _governance;
        controller = _controller;
    }

    function balance() public view returns (uint256) {
        return
        token.balanceOf(address(this)).add(
            IController(controller).balanceOf(address(token))
        );
    }

    function balanceOfToken() public view returns (uint256) {
        if (profitBooster == msg.sender) {
            return balanceOf(profitBooster);
        } else {
            return (balance().sub(balanceOf(profitBooster)))
            .mul(balanceOf(msg.sender))
            .div(totalSupply().sub(balanceOf(profitBooster)));
        }
    }

    function setMin(uint256 _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;
    }

    // Custom logic in here for how much the token allows to be borrowed
    // Sets minimum required on-hand to keep small withdrawals cheap
    function available() public view returns (uint256) {
        return token.balanceOf(address(this)).mul(min).div(max);
    }

    function earn() public {
```

```
        uint256 _bal = available();
        token.safeTransfer(controller, _bal);
        IController(controller).earn(address(token), _bal);
    }

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

    function deposit(uint256 _amount) public {//knownsec// 存款
        uint256 tokenBalanceWithoutBooster = balance().sub(balanceOf(profitBooster));
        uint256 totalSupplyWithoutBooster = totalSupply().sub(balanceOf(profitBooster));

        uint256 _before = token.balanceOf(address(this));
        token.safeTransferFrom(msg.sender, address(this), _amount);
        uint256 _after = token.balanceOf(address(this));//knownsec// 计算余额变化
        _amount = _after.sub(_before);
        // Additional check for deflationary tokens
        uint256 shares = 0;
        if (totalSupplyWithoutBooster == 0 || msg.sender == profitBooster) {
            shares = _amount;
        } else {
            shares = _amount.mul(totalSupplyWithoutBooster).div(tokenBalanceWithoutBooster);//knownsec//
计算流动池占比
        }
        _mint(msg.sender, shares);//knownsec// 给调用者铸造凭证
    }

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

    // Used to swap any borrowed reserve over the debt limit to liquidate to 'token'
    function harvest(address reserve, uint256 amount) external {
        require(msg.sender == controller, "!controller");
        require(reserve != address(token), "token");
        IERC20(reserve).safeTransfer(controller, amount);
    }

    // No rebalance implementation for lower fees and faster swaps
    function withdraw(uint256 _shares) public {
        uint256 tokenBalanceWithoutBooster = balance().sub(balanceOf(profitBooster));
        uint256 totalSupplyWithoutBooster = totalSupply().sub(balanceOf(profitBooster));

        uint256 r = 0;
        if (profitBooster == msg.sender) {
            r = _shares;
        } else {
            r = tokenBalanceWithoutBooster.mul(_shares).div(totalSupplyWithoutBooster);
        }
        _burn(msg.sender, _shares);//knownsec// 烧币 减少供应

        // Check balance
        uint256 b = token.balanceOf(address(this));//knownsec// 余额检查
        if (b < r) {
            uint256 _withdraw = r.sub(b);
            IController(controller).withdraw(address(token), _withdraw);
            uint256 _after = token.balanceOf(address(this));
            uint256 _diff = _after.sub(b);
            if (_diff < _withdraw) {
                r = b.add(_diff);
            }
        }
        token.safeTransfer(msg.sender, r);//knownsec// 给流动性提供者发送资产
    }

    function getRatio() public view returns (uint256) {
        return balance().mul(1e18).div(totalSupply());
    }
}
```

StrategySwerve.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";
import "../interface/IStrategy.sol";
import "../interface/Gauge.sol";
import "../interface/ISwerveFi.sol";
import "../interface/UniswapRouterV2.sol";
```

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

    // gauge of swerve
    address public constant rewards = 0xb4d0C929cD3A1FbDc6d57E7D3315cF0C4d6B4bFa;

    // swUSD lp tokens
    address public constant want = 0x77C6E4a580c0dCE4E5c7a17d0bc077188a83A059;

    // tokens we're farming
    address public constant swrv = 0xB8BAa0e4287890a5F79863aB62b7F175ceCbD433;

    // swusdv2 pool
    address public constant curve = 0x329239599afB305DA0A2eC69c58F8a6697F9F88d;

    // swerve minter
    address constant public mintr = 0x2c988c3974AD7E604E276AE0294a7228DEf67974;//knownsec// 矿工

    // stablecoins
    address public constant dai = 0x6B175474E89094C44Da98b954EedeAC495271d0F;
    address public constant usdc = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;
    address public constant usdt = 0xdAC17F958D2ee523a2206206994597C13D831ec7;
    address public constant tusd = 0x0000000000085d4780B73119b644AE5ecd22b376;

    // weth
    address public constant weth = 0xc778417E063141139Fce010982780140Aa0cD5Ab;

    // dex
    address public constant univ2Router2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

    // Fees 5% in total
    // - 1.5%    keepSWRV for development fund
    // - 2%      performanceFee for community fund
    // - 1.5%    used to burn/repurchase btfs
    uint256 public keepSWRV = 150;
    uint256 public constant keepSWRVMax = 10000;

    uint256 public performanceFee = 200;
    uint256 public constant performanceMax = 10000;

    uint256 public burnFee = 150;
    uint256 public constant burnMax = 10000;

    uint256 public withdrawalFee = 0;
    uint256 public constant withdrawalMax = 10000;

    address public governance;
    address public controller;
    address public strategist;
    address public timelock;
    address public btf;

    constructor(
        address _governance,
        address _strategist,
        address _controller,
        address _timelock,
        address _btf
    ) public {
        governance = _governance;
        strategist = _strategist;
        controller = _controller;
        timelock = _timelock;
        btf = _btf;
    }//knownsec// 初始化合约，设置 governance、strategist、controller、timelock、btf 地址

    // **** Views ****

    function balanceOfWant() public view returns (uint256) {
        return IERC20(want).balanceOf(address(this));
    }

    function balanceOfPool() public view returns (uint256) {
        return Gauge(rewards).balanceOf(address(this));
    }

    function balanceOf() public view returns (uint256) {
        return balanceOfWant().add(balanceOfPool());
    }

    function getName() external pure returns (string memory) {
        return "StrategySwerve";
    }

    function getHarvestable() external returns (uint256) {
```

```
        return Gauge(rewards).claimable_tokens(address(this));
    }

    // **** Setters ****

    function setKeepSWRV(uint256 _keepSWRV) external {
        require(msg.sender == governance, "!governance");
        keepSWRV = _keepSWRV;
    }

    function setWithdrawalFee(uint256 _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        withdrawalFee = _withdrawalFee;
    }

    function setPerformanceFee(uint256 _performanceFee) external {
        require(msg.sender == governance, "!governance");
        performanceFee = _performanceFee;
    }

    function setBurnFee(uint256 _burnFee) external {
        require(msg.sender == governance, "!governance");
        burnFee = _burnFee;
    }

    function setStrategist(address _strategist) external {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
    }

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

    function setTimelock(address _timelock) external {
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
    }

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

    function getMostPremiumStablecoin() public view returns (address, uint256) {// knownsec  获取最多的稳定币
        uint256[] memory balances = new uint256[](4);
        // DAI
        balances[0] = ISwerveFi(curve).balances(0);
        // USDC
        balances[1] = ISwerveFi(curve).balances(1).mul(10 ** 12);
        // USDT
        balances[2] = ISwerveFi(curve).balances(2).mul(10 ** 12);
        // TUSD
        balances[3] = ISwerveFi(curve).balances(3);

        // DAI
        if (
            balances[0] < balances[1] &&
            balances[0] < balances[2] &&
            balances[0] < balances[3]
        ) {
            return (dai, 0);
        }

        // USDC
        if (
            balances[1] < balances[0] &&
            balances[1] < balances[2] &&
            balances[1] < balances[3]
        ) {
            return (usdc, 1);
        }

        // USDT
        if (
            balances[2] < balances[0] &&
            balances[2] < balances[1] &&
            balances[2] < balances[3]
        ) {
            return (usdt, 2);
        }

        // TUSD
        if (
            balances[3] < balances[0] &&
```

```
            balances[3] < balances[1] &&
            balances[3] < balances[2]
        ) {
            return (tusd, 3);
        }

        // If they're somehow equal, we just want DAI
        return (dai, 0);// 稳定币全部相等选择 DAI
    }

    // **** State Mutations ****

    function deposit() public {//knownsec// 流动性挖矿 转钱
        uint256 _want = IERC20(want).balanceOf(address(this));//knownsec 计算本合约输出代币余额
        if (_want > 0) {
            IERC20(want).safeApprove(rewards, 0);// knownsec 解决事务依赖问题 置 0
            IERC20(want).approve(rewards, _want);
            Gauge(rewards).deposit(_want);
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint256 balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        require(swrv != address(_asset), "swrv");

        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Contoller only function for withdrawing for free
    // This is used to swap between vaults
    function freeWithdraw(uint256 _amount) external {// knownsec 仓库间代币置换
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }
        IERC20(want).safeTransfer(msg.sender, _amount);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {//knownsec// 提现资产到 vault 量 _amount 控制器可用
        require(msg.sender == controller, "!controller");//knownsec// 仅控制器调用
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {//knownsec// 处理余额不够的情况
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {//knownsec// 处理提取手续费
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }

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

        IERC20(want).safeTransfer(_vault, _amount);//knownsec// 转账
    }

    // Withdraw all funds, normally used when migrating strategies
    function withdrawAll() external returns (uint256 balance) {//knownsec// 提现所有资产, controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }

    function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec// 部分提现 内部使用
        Gauge(rewards).withdraw(_amount);
        return _amount;
    }
```

```
function brine() public {
        harvest();
    }

    function harvest() public {//knownsec//赚取流动性利息
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
        // i.e. will be be heavily frontrunned?
        //          if so, a new strategy will be deployed.

        // stablecoin we want to convert to
        (address to, uint256 toIndex) = getMostPremiumStablecoin();//knownsec//获取尽量多的稳定币

        // Collects SWRV tokens
        Mintr(mintr).mint(rewards);
        uint256 _swrv = IERC20(swrv).balanceOf(address(this));//knownsec//计算本合约 swrv 余额
        if (_swrv > 0) {
            // 10% is locked up for future gov
            if (keepSWRV > 0) {//knownsec//  治理费用扣除
                uint256 _keepSWRV = _swrv.mul(keepSWRV).div(keepSWRVMax);
                IERC20(swrv).safeTransfer(
                    IController(controller).devAddr(),
                    _keepSWRV
                );//knownsec//  治理费用转移到 controller
                _swrv = _swrv.sub(_keepSWRV);
            }

            _swap(swrv, to, _swrv);
        }

        uint256 _to = IERC20(to).balanceOf(address(this));
        if (_to > 0) {
            // Burn some btfs first
            if (burnFee > 0) {//knownsec// Burn 费用
                uint256 _burnFee = _to.mul(burnFee).div(burnMax);
                _swap(to, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
                );
                _to = _to.sub(_burnFee);
            }
        }

        // Adds in liquidity for swusdv2 pool to get back want (swusd)
        if (_to > 0) {
            IERC20(to).safeApprove(curve, 0);
            IERC20(to).safeApprove(curve, _to);
            uint256[4] memory liquidity;
            liquidity[toIndex] = _to;
            ISwerveFi(curve).add_liquidity(liquidity, 0);
        }

        // We want to get back swrv
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // 4.5% rewards gets sent to treasury
            IERC20(want).safeTransfer(
                IController(controller).comAddr(),
                _want.mul(performanceFee).div(performanceMax)
            );

            deposit();
        }
    }

    // Emergency function call
    function execute(address _target, bytes memory _data)
    public
    payable
    returns (bytes memory response)// knownsec  避险函数     timelock 地址可用
    {
        require(msg.sender == timelock, "!timelock");

        require(_target != address(0), "!target");

        // call contract in current context
        assembly {
            let succeeded := delegatecall(
            sub(gas(), 5000),
            _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
            )
```

```
            let size := returndatasize()

            response := mload(0x40)
            mstore(
            0x40,
            add(response, and(add(add(size, 0x20), 0x1f), not(0x1f)))
            )
            mstore(response, size)
            returndatacopy(add(response, 0x20), 0, size)

            switch iszero(succeeded)
            case 1 {
            // throw if delegatecall failed
                revert(add(response, 0x20), size)
            }
        }
    }

    /**
    * Creates a Swerve lock
    */
    function createLock(address lockToken, address escrow, uint256 value, uint256 unlockTime) public {
        require(msg.sender == governance, "!governance");
        IERC20(lockToken).safeApprove(escrow, 0);
        IERC20(lockToken).safeApprove(escrow, value);
        VotingEscrow(escrow).create_lock(value, unlockTime);
    }

    /**
    * Checkpoints the Swerve lock balance
    */
    function checkpoint(address _gauge) public {
        require(msg.sender == governance, "!governance");
        Gauge(_gauge).user_checkpoint(address(this));
    }

    /**
    * Increases the lock amount for Swerve
    */
    function increaseAmount(address lockToken, address escrow, uint256 value) public {
        require(msg.sender == governance, "!governance");
        IERC20(lockToken).safeApprove(escrow, 0);
        IERC20(lockToken).safeApprove(escrow, value);
        VotingEscrow(escrow).increase_amount(value);
    }

    /**
    * Increases the unlock time for Swerve
    */
    function increaseUnlockTime(address escrow, uint256 unlock_time) public {
        require(msg.sender == governance, "!governance");
        VotingEscrow(escrow).increase_unlock_time(unlock_time);
    }

    /**
    * Withdraws an expired lock
    */
    function withdrawLock(address lockToken, address escrow) public {
        require(msg.sender == governance, "!governance");
        uint256 balanceBefore = IERC20(lockToken).balanceOf(address(this));
        VotingEscrow(escrow).withdraw();
        uint256 balanceAfter = IERC20(lockToken).balanceOf(address(this));
        if (balanceAfter > balanceBefore) {
            IERC20(lockToken).safeTransfer(msg.sender, balanceAfter.sub(balanceBefore));
        }
    }

    // **** Internal functions ****
    function _swap(
        address _from,
        address _to,
        uint256 _amount
    ) internal {// Knownsec  交换代币
        // Swap with uniswap
        IERC20(_from).safeApprove(univ2Router2, 0);
        IERC20(_from).safeApprove(univ2Router2, _amount);

        address[] memory path;

        if (_from == weth || _to == weth) {
            path = new address[](2);
            path[0] = _from;
            path[1] = _to;
        } else {
            path = new address[](3);
            path[0] = _from;
            path[1] = weth;
```

```
            path[2] = _to;
        }

        UniswapRouterV2(univ2Router2).swapExactTokensForTokens(
            _amount,
            0,
            path,
            address(this),
            now.add(60)
        );
    }
}
```

StrategyUniEthDaiLp.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";
import "../interface/IStrategy.sol";
import "../interface/IStakingRewards.sol";
import "../interface/UniswapRouterV2.sol";

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

    // Staking rewards address for ETH/DAI LP providers
    address public constant rewards = 0xa1484C3aa22a66C62b77E0AE78E15258bd0cB711;

    // want eth/dai lp tokens
    address public constant want = 0xA478c2975Ab1Ea89e8196811F51A7B7Ade33eB11;

    // tokens we're farming
    address public constant uni = 0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984;

    // stablecoins
    address public constant dai = 0x6B175474E89094C44Da98b954EedeAC495271d0F;

    // weth
    address public constant weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

    // dex
    address public univ2Router2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

    // Fees 5% in total
    // - 1.5%     keepUNI for development fund
    // - 2%       performanceFee for community fund
    // - 1.5%     used to burn/repurchase btfs
    uint256 public keepUNI = 150;
    uint256 public constant keepUNIMax = 10000;

    uint256 public performanceFee = 200;
    uint256 public constant performanceMax = 10000;

    uint256 public burnFee = 150;
    uint256 public constant burnMax = 10000;

    uint256 public withdrawalFee = 0;
    uint256 public constant withdrawalMax = 10000;

    address public governance;
    address public controller;
    address public strategist;
    address public timelock;
    address public btf;

    constructor(
        address _governance,
        address _strategist,
        address _controller,
        address _timelock,
        address _btf
    ) public {
        governance = _governance;
        strategist = _strategist;
        controller = _controller;
        timelock = _timelock;
        btf = _btf;
    }//knownsec// 初始化合约，设置 governance、strategist、controller、timelock、btf 地址
```

```
// **** Views ****

function balanceOfWant() public view returns (uint256) {
        return IERC20(want).balanceOf(address(this));
}

function balanceOfPool() public view returns (uint256) {
        return IStakingRewards(rewards).balanceOf(address(this));
}

function balanceOf() public view returns (uint256) {
        return balanceOfWant().add(balanceOfPool());
}

function getName() external pure returns (string memory) {
        return "StrategyUniEthDaiLp";
}

function getHarvestable() external view returns (uint256) {
        return IStakingRewards(rewards).earned(address(this));
}

// **** Setters ****

function setKeepUNI(uint256 _keepUNI) external {
        require(msg.sender == governance, "!governance");
        keepUNI = _keepUNI;
}

function setWithdrawalFee(uint256 _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        withdrawalFee = _withdrawalFee;
}

function setPerformanceFee(uint256 _performanceFee) external {
        require(msg.sender == governance, "!governance");
        performanceFee = _performanceFee;
}

function setBurnFee(uint256 _burnFee) external {
        require(msg.sender == governance, "!governance");
        burnFee = _burnFee;
}

function setStrategist(address _strategist) external {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
}

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

function setTimelock(address _timelock) external {
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
}

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

// **** State Mutations ****

function deposit() public {//knownsec// 流动性挖矿 转钱
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
                IERC20(want).safeApprove(rewards, 0);
                IERC20(want).approve(rewards, _want);
                IStakingRewards(rewards).stake(_want);
        }
}

// Controller only function for creating additional rewards from dust
function withdraw(IERC20 _asset) external returns (uint256 balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
}

// Contoller only function for withdrawing for free
// This is used to swap between vaults
function freeWithdraw(uint256 _amount) external {// knownsec 仓库间代币置换
```

```
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }
        IERC20(want).safeTransfer(msg.sender, _amount);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {//knownsec//  提现资产到vault     量 _amount    控制器可用
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }

        address _vault = IController(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 (uint256 balance) {//knownsec//  提现所有资产, controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }

    function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec//  部分提现    内部使用
        IStakingRewards(rewards).withdraw(_amount);
        return _amount;
    }

    function brine() public {
        harvest();
    }

    function harvest() public {//knownsec//赚取流动性利息
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
        // i.e. will be be heavily frontrunned?
        //          if so, a new strategy will be deployed.

        // Collects UNI tokens
        IStakingRewards(rewards).getReward();
        uint256 _uni = IERC20(uni).balanceOf(address(this));
        if (_uni > 0) {
            // 10% is locked up for future gov
            if (keepUNI > 0) {
                uint256 _keepUNI = _uni.mul(keepUNI).div(keepUNIMax);
                IERC20(uni).safeTransfer(
                    IController(controller).devAddr(),
                    _keepUNI
                );
                _uni = _uni.sub(_keepUNI);
            }

            _swap(uni, weth, _uni);
        }

        // Swap half WETH for DAI
        uint256 _weth = IERC20(weth).balanceOf(address(this));
        if (_weth > 0) {
            // Burn some btfs first
            if (burnFee > 0) {
                uint256 _burnFee = _weth.mul(burnFee).div(burnMax);
```

```
                _swap(weth, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
                );
                _weth = _weth.sub(_burnFee);
            }

            _swap(weth, dai, _weth.div(2));
        }

        // Adds in liquidity for ETH/DAI
        _weth = IERC20(weth).balanceOf(address(this));
        uint256 _dai = IERC20(dai).balanceOf(address(this));
        if (_weth > 0 && _dai > 0) {
            IERC20(weth).safeApprove(univ2Router2, 0);
            IERC20(weth).safeApprove(univ2Router2, _weth);

            IERC20(dai).safeApprove(univ2Router2, 0);
            IERC20(dai).safeApprove(univ2Router2, _dai);

            UniswapRouterV2(univ2Router2).addLiquidity(
                weth,
                dai,
                _weth,
                _dai,
                0,
                0,
                address(this),
                now + 60
            );

            // Donates DUST
            IERC20(weth).transfer(
                IController(controller).comAddr(),
                IERC20(weth).balanceOf(address(this))
            );
            IERC20(dai).transfer(
                IController(controller).comAddr(),
                IERC20(dai).balanceOf(address(this))
            );
        }

        // We want to get back UNI ETH/DAI LP tokens
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // Performance fee
            if (performanceFee > 0) {
                IERC20(want).safeTransfer(
                    IController(controller).comAddr(),
                    _want.mul(performanceFee).div(performanceMax)
                );
            }

            deposit();
        }
    }

    // Emergency function call
    function execute(address _target, bytes memory _data)
    public
    payable
    returns (bytes memory response)// knownsec  避险函数     timelock 地址可用
    {
        require(msg.sender == timelock, "!timelock");

        require(_target != address(0), "!target");

        // call contract in current context
        assembly {
            let succeeded := delegatecall(
            sub(gas(), 5000),
            _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
            )
            let size := returndatasize()

            response := mload(0x40)
            mstore(
            0x40,
            add(response, and(add(add(size, 0x20), 0x1f), not(0x1f)))
            )
            mstore(response, size)
            returndatacopy(add(response, 0x20), 0, size)
```

```
                switch iszero(succeeded)
                case 1 {
                // throw if delegatecall failed
                    revert(add(response, 0x20), size)
                }
            }
        }

        // **** Internal functions ****
        function _swap(
            address _from,
            address _to,
            uint256 _amount
        ) internal {// knownsec  交换代币
            // Swap with uniswap
            IERC20(_from).safeApprove(univ2Router2, 0);
            IERC20(_from).safeApprove(univ2Router2, _amount);

            address[] memory path;

            if (_from == weth || _to == weth) {
                path = new address[](2);
                path[0] = _from;
                path[1] = _to;
            } else {
                path = new address[](3);
                path[0] = _from;
                path[1] = weth;
                path[2] = _to;
            }

            UniswapRouterV2(univ2Router2).swapExactTokensForTokens(
                _amount,
                0,
                path,
                address(this),
                now.add(60)
            );
        }
}
```

StrategyUniEthUsdcLp.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";
import "../interface/IStrategy.sol";
import "../interface/IStakingRewards.sol";
import "../interface/UniswapRouterV2.sol";

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

    // Staking rewards address for ETH/USDC LP providers
    address public constant rewards = 0x7FBa4B8Dc5E7616e59622806932DBea72537A56b;

    // want eth/usdc lp tokens
    address public constant want = 0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc;

    // tokens we're farming
    address public constant uni = 0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984;

    // stablecoins
    address public constant usdc = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;

    // weth
    address public constant weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

    // dex
    address public univ2Router2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

    // Fees 5% in total
    // - 1.5%     keepUNI for development fund
    // - 2%       performanceFee for community fund
    // - 1.5%     used to burn/repurchase btfs
    uint256 public keepUNI = 150;
    uint256 public constant keepUNIMax = 10000;
```

```
    uint256 public performanceFee = 200;
    uint256 public constant performanceMax = 10000;

    uint256 public burnFee = 150;
    uint256 public constant burnMax = 10000;

    uint256 public withdrawalFee = 0;
    uint256 public constant withdrawalMax = 10000;

    address public governance;
    address public controller;
    address public strategist;
    address public timelock;
    address public btf;

    constructor(
        address _governance,
        address _strategist,
        address _controller,
        address _timelock,
        address _btf
    ) public {
        governance = _governance;
        strategist = _strategist;
        controller = _controller;
        timelock = _timelock;
        btf = _btf;
    }//knownsec// 初始化合约，设置 governance、strategist、controller、timelock、btf 地址

    // **** Views ****

    function balanceOfWant() public view returns (uint256) {
        return IERC20(want).balanceOf(address(this));
    }

    function balanceOfPool() public view returns (uint256) {
        return IStakingRewards(rewards).balanceOf(address(this));
    }

    function balanceOf() public view returns (uint256) {
        return balanceOfWant().add(balanceOfPool());
    }

    function getName() external pure returns (string memory) {
        return "StrategyUniEthUsdcLp";
    }

    function getHarvestable() external view returns (uint256) {
        return IStakingRewards(rewards).earned(address(this));
    }

    // **** Setters ****

    function setKeepUNI(uint256 _keepUNI) external {
        require(msg.sender == governance, "!governance");
        keepUNI = _keepUNI;
    }

    function setWithdrawalFee(uint256 _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        withdrawalFee = _withdrawalFee;
    }

    function setPerformanceFee(uint256 _performanceFee) external {
        require(msg.sender == governance, "!governance");
        performanceFee = _performanceFee;
    }

    function setBurnFee(uint256 _burnFee) external {
        require(msg.sender == governance, "!governance");
        burnFee = _burnFee;
    }

    function setStrategist(address _strategist) external {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
    }

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

    function setTimelock(address _timelock) external {
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
```

```
    }

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

    // **** State Mutations ****

    function deposit() public {//knownsec//  流动性挖矿  转钱
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            IERC20(want).safeApprove(rewards, 0);
            IERC20(want).approve(rewards, _want);
            IStakingRewards(rewards).stake(_want);
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint256 balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Contoller only function for withdrawing for free
    // This is used to swap between vaults
    function freeWithdraw(uint256 _amount) external {// knownsec  仓库间代币置换
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }
        IERC20(want).safeTransfer(msg.sender, _amount);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {//knownsec//  提现资产到vault    量 _amount    控制器可用
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }

        address _vault = IController(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 (uint256 balance) {//knownsec//  提现所有资产, controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }

    function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec//  部分提现    内部使用
        IStakingRewards(rewards).withdraw(_amount);
        return _amount;
    }

    function brine() public {
        harvest();
    }

    function harvest() public {//knownsec//赚取流动性利息
```

```
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
        // i.e. will be be heavily frontrunned?
        //          if so, a new strategy will be deployed.

        // Collects UNI tokens
        IStakingRewards(rewards).getReward();
        uint256 _uni = IERC20(uni).balanceOf(address(this));
        if (_uni > 0) {
            if (keepUNI > 0) {
                // 10% is locked up for future gov
                uint256 _keepUNI = _uni.mul(keepUNI).div(keepUNIMax);
                IERC20(uni).safeTransfer(
                    IController(controller).devAddr(),
                    _keepUNI
                );
                _uni = _uni.sub(_keepUNI);
            }

            _swap(uni, weth, _uni);
        }

        // Swap half WETH for USDC
        uint256 _weth = IERC20(weth).balanceOf(address(this));
        if (_weth > 0) {
            // Burn some btfs first
            if (burnFee > 0) {
                uint256 _burnFee = _weth.mul(burnFee).div(burnMax);
                _swap(weth, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
                );
                _weth = _weth.sub(_burnFee);
            }

            _swap(weth, usdc, _weth.div(2));
        }

        // Adds in liquidity for ETH/USDC
        _weth = IERC20(weth).balanceOf(address(this));
        uint256 _usdc = IERC20(usdc).balanceOf(address(this));
        if (_weth > 0 && _usdc > 0) {
            IERC20(weth).safeApprove(univ2Router2, 0);
            IERC20(weth).safeApprove(univ2Router2, _weth);

            IERC20(usdc).safeApprove(univ2Router2, 0);
            IERC20(usdc).safeApprove(univ2Router2, _usdc);

            UniswapRouterV2(univ2Router2).addLiquidity(
                weth,
                usdc,
                _weth,
                _usdc,
                0,
                0,
                address(this),
                now + 60
            );

            // Donates DUST
            IERC20(weth).transfer(
                IController(controller).comAddr(),
                IERC20(weth).balanceOf(address(this))
            );
            IERC20(usdc).transfer(
                IController(controller).comAddr(),
                IERC20(usdc).balanceOf(address(this))
            );
        }

        // We want to get back UNI ETH/USDC LP tokens
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // Performance fee
            if (performanceFee > 0) {
                IERC20(want).safeTransfer(
                    IController(controller).comAddr(),
                    _want.mul(performanceFee).div(performanceMax)
                );
            }

            deposit();
        }
    }
```

```
    // Emergency function call
    function execute(address _target, bytes memory _data)
    public
    payable
    returns (bytes memory response)// knownsec  避险函数     timelock 地址可用
    {
        require(msg.sender == timelock, "!timelock");

        require(_target != address(0), "!target");

        // call contract in current context
        assembly {
            let succeeded := delegatecall(
            sub(gas(), 5000),
            _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
            )
            let size := returndatasize()

            response := mload(0x40)
            mstore(
            0x40,
            add(response, and(add(add(size, 0x20), 0x1f), not(0x1f)))
            )
            mstore(response, size)
            returndatacopy(add(response, 0x20), 0, size)

            switch iszero(succeeded)
            case 1 {
            // throw if delegatecall failed
                revert(add(response, 0x20), size)
            }
        }
    }

    // **** Internal functions ****
    function _swap(
        address _from,
        address _to,
        uint256 _amount
    ) internal {// knownsec  交换代币
        // Swap with uniswap
        IERC20(_from).safeApprove(univ2Router2, 0);
        IERC20(_from).safeApprove(univ2Router2, _amount);

        address[] memory path;

        if (_from == weth || _to == weth) {
            path = new address[](2);
            path[0] = _from;
            path[1] = _to;
        } else {
            path = new address[](3);
            path[0] = _from;
            path[1] = weth;
            path[2] = _to;
        }

        UniswapRouterV2(univ2Router2).swapExactTokensForTokens(
            _amount,
            0,
            path,
            address(this),
            now.add(60)
        );
    }
}
```

StrategyUniEthUsdtLp.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";
import "../interface/IStrategy.sol";
import "../interface/IStakingRewards.sol";
import "../interface/UniswapRouterV2.sol";
import "../interface/USDT.sol";
```

```
contract StrategyUniEthUsdtLp {
    // v2 Uses uniswap for less gas
    // We can roll back to v1 if the liquidity is there

    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint256;

    // Staking rewards address for ETH/USDT LP providers
    address
    public constant rewards = 0x6C3e4cb2E96B01F4b866965A91ed4437839A121a;

    // want eth/usdt lp tokens
    address public constant want = 0x0d4a11d5EEaaC28EC3F61d100daF4d40471f1852;

    // tokens we're farming
    address public constant uni = 0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984;

    // stablecoins
    address public constant usdt = 0xdAC17F958D2ee523a2206206994597C13D831ec7;

    // weth
    address public constant weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

    // dex
    address public univ2Router2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

    // Fees 5% in total
    // - 1.5%   keepUNI for development fund
    // - 2%     performanceFee for community fund
    // - 1.5%   used to burn/repurchase btfs
    uint256 public keepUNI = 150;
    uint256 public constant keepUNIMax = 10000;

    uint256 public performanceFee = 200;
    uint256 public constant performanceMax = 10000;

    uint256 public burnFee = 150;
    uint256 public constant burnMax = 10000;

    uint256 public withdrawalFee = 0;
    uint256 public constant withdrawalMax = 10000;

    address public governance;
    address public controller;
    address public strategist;
    address public timelock;
    address public btf;

    constructor(
        address _governance,
        address _strategist,
        address _controller,
        address _timelock,
        address _btf
    ) public {
        governance = _governance;
        strategist = _strategist;
        controller = _controller;
        timelock = _timelock;
        btf = _btf;
    }//knownsec// 初始化合约，设置 governance、strategist、controller、timelock、btf 地址

    // **** Views ****

    function balanceOfWant() public view returns (uint256) {
        return IERC20(want).balanceOf(address(this));
    }

    function balanceOfPool() public view returns (uint256) {
        return IStakingRewards(rewards).balanceOf(address(this));
    }

    function balanceOf() public view returns (uint256) {
        return balanceOfWant().add(balanceOfPool());
    }

    function getName() external pure returns (string memory) {
        return "StrategyUniEthUsdtLp";
    }

    function getHarvestable() external view returns (uint256) {
        return IStakingRewards(rewards).earned(address(this));
    }

    // **** Setters ****
```

```
    function setKeepUNI(uint256 _keepUNI) external {
        require(msg.sender == governance, "!governance");
        keepUNI = _keepUNI;
    }

    function setWithdrawalFee(uint256 _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        withdrawalFee = _withdrawalFee;
    }

    function setPerformanceFee(uint256 _performanceFee) external {
        require(msg.sender == governance, "!governance");
        performanceFee = _performanceFee;
    }

    function setBurnFee(uint256 _burnFee) external {
        require(msg.sender == governance, "!governance");
        burnFee = _burnFee;
    }

    function setStrategist(address _strategist) external {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
    }

    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 setTimelock(address _timelock) external {
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
    }

    // **** State Mutations ****

    function deposit() public {//knownsec// 流动性挖矿 转钱
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            IERC20(want).safeApprove(rewards, 0);
            IERC20(want).approve(rewards, _want);
            IStakingRewards(rewards).stake(_want);
        }
    }

    // Contoller only function for withdrawing for free
    // This is used to swap between vaults
    function freeWithdraw(uint256 _amount) external {// knownsec 仓库间代币置换
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }
        IERC20(want).safeTransfer(msg.sender, _amount);
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint256 balance) {//knownsec// 提现资产到vault 量
_amount 控制器可用
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }
```

```
        address _vault = IController(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 (uint256 balance) {//knownsec//  提现所有资产, controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }

    function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec//  部分提现   内部使用
        IStakingRewards(rewards).withdraw(_amount);
        return _amount;
    }

    function brine() public {
        harvest();
    }

    function harvest() public {//knownsec//赚取流动性利息
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
        // i.e. will be be heavily frontrunned?
        //          if so, a new strategy will be deployed.

        // Collects UNI tokens
        IStakingRewards(rewards).getReward();
        uint256 _uni = IERC20(uni).balanceOf(address(this));
        if (_uni > 0) {
            // 10% is locked up for future gov
            if (keepUNI > 0) {
                uint256 _keepUNI = _uni.mul(keepUNI).div(keepUNIMax);
                IERC20(uni).safeTransfer(
                    IController(controller).devAddr(),
                    _keepUNI
                );
                _uni = _uni.sub(_keepUNI);
            }

            _swap(uni, weth, _uni);
        }

        // Swap half WETH for USDT
        uint256 _weth = IERC20(weth).balanceOf(address(this));
        if (_weth > 0) {
            // Burn some btfs first
            if (burnFee > 0) {
                uint256 _burnFee = _weth.mul(burnFee).div(burnMax);
                _swap(weth, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
                );
                _weth = _weth.sub(_burnFee);
            }

            _swap(weth, usdt, _weth.div(2));
        }

        // Adds in liquidity for ETH/usdt
        _weth = IERC20(weth).balanceOf(address(this));
        uint256 _usdt = IERC20(usdt).balanceOf(address(this));
        if (_weth > 0 && _usdt > 0) {
            IERC20(weth).safeApprove(univ2Router2, 0);
            IERC20(weth).safeApprove(univ2Router2, _weth);

            USDT(usdt).approve(univ2Router2, _usdt);

            UniswapRouterV2(univ2Router2).addLiquidity(
                weth,
                usdt,
```

```
                _weth,
                _usdt,
                0,
                0,
                address(this),
                now + 60
            );

            // Donates DUST
            IERC20(weth).transfer(
                IController(controller).comAddr(),
                IERC20(weth).balanceOf(address(this))
            );
            USDT(usdt).transfer(
                IController(controller).comAddr(),
                IERC20(usdt).balanceOf(address(this))
            );
        }

        // We want to get back UNI ETH/usdt LP tokens
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // Performance fee
            if (performanceFee > 0) {
                IERC20(want).safeTransfer(
                    IController(controller).comAddr(),
                    _want.mul(performanceFee).div(performanceMax)
                );
            }

            deposit();
        }
    }

    // Emergency function call
    function execute(address _target, bytes memory _data)
    public
    payable
    returns (bytes memory response)// knownsec  避险函数     timelock 地址可用
    {
        require(msg.sender == timelock, "!timelock");

        require(_target != address(0), "!target");

        // call contract in current context
        assembly {
            let succeeded := delegatecall(
            sub(gas(), 5000),
            _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
            )
            let size := returndatasize()

            response := mload(0x40)
            mstore(
            0x40,
            add(response, and(add(add(size, 0x20), 0x1f), not(0x1f)))
            )
            mstore(response, size)
            returndatacopy(add(response, 0x20), 0, size)

            switch iszero(succeeded)
            case 1 {
            // throw if delegatecall failed
                revert(add(response, 0x20), size)
            }
        }
    }

    // **** Internal functions ****

    function _swap(
        address _from,
        address _to,
        uint256 _amount
    ) internal {// knownsec  交换代币
        // Swap with uniswap
        IERC20(_from).safeApprove(univ2Router2, 0);
        IERC20(_from).safeApprove(univ2Router2, _amount);

        address[] memory path;

        if (_from == weth || _to == weth) {
            path = new address[](2);
```

```
                path[0] = _from;
                path[1] = _to;
            } else {
                path = new address[](3);
                path[0] = _from;
                path[1] = weth;
                path[2] = _to;
            }

            UniswapRouterV2(univ2Router2).swapExactTokensForTokens(
                _amount,
                0,
                path,
                address(this),
                now.add(60)
            );
        }
}
```

StrategyUniEthWbtcLp.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";
import "../interface/IStrategy.sol";
import "../interface/IStakingRewards.sol";
import "../interface/UniswapRouterV2.sol";

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

    // Staking rewards address for ETH/WBTC LP providers
    address
    public constant rewards = 0xCA35e32e7926b96A9988f61d510E038108d8068e;

    // want eth/wbtc lp tokens
    address public constant want = 0xBb2b8038a1640196FbE3e38816F3e67Cba72D940;

    // tokens we're farming
    address public constant uni = 0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984;

    // stablecoins
    address public constant wbtc = 0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599;

    // weth
    address public constant weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

    // dex
    address public univ2Router2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

    // Fees 5% in total
    // - 1.5%     keepUNI for development fund
    // - 2%       performanceFee for community fund
    // - 1.5%     used to burn/repurchase btfs
    uint256 public keepUNI = 150;
    uint256 public constant keepUNIMax = 10000;

    uint256 public performanceFee = 200;
    uint256 public constant performanceMax = 10000;

    uint256 public burnFee = 150;
    uint256 public constant burnMax = 10000;

    uint256 public withdrawalFee = 0;
    uint256 public constant withdrawalMax = 10000;

    address public governance;
    address public controller;
    address public strategist;
    address public timelock;
    address public btf;

    constructor(
        address _governance,
        address _strategist,
        address _controller,
        address _timelock,
        address _btf
```

```
) public {
        governance = _governance;
        strategist = _strategist;
        controller = _controller;
        timelock = _timelock;
        btf = _btf;
}//knownsec// 初始化合约，设置 governance、strategist、controller、timelock、btf 地址

// **** Views ****

function balanceOfWant() public view returns (uint256) {
        return IERC20(want).balanceOf(address(this));
}

function balanceOfPool() public view returns (uint256) {
        return IStakingRewards(rewards).balanceOf(address(this));
}

function balanceOf() public view returns (uint256) {
        return balanceOfWant().add(balanceOfPool());
}

function getName() external pure returns (string memory) {
        return "StrategyUniEthWbtcLp";
}

function getHarvestable() external view returns (uint256) {
        return IStakingRewards(rewards).earned(address(this));
}

// **** Setters ****

function setKeepUNI(uint256 _keepUNI) external {
        require(msg.sender == governance, "!governance");
        keepUNI = _keepUNI;
}

function setWithdrawalFee(uint256 _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        withdrawalFee = _withdrawalFee;
}

function setPerformanceFee(uint256 _performanceFee) external {
        require(msg.sender == governance, "!governance");
        performanceFee = _performanceFee;
}

function setBurnFee(uint256 _burnFee) external {
        require(msg.sender == governance, "!governance");
        burnFee = _burnFee;
}

function setStrategist(address _strategist) external {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
}

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

function setTimelock(address _timelock) external {
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
}

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

// **** State Mutations ****

function deposit() public {//knownsec// 流动性挖矿 转钱
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
                IERC20(want).safeApprove(rewards, 0);
                IERC20(want).approve(rewards, _want);
                IStakingRewards(rewards).stake(_want);
        }
}

// Controller only function for creating additional rewards from dust
function withdraw(IERC20 _asset) external returns (uint256 balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
```

```
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Contoller only function for withdrawing for free
    // This is used to swap between vaults
    function freeWithdraw(uint256 _amount) external {// knownsec  仓库间代币置换
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }
        IERC20(want).safeTransfer(msg.sender, _amount);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {//knownsec//  提现资产到vault     量 _amount    控制器可用
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }

        address _vault = IController(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 (uint256 balance) {//knownsec//  提现所有资产, controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }

    function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec//  部分提现    内部使用
        IStakingRewards(rewards).withdraw(_amount);
        return _amount;
    }

    function brine() public {
        harvest();
    }

    function harvest() public {//knownsec//赚取流动性利息
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
        // i.e. will be be heavily frontrunned?
        //          if so, a new strategy will be deployed.

        // Collects UNI tokens
        IStakingRewards(rewards).getReward();
        uint256 _uni = IERC20(uni).balanceOf(address(this));
        if (_uni > 0) {
            // 10% is locked up for future gov
            if (keepUNI > 0) {
                uint256 _keepUNI = _uni.mul(keepUNI).div(keepUNIMax);
                IERC20(uni).safeTransfer(
                    IController(controller).devAddr(),
                    _keepUNI
                );
                _uni = _uni.sub(_keepUNI);
            }
            _swap(uni, weth, _uni);
        }
```

```
        // Swap half WETH for WBTC
        uint256 _weth = IERC20(weth).balanceOf(address(this));
        if (_weth > 0) {
            // Burn some btfs first
            if (burnFee > 0) {
                uint256 _burnFee = _weth.mul(burnFee).div(burnMax);
                _swap(weth, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
                );
                _weth = _weth.sub(_burnFee);
            }

            _swap(weth, wbtc, _weth.div(2));
        }

        // Adds in liquidity for ETH/WBTC
        _weth = IERC20(weth).balanceOf(address(this));
        uint256 _wbtc = IERC20(wbtc).balanceOf(address(this));
        if (_weth > 0 && _wbtc > 0) {
            IERC20(weth).safeApprove(univ2Router2, 0);
            IERC20(weth).safeApprove(univ2Router2, _weth);

            IERC20(wbtc).safeApprove(univ2Router2, 0);
            IERC20(wbtc).safeApprove(univ2Router2, _wbtc);

            UniswapRouterV2(univ2Router2).addLiquidity(
                weth,
                wbtc,
                _weth,
                _wbtc,
                0,
                0,
                address(this),
                now + 60
            );

            // Donates DUST
            IERC20(weth).transfer(
                IController(controller).comAddr(),
                IERC20(weth).balanceOf(address(this))
            );
            IERC20(wbtc).transfer(
                IController(controller).comAddr(),
                IERC20(wbtc).balanceOf(address(this))
            );
        }

        // We want to get back UNI ETH/WBTC LP tokens
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // Performance fee
            if (performanceFee > 0) {
                IERC20(want).safeTransfer(
                    IController(controller).comAddr(),
                    _want.mul(performanceFee).div(performanceMax)
                );
            }

            deposit();
        }
    }

    // Emergency function call
    function execute(address _target, bytes memory _data)
    public
    payable
    returns (bytes memory response)// knownsec  避险函数     timelock 地址可用
    {
        require(msg.sender == timelock, "!timelock");

        require(_target != address(0), "!target");

        // call contract in current context
        assembly {
            let succeeded := delegatecall(
            sub(gas(), 5000),
            _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
            )
            let size := returndatasize()

            response := mload(0x40)
```

```
                mstore(
                0x40,
                add(response, and(add(add(size, 0x20), 0x1f), not(0x1f)))
                )
                mstore(response, size)
                returndatacopy(add(response, 0x20), 0, size)

                switch iszero(succeeded)
                case 1 {
                // throw if delegatecall failed
                      revert(add(response, 0x20), size)
                }
            }
        }

        // **** Internal functions ****
        function _swap(
            address _from,
            address _to,
            uint256 _amount
        ) internal {// Knownsec  交换代币
            // Swap with uniswap
            IERC20(_from).safeApprove(univ2Router2, 0);
            IERC20(_from).safeApprove(univ2Router2, _amount);

            address[] memory path;

            if (_from == weth || _to == weth) {
                path = new address[](2);
                path[0] = _from;
                path[1] = _to;
            } else {
                path = new address[](3);
                path[0] = _from;
                path[1] = weth;
                path[2] = _to;
            }

            UniswapRouterV2(univ2Router2).swapExactTokensForTokens(
                _amount,
                0,
                path,
                address(this),
                now.add(60)
            );
        }
}
```

StrategyYfvDai.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";
import "../interface/IStrategy.sol";
import "../interface/IStakingRewards.sol";
import "../interface/UniswapRouterV2.sol";
import "../interface/IYfvRewards.sol";

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

    // Staking rewards address for dai LP providers
    address public constant rewards = 0xC2D55CE14a8e04AEF9B6bCfD105079b63C6a0AC8;

    // want dai stablecoins
    address public constant want = 0x6B175474E89094C44Da98b954EedeAC495271d0F;

    // tokens we're farming
    address public constant yfv = 0x45f24BaEef268BB6d63AEe5129015d69702BCDfa;

    // weth
    address public constant weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

    // yfv vUSD
    address public vUSD = 0x1B8E12F839BD4e73A47adDF76cF7F0097d74c14C;

    // yfv vETH
    address public vETH = 0x76A034e76Aa835363056dd418611E4f81870f16e;
```

```
// dex
address public univ2Router2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

// Fees 5% in total
// - 1.5%     keepYFV for development fund
// - 2%       performanceFee for community fund
// - 1.5%     used to burn/repurchase btfs
uint256 public keepYFV = 150;
uint256 public constant keepYFVMax = 10000;

uint256 public performanceFee = 200;
uint256 public constant performanceMax = 10000;

uint256 public burnFee = 150;
uint256 public constant burnMax = 10000;

uint256 public withdrawalFee = 0;
uint256 public constant withdrawalMax = 10000;

address public governance;
address public controller;
address public strategist;
address public timelock;
address public btf;

constructor(
        address _governance,
        address _strategist,
        address _controller,
        address _timelock,
        address _btf
) public {
        governance = _governance;
        strategist = _strategist;
        controller = _controller;
        timelock = _timelock;
        btf = _btf;
}//knownsec// 初始化合约，设置 governance、strategist、controller、timelock、btf 地址

// **** Views ****

function balanceOfWant() public view returns (uint256) {
        return IERC20(want).balanceOf(address(this));
}

function balanceOfPool() public view returns (uint256) {
        return IYfvRewards(rewards).balanceOf(want, address(this));
}

function balanceOf() public view returns (uint256) {
        return balanceOfWant().add(balanceOfPool());
}

function getName() external pure returns (string memory) {
        return "StrategyYfvDai";
}

function getHarvestable() external view returns (uint256) {
        return IYfvRewards(rewards).earned(address(this));
}
// **** Setters ****

function setKeepYFV(uint256 _keepYFV) external {
        require(msg.sender == governance, "!governance");
        keepYFV = _keepYFV;
}

function setWithdrawalFee(uint256 _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        withdrawalFee = _withdrawalFee;
}

function setPerformanceFee(uint256 _performanceFee) external {
        require(msg.sender == governance, "!governance");
        performanceFee = _performanceFee;
}

function setBurnFee(uint256 _burnFee) external {
        require(msg.sender == governance, "!governance");
        burnFee = _burnFee;
}

function setStrategist(address _strategist) external {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
```

```
    }

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

    function setTimelock(address _timelock) external {
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
    }

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

    // **** State Mutations ****

    function deposit() public {//knownsec// 流动性挖矿 转钱
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            IERC20(want).safeApprove(rewards, 0);
            IERC20(want).approve(rewards, _want);
            IYfvRewards(rewards).stake(want, _want, IController(controller).comAddr());
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint256 balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Contoller only function for withdrawing for free
    // This is used to swap between vaults
    function freeWithdraw(uint256 _amount) external {// knownsec 仓库间代币置换
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }
        IERC20(want).safeTransfer(msg.sender, _amount);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {//knownsec// 提现资产到vault 量 _amount 控制器可用
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }

        address _vault = IController(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 (uint256 balance) {//knownsec// 提现所有资产, controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }
```

```
function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec// 部分提现  内部使用
        IYfvRewards(rewards).withdraw(want, _amount);
        return _amount;
    }

    function brine() public {
        harvest();
    }

    function harvest() public {//knownsec//赚取流动性利息
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
        // i.e. will be be heavily frontrunned?
        //          if so, a new strategy will be deployed.

        // Collects YFV tokens
        IYfvRewards(rewards).getReward();
        uint256 _yfv = IERC20(yfv).balanceOf(address(this));
        if (_yfv > 0) {
            if (keepYFV > 0) {
                // some yfv locked up for future gov
                uint256 _keepYFV = _yfv.mul(keepYFV).div(keepYFVMax);
                IERC20(yfv).safeTransfer(
                    IController(controller).devAddr(),
                    _keepYFV
                );
                _yfv = _yfv.sub(_keepYFV);
            }

            if (burnFee > 0) {
                // Burn some btf
                uint256 _burnFee =  _yfv.mul(burnFee).div(burnMax);
                 _swap(yfv, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
                );
                _yfv = _yfv.sub(_burnFee);
            }

            // swap for want
            _swap(yfv, want, _yfv);
        }

        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // Performance fee
            if (performanceFee > 0) {
                IERC20(want).safeTransfer(
                    IController(controller).comAddr(),
                    _want.mul(performanceFee).div(performanceMax)
                );
            }

            deposit();
        }
    }

    // Emergency function call
    function execute(address _target, bytes memory _data)
    public
    payable
    returns (bytes memory response)// knownsec  避险函数     timelock 地址可用
    {
        require(msg.sender == timelock, "!timelock");

        require(_target != address(0), "!target");

        // call contract in current context
        assembly {
            let succeeded := delegatecall(
            sub(gas(), 5000),
            _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
            )
            let size := returndatasize()

            response := mload(0x40)
            mstore(
            0x40,
            add(response, and(add(add(size, 0x20), 0x1f), not(0x1f)))
            )
            mstore(response, size)
```

```
                returndatacopy(add(response, 0x20), 0, size)

                switch iszero(succeeded)
                case 1 {
                // throw if delegatecall failed
                    revert(add(response, 0x20), size)
                }
            }
        }

        // **** Internal functions ****
        function _swap(
            address _from,
            address _to,
            uint256 _amount
        ) internal {// Knownsec  交换代币
            // Swap with uniswap
            IERC20(_from).safeApprove(univ2Router2, 0);
            IERC20(_from).safeApprove(univ2Router2, _amount);

            address[] memory path;

            if (_from == weth || _to == weth) {
                path = new address[](2);
                path[0] = _from;
                path[1] = _to;
            } else {
                path = new address[](3);
                path[0] = _from;
                path[1] = weth;
                path[2] = _to;
            }

            UniswapRouterV2(univ2Router2).swapExactTokensForTokens(
                _amount,
                0,
                path,
                address(this),
                now.add(60)
            );
        }
}
```

StrategyYfvTusd.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";
import "../interface/IStrategy.sol";
import "../interface/IStakingRewards.sol";
import "../interface/UniswapRouterV2.sol";
import "../interface/IYfvRewards.sol";

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

    // Staking rewards address for tusd LP providers
    address public constant rewards = 0xC2D55CE14a8e04AEF9B6bCfD105079b63C6a0AC8;

    // want tusd stablecoins
    address public constant want = 0x0000000000085d4780B73119b644AE5ecd22b376;

    // tokens we're farming
    address public constant yfv = 0x45f24BaEef268BB6d63AEe5129015d69702BCDfa;

    // weth
    address public constant weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

    // yfv vUSD
    address public vUSD = 0x1B8E12F839BD4e73A47adDF76cF7F0097d74c14C;

    // yfv vETH
    address public vETH = 0x76A034e76Aa835363056dd418611E4f81870f16e;

    // dex
    address public univ2Router2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

    // Fees 5% in total
```

```
// - 1.5%     keepYFV for development fund
// - 2%       performanceFee for community fund
// - 1.5%     used to burn/repurchase btfs
uint256 public keepYFV = 150;
uint256 public constant keepYFVMax = 10000;

uint256 public performanceFee = 200;
uint256 public constant performanceMax = 10000;

uint256 public burnFee = 150;
uint256 public constant burnMax = 10000;

uint256 public withdrawalFee = 0;
uint256 public constant withdrawalMax = 10000;

address public governance;
address public controller;
address public strategist;
address public timelock;
address public btf;

constructor(
        address _governance,
        address _strategist,
        address _controller,
        address _timelock,
        address _btf
) public {
        governance = _governance;
        strategist = _strategist;
        controller = _controller;
        timelock = _timelock;
        btf = _btf;
}//knownsec// 初始化合约，设置 governance、strategist、controller、timelock、btf 地址

// **** Views ****

function balanceOfWant() public view returns (uint256) {
        return IERC20(want).balanceOf(address(this));
}

function balanceOfPool() public view returns (uint256) {
        return IYfvRewards(rewards).balanceOf(want, address(this));
}

function balanceOf() public view returns (uint256) {
        return balanceOfWant().add(balanceOfPool());
}

function getName() external pure returns (string memory) {
        return "StrategyYfvTusd";
}

function getHarvestable() external view returns (uint256) {
        return IYfvRewards(rewards).earned(address(this));
}

// **** Setters ****

function setKeepYFV(uint256 _keepYFV) external {
        require(msg.sender == governance, "!governance");
        keepYFV = _keepYFV;
}

function setWithdrawalFee(uint256 _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        withdrawalFee = _withdrawalFee;
}

function setPerformanceFee(uint256 _performanceFee) external {
        require(msg.sender == governance, "!governance");
        performanceFee = _performanceFee;
}

function setBurnFee(uint256 _burnFee) external {
        require(msg.sender == governance, "!governance");
        burnFee = _burnFee;
}

function setStrategist(address _strategist) external {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
}

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

```
    }

    function setTimelock(address _timelock) external {
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
    }

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

    // **** State Mutations ****

    function deposit() public {//knownsec// 流动性挖矿 转钱
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            IERC20(want).safeApprove(rewards, 0);
            IERC20(want).approve(rewards, _want);
            IYfvRewards(rewards).stake(want, _want, IController(controller).comAddr());
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint256 balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Contoller only function for withdrawing for free
    // This is used to swap between vaults
    function freeWithdraw(uint256 _amount) external {// knownsec 仓库间代币置换
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }
        IERC20(want).safeTransfer(msg.sender, _amount);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {//knownsec// 提现资产到vault 量 _amount 控制器可用
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }

        address _vault = IController(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 (uint256 balance) {//knownsec// 提现所有资产, controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }

    function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec// 部分提现 内部使用
        IYfvRewards(rewards).withdraw(want, _amount);
        return _amount;
    }
```

```
function brine() public {
        harvest();
    }

    function harvest() public {//knownsec//赚取流动性利息
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
        // i.e. will be be heavily frontrunned?
        //          if so, a new strategy will be deployed.

        // Collects YFV tokens
        IYfvRewards(rewards).getReward();
        uint256 _yfv = IERC20(yfv).balanceOf(address(this));
        if (_yfv > 0) {
            if (keepYFV > 0) {
                // some yfv locked up for future gov
                uint256 _keepYFV = _yfv.mul(keepYFV).div(keepYFVMax);
                IERC20(yfv).safeTransfer(
                    IController(controller).devAddr(),
                    _keepYFV
                );
                _yfv = _yfv.sub(_keepYFV);
            }

            if (burnFee > 0) {
                // Burn some btf
                uint256 _burnFee = _yfv.mul(burnFee).div(burnMax);
                _swap(yfv, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
                );
                _yfv = _yfv.sub(_burnFee);
            }

            // swap for want
            _swap(yfv, want, _yfv);
        }

        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // Performance fee
            if (performanceFee > 0) {
                IERC20(want).safeTransfer(
                    IController(controller).comAddr(),
                    _want.mul(performanceFee).div(performanceMax)
                );
            }

            deposit();
        }
    }

    // Emergency function call
    function execute(address _target, bytes memory _data)
    public
    payable
    returns (bytes memory response)// knownsec  避险函数     timelock 地址可用
    {
        require(msg.sender == timelock, "!timelock");

        require(_target != address(0), "!target");

        // call contract in current context
        assembly {
            let succeeded := delegatecall(
            sub(gas(), 5000),
            _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
            )
            let size := returndatasize()

            response := mload(0x40)
            mstore(
            0x40,
            add(response, and(add(add(size, 0x20), 0x1f), not(0x1f)))
            )
            mstore(response, size)
            returndatacopy(add(response, 0x20), 0, size)

            switch iszero(succeeded)
            case 1 {
            // throw if delegatecall failed
```

```
                    revert(add(response, 0x20), size)
                }
            }
        }

        // **** Internal functions ****
        function _swap(
            address _from,
            address _to,
            uint256 _amount
        ) internal {// knownsec  交换代币
            // Swap with uniswap
            IERC20(_from).safeApprove(univ2Router2, 0);
            IERC20(_from).safeApprove(univ2Router2, _amount);

            address[] memory path;

            if (_from == weth || _to == weth) {
                path = new address[](2);
                path[0] = _from;
                path[1] = _to;
            } else {
                path = new address[](3);
                path[0] = _from;
                path[1] = weth;
                path[2] = _to;
            }

            UniswapRouterV2(univ2Router2).swapExactTokensForTokens(
                _amount,
                0,
                path,
                address(this),
                now.add(60)
            );
        }
}
```

StrategyYfvUsdc.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";
import "../interface/IStrategy.sol";
import "../interface/IStakingRewards.sol";
import "../interface/UniswapRouterV2.sol";
import "../interface/IYfvRewards.sol";

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

    // Staking rewards address for usdc LP providers
    address public constant rewards = 0xC2D55CE14a8e04AEF9B6bCfD105079b63C6a0AC8;

    // want usdc stablecoins
    address public constant want = 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48;

    // tokens we're farming
    address public constant yfv = 0x45f24BaEef268BB6d63AEe5129015d69702BCDfa;

    // weth
    address public constant weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

    // yfv vUSD
    address public vUSD = 0x1B8E12F839BD4e73A47adDF76cF7F0097d74c14C;

    // yfv vETH
    address public vETH = 0x76A034e76Aa835363056dd418611E4f81870f16e;

    // dex
    address public univ2Router2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

    // Fees 5% in total
    // - 1.5%   keepYFV for development fund
    // - 2%     performanceFee for community fund
    // - 1.5%   used to burn/repurchase btfs
    uint256 public keepYFV = 150;
```

```
    uint256 public constant keepYFVMax = 10000;

    uint256 public performanceFee = 200;
    uint256 public constant performanceMax = 10000;

    uint256 public burnFee = 150;
    uint256 public constant burnMax = 10000;

    uint256 public withdrawalFee = 0;
    uint256 public constant withdrawalMax = 10000;

    address public governance;
    address public controller;
    address public strategist;
    address public timelock;
    address public btf;

    constructor(
        address _governance,
        address _strategist,
        address _controller,
        address _timelock,
        address _btf
    ) public {
        governance = _governance;
        strategist = _strategist;
        controller = _controller;
        timelock = _timelock;
        btf = _btf;
    }//knownsec// 初始化合约，设置 governance、strategist、controller、timelock、btf 地址

    // **** Views ****

    function balanceOfWant() public view returns (uint256) {
        return IERC20(want).balanceOf(address(this));
    }

    function balanceOfPool() public view returns (uint256) {
        return IYfvRewards(rewards).balanceOf(want, address(this));
    }

    function balanceOf() public view returns (uint256) {
        return balanceOfWant().add(balanceOfPool());
    }

    function getName() external pure returns (string memory) {
        return "StrategyYfvUsdc";
    }

    function getHarvestable() external view returns (uint256) {
        return IYfvRewards(rewards).earned(address(this));
    }

    // **** Setters ****

    function setKeepYFV(uint256 _keepYFV) external {
        require(msg.sender == governance, "!governance");
        keepYFV = _keepYFV;
    }

    function setWithdrawalFee(uint256 _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        withdrawalFee = _withdrawalFee;
    }

    function setPerformanceFee(uint256 _performanceFee) external {
        require(msg.sender == governance, "!governance");
        performanceFee = _performanceFee;
    }

    function setBurnFee(uint256 _burnFee) external {
        require(msg.sender == governance, "!governance");
        burnFee = _burnFee;
    }

    function setStrategist(address _strategist) external {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
    }

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

    function setTimelock(address _timelock) external {
        require(msg.sender == timelock, "!timelock");
```

```
        timelock = _timelock;
    }

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

    // **** State Mutations ****

    function deposit() public {//knownsec// 流动性挖矿 转钱
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            IERC20(want).safeApprove(rewards, 0);
            IERC20(want).approve(rewards, _want);
            IYfvRewards(rewards).stake(want, _want, IController(controller).comAddr());
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint256 balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Contoller only function for withdrawing for free
    // This is used to swap between vaults
    function freeWithdraw(uint256 _amount) external {// knownsec 仓库间代币置换
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }
        IERC20(want).safeTransfer(msg.sender, _amount);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {//knownsec// 提现资产到 vault 量 _amount 控制器可用
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }

        address _vault = IController(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 (uint256 balance) {//knownsec// 提现所有资产, controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }

    function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec// 部分提现 内部使用
        IYfvRewards(rewards).withdraw(want, _amount);
        return _amount;
    }

    function brine() public {
        harvest();
    }
```

```
function harvest() public {//knownsec//赚取流动性利息
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
        // i.e. will be be heavily frontrunned?
        //          if so, a new strategy will be deployed.

        // Collects YFV tokens
        IYfvRewards(rewards).getReward();
        uint256 _yfv = IERC20(yfv).balanceOf(address(this));
        if (_yfv > 0) {
            if (keepYFV > 0) {
                // some yfv locked up for future gov
                uint256 _keepYFV = _yfv.mul(keepYFV).div(keepYFVMax);
                IERC20(yfv).safeTransfer(
                    IController(controller).devAddr(),
                    _keepYFV
                );
                _yfv = _yfv.sub(_keepYFV);
            }

            if (burnFee > 0) {
                // Burn some btf
                uint256 _burnFee = _yfv.mul(burnFee).div(burnMax);
                _swap(yfv, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
                );
                _yfv = _yfv.sub(_burnFee);
            }

            // swap for want
            _swap(yfv, want, _yfv);
        }

        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // Performance fee
            if (performanceFee > 0) {
                IERC20(want).safeTransfer(
                    IController(controller).comAddr(),
                    _want.mul(performanceFee).div(performanceMax)
                );
            }

            deposit();
        }
    }

    // Emergency function call
    function execute(address _target, bytes memory _data)
    public
    payable
    returns (bytes memory response)// knownsec  避险函数     timelock 地址可用
    {
        require(msg.sender == timelock, "!timelock");

        require(_target != address(0), "!target");

        // call contract in current context
        assembly {
            let succeeded := delegatecall(
            sub(gas(), 5000),
              _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
            )
            let size := returndatasize()

            response := mload(0x40)
            mstore(
            0x40,
            add(response, and(add(add(size, 0x20), 0x1f), not(0x1f)))
            )
            mstore(response, size)
            returndatacopy(add(response, 0x20), 0, size)

            switch iszero(succeeded)
            case 1 {
            // throw if delegatecall failed
                revert(add(response, 0x20), size)
            }
        }
    }
```

```
    // **** Internal functions ****
    function _swap(
        address _from,
        address _to,
        uint256 _amount
    ) internal {// knownsec  交换代币
        // Swap with uniswap
        IERC20(_from).safeApprove(univ2Router2, 0);
        IERC20(_from).safeApprove(univ2Router2, _amount);

        address[] memory path;

        if (_from == weth || _to == weth) {
            path = new address[](2);
            path[0] = _from;
            path[1] = _to;
        } else {
            path = new address[](3);
            path[0] = _from;
            path[1] = weth;
            path[2] = _to;
        }

        UniswapRouterV2(univ2Router2).swapExactTokensForTokens(
            _amount,
            0,
            path,
            address(this),
            now.add(60)
        );
    }
}
```

StrategyYfvUsdt.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

import "../interface/IController.sol";
import "../interface/IStrategy.sol";
import "../interface/IStakingRewards.sol";
import "../interface/UniswapRouterV2.sol";
import "../interface/IYfvRewards.sol";
import "../interface/USDT.sol";

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

    // Staking rewards address for usdt LP providers
    address public constant rewards = 0xC2D55CE14a8e04AEF9B6bCfD105079b63C6a0AC8;

    // want usdt stablecoins
    address public constant want = 0xdAC17F958D2ee523a2206206994597C13D831ec7;

    // tokens we're farming
    address public constant yfv = 0x45f24BaEef268BB6d63AEe5129015d69702BCDfa;

    // weth
    address public constant weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;

    // yfv vUSD
    address public vUSD = 0x1B8E12F839BD4e73A47adDF76cF7F0097d74c14C;

    // yfv vETH
    address public vETH = 0x76A034e76Aa835363056dd418611E4f81870f16e;

    // dex
    address public univ2Router2 = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;

    // Fees 5% in total
    // - 1.5%     keepYFV for development fund
    // - 2%       performanceFee for community fund
    // - 1.5%     used to burn/repurchase btfs
    uint256 public keepYFV = 150;
    uint256 public constant keepYFVMax = 10000;

    uint256 public performanceFee = 200;
    uint256 public constant performanceMax = 10000;
```

```
uint256 public burnFee = 150;
uint256 public constant burnMax = 10000;

uint256 public withdrawalFee = 0;
uint256 public constant withdrawalMax = 10000;

address public governance;
address public controller;
address public strategist;
address public timelock;
address public btf;

constructor(
        address _governance,
        address _strategist,
        address _controller,
        address _timelock,
        address _btf
) public {
        governance = _governance;
        strategist = _strategist;
        controller = _controller;
        timelock = _timelock;
        btf = _btf;
}//knownsec// 初始化合约，设置 governance、strategist、controller、timelock、btf 地址

// **** Views ****

function balanceOfWant() public view returns (uint256) {
        return IERC20(want).balanceOf(address(this));
}

function balanceOfPool() public view returns (uint256) {
        return IYfvRewards(rewards).balanceOf(want, address(this));
}

function balanceOf() public view returns (uint256) {
        return balanceOfWant().add(balanceOfPool());
}

function getName() external pure returns (string memory) {
        return "StrategyYfvUsdt";
}

function getHarvestable() external view returns (uint256) {
        return IYfvRewards(rewards).earned(address(this));
}

// **** Setters ****

function setKeepYFV(uint256 _keepYFV) external {
        require(msg.sender == governance, "!governance");
        keepYFV = _keepYFV;
}

function setWithdrawalFee(uint256 _withdrawalFee) external {
        require(msg.sender == governance, "!governance");
        withdrawalFee = _withdrawalFee;
}

function setPerformanceFee(uint256 _performanceFee) external {
        require(msg.sender == governance, "!governance");
        performanceFee = _performanceFee;
}

function setBurnFee(uint256 _burnFee) external {
        require(msg.sender == governance, "!governance");
        burnFee = _burnFee;
}

function setStrategist(address _strategist) external {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
}

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

function setTimelock(address _timelock) external {
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
}

function setController(address _controller) external {
```

```
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }

    // **** State Mutations ****

    function deposit() public {//knownsec//  流动性挖矿  转钱
        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            IERC20(want).safeApprove(rewards, 0);
            IERC20(want).approve(rewards, _want);
            IYfvRewards(rewards).stake(want, _want, IController(controller).comAddr());
        }
    }

    // Controller only function for creating additional rewards from dust
    function withdraw(IERC20 _asset) external returns (uint256 balance) {
        require(msg.sender == controller, "!controller");
        require(want != address(_asset), "want");
        balance = _asset.balanceOf(address(this));
        _asset.safeTransfer(controller, balance);
    }

    // Contoller only function for withdrawing for free
    // This is used to swap between vaults
    function freeWithdraw(uint256 _amount) external {// knownsec  仓库间代币置换
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }
        IERC20(want).safeTransfer(msg.sender, _amount);
    }

    // Withdraw partial funds, normally used with a vault withdrawal
    function withdraw(uint256 _amount) external {//knownsec//  提现资产到 vault     量 _amount    控制器可用
        require(msg.sender == controller, "!controller");
        uint256 _balance = IERC20(want).balanceOf(address(this));
        if (_balance < _amount) {
            _amount = _withdrawSome(_amount.sub(_balance));
            _amount = _amount.add(_balance);
        }

        if (withdrawalFee > 0) {
            uint256 _fee = _amount.mul(withdrawalFee).div(withdrawalMax);
            IERC20(want).safeTransfer(IController(controller).comAddr(), _fee);
            _amount = _amount.sub(_fee);
        }

        address _vault = IController(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 (uint256 balance) {//knownsec//  提现所有资产, controller 可用
        require(msg.sender == controller, "!controller");
        _withdrawAll();

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

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

    function _withdrawAll() internal {
        _withdrawSome(balanceOfPool());
    }

    function _withdrawSome(uint256 _amount) internal returns (uint256) {//knownsec//  部分提现    内部使用
        IYfvRewards(rewards).withdraw(want, _amount);
        return _amount;
    }

    function brine() public {
        harvest();
    }

    function harvest() public {//knownsec//赚取流动性利息
        // Anyone can harvest it at any given time.
        // I understand the possibility of being frontrun
        // But ETH is a dark forest, and I wanna see how this plays out
```

```
        // i.e. will be be heavily frontrunned?
        //          if so, a new strategy will be deployed.

        // Collects YFV tokens
        IYfvRewards(rewards).getReward();
        uint256 _yfv = IERC20(yfv).balanceOf(address(this));
        if (_yfv > 0) {
            if (keepYFV > 0) {
                // some yfv locked up for future gov
                uint256 _keepYFV = _yfv.mul(keepYFV).div(keepYFVMax);
                IERC20(yfv).safeTransfer(
                    IController(controller).devAddr(),
                    _keepYFV
                );
                _yfv = _yfv.sub(_keepYFV);
            }

            if (burnFee > 0) {
                // Burn some btf
                uint256 _burnFee = _yfv.mul(burnFee).div(burnMax);
                _swap(yfv, btf, _burnFee);
                IERC20(btf).transfer(
                    IController(controller).burnAddr(),
                    IERC20(btf).balanceOf(address(this))
                );
                _yfv = _yfv.sub(_burnFee);
            }

            // swap for want
            _swap(yfv, want, _yfv);
        }

        uint256 _want = IERC20(want).balanceOf(address(this));
        if (_want > 0) {
            // Performance fee
            if (performanceFee > 0) {
                IERC20(want).safeTransfer(
                    IController(controller).comAddr(),
                    _want.mul(performanceFee).div(performanceMax)
                );
            }

            deposit();
        }
    }

    // Emergency function call
    function execute(address _target, bytes memory _data)
    public
    payable
    returns (bytes memory response)// knownsec  避险函数     timelock 地址可用
    {
        require(msg.sender == timelock, "!timelock");

        require(_target != address(0), "!target");

        // call contract in current context
        assembly {
            let succeeded := delegatecall(
            sub(gas(), 5000),
            _target,
            add(_data, 0x20),
            mload(_data),
            0,
            0
            )
            let size := returndatasize()

            response := mload(0x40)
            mstore(
            0x40,
            add(response, and(add(add(size, 0x20), 0x1f), not(0x1f)))
            )
            mstore(response, size)
            returndatacopy(add(response, 0x20), 0, size)

            switch iszero(succeeded)
            case 1 {
            // throw if delegatecall failed
                revert(add(response, 0x20), size)
            }
        }
    }

    // **** Internal functions ****
    function _swap(
        address _from,
```

```
        address _to,
        uint256 _amount
    ) internal {// knownsec 交换代币
        // Swap with uniswap
        IERC20(_from).safeApprove(univ2Router2, 0);
        IERC20(_from).safeApprove(univ2Router2, _amount);

        address[] memory path;

        if (_from == weth || _to == weth) {
            path = new address[](2);
            path[0] = _from;
            path[1] = _to;
        } else {
            path = new address[](3);
            path[0] = _from;
            path[1] = weth;
            path[2] = _to;
        }

        UniswapRouterV2(univ2Router2).swapExactTokensForTokens(
            _amount,
            0,
            path,
            address(this),
            now.add(60)
        );
    }
}
```

BTFReferral.sol

```
pragma solidity ^0.6.7;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract BTFReferral {
    mapping(address => address) public referrers; // account_address -> referrer_address
    mapping(address => uint256) public referredCount; // referrer_address -> num_of_referred

    event Referral(address indexed referrer, address indexed farmer);

    // Standard contract ownership transfer.
    address public owner;
    address private nextOwner;

    mapping(address => bool) public isAdmin;// knownsec 管理员数组

    constructor () public {
        owner = msg.sender;
    }// knownsec 初始化 owner 最高权限

    // Standard modifier on methods invokable only by contract owner.
    modifier onlyOwner {
        require(msg.sender == owner, "OnlyOwner methods called by non-owner.");
        _;
    }

    modifier onlyAdmin {
        require(isAdmin[msg.sender], "OnlyAdmin methods called by non-admin.");
        _;
    }

    // Standard contract ownership transfer implementation,
    function approveNextOwner(address _nextOwner) external onlyOwner {// knownsec 所属权限转让 原
Owner 审批 原 Owner 可用
        require(_nextOwner != owner, "Cannot approve current owner.");
        nextOwner = _nextOwner;
    }

    function acceptNextOwner() external {// knownsec 所属权限转让 新 Owner 接受 新 Owner 可用
require 校验
        require(msg.sender == nextOwner, "Can only accept preapproved new owner.");
        owner = nextOwner;
    }

    function setReferrer(address farmer, address referrer) public onlyAdmin {
        if (referrers[farmer] == address(0) && referrer != address(0)) {
            referrers[farmer] = referrer;
            referredCount[referrer] += 1;
            emit Referral(referrer, farmer);
        }
    }

    function getReferrer(address farmer) public view returns (address) {
        return referrers[farmer];
```

```
    }

    // Set admin status.
    function setAdminStatus(address _admin, bool _status) external onlyOwner {// knownsec 管理管理员数组 owner 可用
        isAdmin[_admin] = _status;
    }

    event EmergencyERC20Drain(address token, address owner, uint256 amount);

    // owner can drain tokens that are sent here by mistake
    function emergencyERC20Drain(ERC20 token, uint amount) external onlyOwner {// knownsec 管理管理员数组 紧急转出 token
        emit EmergencyERC20Drain(address(token), owner, amount);
        token.transfer(owner, amount);
    }
}
```

BTFToken.sol

```
pragma solidity 0.6.12;
// SPDX-License-Identifier: MIT

import "@openzeppelin/contracts/GSN/Context.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

// BTFToken with Governance.
contract BTFToken is ERC20("BTFToken", "BTF"), Ownable {
    /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
    function mint(address _to, uint256 _amount) public onlyOwner {//    knownsec//增发代币
        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _amount);
    }

    // Copied and modified from YAM code:
    // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernanceStorage.sol
    // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernance.sol
    // Which is copied and modified from COMPOUND:
    // https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/Comp.sol

    /// @notice A record of each accounts delegate
    mapping (address => address) internal _delegates;

    /// @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    /// @notice A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

    /// @notice The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints;

    /// @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

    /// @notice The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    /// @notice A record of states for signing / validating signatures
    mapping (address => uint) public nonces;

    /// @notice An event thats emitted when an account changes its delegate
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

    /// @notice An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
     * @param delegator The address to get delegatee for
     */
    function delegates(address delegator)
    external
    view
    returns (address)
    {
```

```
        return _delegates[delegator];
    }

    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
     * @param delegatee The address to delegate votes to
     */
    function delegate(address delegatee) external {
        return _delegate(msg.sender, delegatee);
    }

    /**
     * @notice Delegates votes from signatory to `delegatee`
     * @param delegatee The address to delegate votes to
     * @param nonce The contract state required to match the signature
     * @param expiry The time at which to expire the signature
     * @param v The recovery byte of the signature
     * @param r Half of the ECDSA signature pair
     * @param s Half of the ECDSA signature pair
     */
    function delegateBySig(
        address delegatee,
        uint nonce,
        uint expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
    )
    external
    {
        bytes32 domainSeparator = keccak256(
            abi.encode(
                DOMAIN_TYPEHASH,
                keccak256(bytes(name())),
                getChainId(),
                address(this)
            )
        );

        bytes32 structHash = keccak256(
            abi.encode(
                DELEGATION_TYPEHASH,
                delegatee,
                nonce,
                expiry
            )
        );

        bytes32 digest = keccak256(
            abi.encodePacked(
                "\x19\x01",
                domainSeparator,
                structHash
            )
        );

        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "BTF::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "BTF::delegateBySig: invalid nonce");
        require(now <= expiry, "BTF::delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
    }

    /**
     * @notice Gets the current votes balance for `account`
     * @param account The address to get votes balance
     * @return The number of current votes for `account`
     */
    function getCurrentVotes(address account)
    external
    view
    returns (uint256)
    {
        uint32 nCheckpoints = numCheckpoints[account];
        return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
    }

    /**
     * @notice Determine the prior number of votes for an account as of a block number
     * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
     * @param account The address of the account to check
     * @param blockNumber The block number to get the vote balance at
     * @return The number of votes the account had as of the given block
     */
    function getPriorVotes(address account, uint blockNumber)
    external
    view
```

```
returns (uint256)
{
        require(blockNumber < block.number, "BTF::getPriorVotes: not yet determined");

        uint32 nCheckpoints = numCheckpoints[account];
        if (nCheckpoints == 0) {
                return 0;
        }

        // First check most recent balance
        if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
                return checkpoints[account][nCheckpoints - 1].votes;
        }

        // Next check implicit zero balance
        if (checkpoints[account][0].fromBlock > blockNumber) {
                return 0;
        }

        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
                uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
                Checkpoint memory cp = checkpoints[account][center];
                if (cp.fromBlock == blockNumber) {
                        return cp.votes;
                } else if (cp.fromBlock < blockNumber) {
                        lower = center;
                } else {
                        upper = center - 1;
                }
        }
        return checkpoints[account][lower].votes;
}

function _delegate(address delegator, address delegatee)// knownsec   委托   内部调用
internal
{
        address currentDelegate = _delegates[delegator];
        uint256 delegatorBalance = balanceOf(delegator); // balance of underlying BTFs (not scaled);
        _delegates[delegator] = delegatee;

        emit DelegateChanged(delegator, currentDelegate, delegatee);

        _moveDelegates(currentDelegate, delegatee, delegatorBalance);
}

function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
        if (srcRep != dstRep && amount > 0) {
                if (srcRep != address(0)) {
                        // decrease old representative
                        uint32 srcRepNum = numCheckpoints[srcRep];
                        uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
                        uint256 srcRepNew = srcRepOld.sub(amount);
                        _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
                }

                if (dstRep != address(0)) {
                        // increase new representative
                        uint32 dstRepNum = numCheckpoints[dstRep];
                        uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
                        uint256 dstRepNew = dstRepOld.add(amount);
                        _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
                }
        }
}

function _writeCheckpoint(
        address delegatee,
        uint32 nCheckpoints,
        uint256 oldVotes,
        uint256 newVotes
)
internal
{
        uint32 blockNumber = safe32(block.number, "BTF::_writeCheckpoint: block number exceeds 32 bits");

        if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
                checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
        } else {
                checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
                numCheckpoints[delegatee] = nCheckpoints + 1;
        }

        emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
}
```

```
    function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
        require(n < 2**32, errorMessage);
        return uint32(n);
    }

    function getChainId() internal pure returns (uint) {
        uint256 chainId;
        assembly { chainId := chainid() }
        return chainId;
    }
}
```

MasterChef.sol

```
pragma solidity 0.6.12;

import "@openzeppelin/contracts/GSN/Context.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

import "./BTFToken.sol";
import "./interface/IMigratorChef.sol";
import "./interface/IBTFReferral.sol";

// MasterChef is the master of BTF
contract MasterChef is Ownable {
    uint256 public constant DURATION = 7 days;

    uint256 public TotalSupply = 1000000 * 1e18;
    uint256 public initReward = 100000 * 1e18;
    uint256 public periodFinish = 0;
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    uint256 public rewardRate = 0;

    address public rewardReferral;
    uint256 public constant referralMax = 10000;
    // 1%
    uint256 public referralPercent = 100;

    // Info of each user.
    struct UserInfo {
        uint256 amount;
        uint256 rewardPerTokenPaid;
        uint256 rewards;
    }

    // Info of each pool.
    struct PoolInfo {
        IERC20 lpToken;
        uint256 allocPoint;
        uint256 lastUpdateTime;
        uint256 rewardPerTokenStored;
    }

    // The BTF TOKEN!
    BTFToken public btf;
    // The migrator contract. It has a lot of power. Can only be set through governance (owner).
    IMigratorChef public migrator;
    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;
    // Total allocation poitns. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when BTF mining starts.
    uint256 public startBlock;
    // add the same LP token only once
    mapping(address => bool) lpExists;

    event RewardAdded(uint256 reward);
    event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);
    event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

    constructor(BTFToken _btf, uint256 _startBlock) public {
        btf = _btf;
        startBlock = _startBlock;
    }

    function setRewardReferral(address _rewardReferral) external onlyOwner {
```

```
        rewardReferral = _rewardReferral;
    }

    function setReferralPercent(uint256 _referralPercent) external onlyOwner {
        require(_referralPercent > 0 && _referralPercent < referralMax, "_referralPercent is wrong");
        referralPercent = _referralPercent;
    }

    modifier reduceHalve() {
        require(btf.totalSupply() <= TotalSupply, "Out of limited.");

        if (periodFinish == 0) {
            periodFinish = block.timestamp.add(DURATION);
            rewardRate = initReward.div(DURATION);
            btf.mint(address(this), initReward);
        } else if (block.timestamp >= periodFinish) {
            initReward = initReward.sub(initReward.mul(10).div(100));
            rewardRate = initReward.div(DURATION);
            btf.mint(address(this), initReward);
            periodFinish = block.timestamp.add(DURATION);
            emit RewardAdded(initReward);
        }
        _;
    }

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

    modifier updateReward(uint256 _pid, address _user) {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];

        pool.rewardPerTokenStored = rewardPerToken(_pid);
        pool.lastUpdateTime = lastTimeRewardApplicable();
        if (_user != address(0)) {
            user.rewards = earned(_pid, _user);
            user.rewardPerTokenPaid = pool.rewardPerTokenStored;
        }
        _;
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        return (periodFinish == 0 || block.timestamp < periodFinish) ? block.timestamp : periodFinish;
    }

    function rewardPerToken(uint256 _pid) public view returns (uint256) {
        PoolInfo storage pool = poolInfo[_pid];
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (lpSupply == 0) {
            return pool.rewardPerTokenStored;
        }
        return pool.rewardPerTokenStored.add(
            lastTimeRewardApplicable()
            .sub(pool.lastUpdateTime == 0 ? block.timestamp : pool.lastUpdateTime)
            .mul(rewardRate)
            .mul(pool.allocPoint)
            .div(totalAllocPoint)
            .mul(1e18)
            .div(lpSupply)
        );
    }

    function earned(uint256 _pid, address _user) public view returns (uint256) {
        UserInfo storage user = userInfo[_pid][_user];
        return user.amount
        .mul(rewardPerToken(_pid).sub(user.rewardPerTokenPaid))
        .div(1e18)
        .add(user.rewards);
    }

    // Add a new lp to the pool. Can only be called by the owner.
    function add(uint256 _allocPoint, IERC20 _lpToken) public onlyOwner {
        require(!lpExists[address(_lpToken)], "do not add the same lp token more than once");

        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
            lpToken : _lpToken,
            allocPoint : _allocPoint,
            lastUpdateTime : 0,
            rewardPerTokenStored : 0
            }));

        lpExists[address(_lpToken)] = true;
    }
```

```
    function poolLength() external view returns (uint256) {
        return poolInfo.length;
    }

    // Update the given pool's BTF allocation point. Can only be called by the owner.
    function set(uint256 _pid, uint256 _allocPoint) public onlyOwner {
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // View function to see pending BTFs on frontend.
    function pendingBTF(uint256 _pid, address _user) external view returns (uint256) {
        return earned(_pid, _user);
    }

    function _getReward(uint256 _pid, address _user) private {
        UserInfo storage user = userInfo[_pid][_user];
        uint256 reward = earned(_pid, _user);
        if (reward > 0) {
            user.rewards = 0;

            uint256 btfBal = btf.balanceOf(address(this));
            if (reward > btfBal) {
                reward = btfBal;
            }

            uint256 referrerReward = 0;
            address referrer = address(0);
            if (rewardReferral != address(0)) {
                referrer = IBTFReferral(rewardReferral).getReferrer(_user);
            }
            if (referrer != address(0)) {
                referrerReward = reward.mul(referralPercent).div(referralMax);
                btf.transfer(referrer, referrerReward);
                emit RewardPaid(referrer, referrerReward);
            }

            btf.transfer(_user, reward.sub(referrerReward));
            emit RewardPaid(_user, reward.sub(referrerReward));
        }
    }

    // Deposit LP tokens to MasterChef for BTF allocation.
    function deposit(uint256 _pid, uint256 _amount, address referrer) public updateReward(_pid, msg.sender)
reduceHalve checkStart {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        _getReward(_pid, msg.sender);
        pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
        user.amount = user.amount.add(_amount);
        emit Deposit(msg.sender, _pid, _amount);

        if (rewardReferral != address(0) && referrer != address(0)) {
            IBTFReferral(rewardReferral).setReferrer(msg.sender, referrer);
        }
    }

    // Withdraw LP tokens from MasterChef.
    function withdraw(uint256 _pid, uint256 _amount) public updateReward(_pid, msg.sender) reduceHalve
checkStart {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");
        _getReward(_pid, msg.sender);
        user.amount = user.amount.sub(_amount);
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
        emit Withdraw(msg.sender, _pid, _amount);
    }

    // Withdraw LP tokens & Rewards from MasterChef
    function exit(uint256 _pid) external {
        withdraw(_pid, balanceOf(msg.sender));
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        pool.lpToken.safeTransfer(address(msg.sender), user.amount);
        emit EmergencyWithdraw(msg.sender, _pid, user.amount);
        user.amount = 0;
        user.rewards = 0;
    }

    // Set the migrator contract. Can only be called by the owner.
    function setMigrator(IMigratorChef _migrator) public onlyOwner {// knownsec  设定 migrator 的值  Owner
可用，migrator 拥有非常高的代码执行能力，关联到 255 行  IERC20 newLpToken = migrator.migrate(lpToken);
```

```
        migrator = _migrator;
    }

    // Migrate lp token to another lp contract. Can be called by anyone. We trust that migrator contract is good.
    function migrate(uint256 _pid) public {
        require(address(migrator) != address(0), "migrate: no migrator");
        PoolInfo storage pool = poolInfo[_pid];
        IERC20 lpToken = pool.lpToken;
        uint256 bal = lpToken.balanceOf(address(this));
        lpToken.safeApprove(address(migrator), bal);
        IERC20 newLpToken = migrator.migrate(lpToken);
        require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");
        pool.lpToken = newLpToken;
    }

}
```

Migrations.sol

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

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

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

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

MockERC20.sol

```
pragma solidity 0.6.12;


import "@openzeppelin/contracts/token/ERC20/ERC20.sol";


contract MockERC20 is ERC20 {
    constructor(
        string memory name,
        string memory symbol,
        uint256 supply
    ) public ERC20(name, symbol) {
        _mint(msg.sender, supply);
    }
}
```

Timelock.sol

```
// COPIED FROM https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/GovernorAlpha.sol
// Copyright 2020 Compound Labs, Inc.
// Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
// 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
// 3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
// Ctrl+f for XXX to see all the modifications.

// XXX: pragma solidity ^0.5.16;
pragma solidity 0.6.12;

import "@openzeppelin/contracts/math/SafeMath.sol";
```

```
contract Timelock {
    using SafeMath for uint;

    event NewAdmin(address indexed newAdmin);
    event NewPendingAdmin(address indexed newPendingAdmin);
    event NewDelay(uint indexed newDelay);
    event CancelTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
    event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
    event QueueTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);

    uint public constant GRACE_PERIOD = 14 days;
    uint public constant MINIMUM_DELAY = 12 hours;
    uint public constant MAXIMUM_DELAY = 30 days;

    address public admin;
    address public pendingAdmin;
    uint public delay;
    bool public admin_initialized;

    mapping(bytes32 => bool) public queuedTransactions;


    constructor(address admin_, uint delay_) public {
        require(delay_ >= MINIMUM_DELAY, "Timelock::constructor: Delay must exceed minimum delay.");
        require(delay_ <= MAXIMUM_DELAY, "Timelock::constructor: Delay must not exceed maximum delay.");

        admin = admin_;
        delay = delay_;
        admin_initialized = false;
    }

    // XXX: function() external payable { }
    receive() external payable {}

    function setDelay(uint delay_) public {
        require(msg.sender == address(this), "Timelock::setDelay: Call must come from Timelock.");
        require(delay_ >= MINIMUM_DELAY, "Timelock::setDelay: Delay must exceed minimum delay.");
        require(delay_ <= MAXIMUM_DELAY, "Timelock::setDelay: Delay must not exceed maximum delay.");
        delay = delay_;

        emit NewDelay(delay);
    }

    function acceptAdmin() public {
        require(msg.sender == pendingAdmin, "Timelock::acceptAdmin: Call must come from pendingAdmin.");
        admin = msg.sender;
        pendingAdmin = address(0);

        emit NewAdmin(admin);
    }

    function setPendingAdmin(address pendingAdmin_) public {
        // allows one time setting of admin for deployment purposes
        if (admin_initialized) {
            require(msg.sender == address(this), "Timelock::setPendingAdmin: Call must come from Timelock.");
        } else {
            require(msg.sender == admin, "Timelock::setPendingAdmin: First call must come from admin.");
            admin_initialized = true;
        }
        pendingAdmin = pendingAdmin_;

        emit NewPendingAdmin(pendingAdmin);
    }

    function queueTransaction(address target, uint value, string memory signature, bytes memory data, uint eta) public returns (bytes32) {
        require(msg.sender == admin, "Timelock::queueTransaction: Call must come from admin.");
        require(eta >= getBlockTimestamp().add(delay), "Timelock::queueTransaction: Estimated execution block must satisfy delay.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        queuedTransactions[txHash] = true;

        emit QueueTransaction(txHash, target, value, signature, data, eta);
        return txHash;
    }

    function cancelTransaction(address target, uint value, string memory signature, bytes memory data, uint eta) public {
        require(msg.sender == admin, "Timelock::cancelTransaction: Call must come from admin.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
```

```
        queuedTransactions[txHash] = false;

        emit CancelTransaction(txHash, target, value, signature, data, eta);
    }

    function executeTransaction(address target, uint value, string memory signature, bytes memory data, uint eta)
public payable returns (bytes memory) {
        require(msg.sender == admin, "Timelock::executeTransaction: Call must come from admin.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        require(queuedTransactions[txHash], "Timelock::executeTransaction: Transaction hasn't been
queued.");
        require(getBlockTimestamp() >= eta, "Timelock::executeTransaction: Transaction hasn't surpassed time
lock.");
        require(getBlockTimestamp() <= eta.add(GRACE_PERIOD), "Timelock::executeTransaction:
Transaction is stale.");

        queuedTransactions[txHash] = false;

        bytes memory callData;

        if (bytes(signature).length == 0) {
            callData = data;
        } else {
            callData = abi.encodePacked(bytes4(keccak256(bytes(signature))), data);
        }

        // solium-disable-next-line security/no-call-value
        (bool success, bytes memory returnData) = target.call{value : value}(callData);
        require(success, "Timelock::executeTransaction: Transaction execution reverted.");

        emit ExecuteTransaction(txHash, target, value, signature, data, eta);

        return returnData;
    }

    function getBlockTimestamp() internal view returns (uint) {
        // solium-disable-next-line security/no-block-members
        return block.timestamp;
    }
}
```

Converter.sol

```
pragma solidity ^0.6.2;

interface Converter {
    function convert(address) external returns (uint256);
}
```

Gauge.sol

```
pragma solidity ^0.6.2;

interface Gauge {
    function deposit(uint) external;

    function balanceOf(address) external view returns (uint);

    function withdraw(uint) external;

    function user_checkpoint(address) external;

    function claimable_tokens(address) external returns (uint256);
}

interface VotingEscrow {
    function create_lock(uint256 v, uint256 time) external;

    function increase_amount(uint256 _value) external;

    function increase_unlock_time(uint256 _unlock_time) external;

    function withdraw() external;
}

interface Mintr {
    function mint(address) external;
}
```

IBTFReferral.sol

```
pragma solidity ^0.6.2;
```

```
interface IBTFReferral {
    function setReferrer(address farmer, address referrer) external;
    function getReferrer(address farmer) external view returns (address);
}
```

IController.sol

```
pragma solidity ^0.6.0;

interface IController {
    function vaults(address) external view returns (address);

    function comAddr() external view returns (address);

    function devAddr() external view returns (address);

    function burnAddr() external view returns (address);

    function want(address) external view returns (address); // NOTE: Only StrategyControllerV2 implements this

    function balanceOf(address) external view returns (uint256);

    function withdraw(address, uint256) external;

    function freeWithdraw(address, uint256) external;

    function earn(address, uint256) external;
}
```

IMigratorChef.sol

```
pragma solidity 0.6.12;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface IMigratorChef {
    // Perform LP token migration from legacy UniswapV2 to BTFSwap.
    // Take the current LP token address and return the new LP token address.
    // Migrator should have full access to the caller's LP token.
    // Return the new LP token address.
    //
    // XXX Migrator must have allowance access to UniswapV2 LP tokens.
    // BTFSwap must mint EXACTLY the same amount of BTFSwap LP tokens or
    // else something bad will happen. Traditional UniswapV2 does not
    // do that so be careful!
    function migrate(IERC20 token) external returns (IERC20);
}
```

IStakingRewards.sol

```
pragma solidity ^0.6.2;

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

    function earned(address account) external view returns (uint256);

    function exit() external;

    function getReward() external;

    function getRewardForDuration() external view returns (uint256);

    function lastTimeRewardApplicable() external view returns (uint256);

    function lastUpdateTime() external view returns (uint256);

    function notifyRewardAmount(uint256 reward) external;

    function periodFinish() external view returns (uint256);

    function rewardPerToken() external view returns (uint256);

    function rewardPerTokenStored() external view returns (uint256);

    function rewardRate() external view returns (uint256);

    function rewards(address) external view returns (uint256);

    function rewardsDistribution() external view returns (address);
```

```
    function rewardsDuration() external view returns (uint256);

    function rewardsToken() external view returns (address);

    function stake(uint256 amount) external;

    function stakeWithPermit(
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    function stakingToken() external view returns (address);

    function totalSupply() external view returns (uint256);

    function userRewardPerTokenPaid(address) external view returns (uint256);

    function withdraw(uint256 amount) external;
}
```

IStrategy.sol

```
pragma solidity ^0.6.2;

interface IStrategy {//knownsec//  策略池接口
    function want() external view returns (address);

    function deposit() external;

    function withdraw(address) external;

    function withdraw(uint256) external;

    function withdrawAll() external returns (uint256);

    function balanceOf() external view returns (uint256);

    function freeWithdraw(uint256 _amount) external;

    function harvest() external;

    function getHarvestable() external view returns (uint256);
}
```

IStrategyConverter.sol

```
pragma solidity ^0.6.2;

interface IStrategyConverter {
    function convert(
        address _refundExcess, // address to send the excess amount when adding liquidity
        address _fromWant,
        address _toWant,
        uint256 _wantAmount
    ) external returns (uint256);
}
```

ISwerveFi.sol

```
pragma solidity ^0.6.2;

interface ISwerveFi {
    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;

    function calc_token_amount(
        uint256[4] calldata amounts,
        bool deposit
    ) external view returns (uint);

    function calc_withdraw_one_coin(
        uint256 _token_amount, int128 i) external view returns (uint256);

    function remove_liquidity_one_coin(uint256 _token_amount, int128 i,
        uint256 min_amount) external;

    function balances(int128) external view returns (uint256);
}
```

IUniswapV2Factory.sol

```
pragma solidity ^0.6.2;

interface IUniswapV2Factory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
}
```

IVault.sol

```
pragma solidity ^0.6.2;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface IVault is IERC20 {
    function balance() external view returns (uint256);

    function balanceOfToken() external view returns (uint256);

    function token() external view returns (address);

    function claimInsurance() external; // NOTE: Only yDelegatedVault implements this

    function getRatio() external view returns (uint256);

    function deposit(uint256) external;

    function withdraw(uint256) external;

    function earn() external;
}
```

IYfvRewards.sol

```
pragma solidity ^0.6.2;

interface IYfvRewards {
    function balanceOf(address tokenAddress, address account) external view returns (uint256);

    function lastTimeRewardApplicable() external view returns (uint256);

    function weiTotalSupply() external view returns (uint256);

    function rewardPerToken() external view returns (uint256);

    function weiBalanceOf(address account) external view returns (uint256);

    function earned(address account) external view returns (uint256);
```

```
    function stakingPower(address account) external view returns (uint256);
    function vUSDBalance(address account) external view returns (uint256);
    function vETHBalance(address account) external view returns (uint256);
    function claimVETHReward() external;
    function stake(address tokenAddress, uint256 amount, address referrer) external;
    function withdraw(address tokenAddress, uint256 amount) external;
    function exit() external;
    function getReward() external;
    function nextRewardMultiplier() external view returns (uint16);
    function notifyRewardAmount(uint256 reward) external;
}
```

OneSplitAudit.sol

```
pragma solidity ^0.6.2;

interface OneSplitAudit {
    function getExpectedReturn(
        address fromToken,
        address toToken,
        uint256 amount,
        uint256 parts,
        uint256 featureFlags
    )
    external
    view
    returns (uint256 returnAmount, uint256[] memory distribution);

    function swap(
        address fromToken,
        address toToken,
        uint256 amount,
        uint256 minReturn,
        uint256[] calldata distribution,
        uint256 featureFlags
    ) external payable;
}
```

UniswapRouterV2.sol

```
pragma solidity ^0.6.2;

interface UniswapRouterV2 {
    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
    external
    returns (
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    );

    function addLiquidityETH(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountETHMin,
```

```
        address to,
        uint256 deadline
    )
    external
    payable
    returns (
        uint256 amountToken,
        uint256 amountETH,
        uint256 liquidity
    );

    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountA, uint256 amountB);

    function getAmountsOut(uint256 amountIn, address[] calldata path)
    external
    view
    returns (uint256[] memory amounts);

    function getAmountsIn(uint256 amountOut, address[] calldata path)
    external
    view
    returns (uint256[] memory amounts);

    function swapETHForExactTokens(
        uint256 amountOut,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);

    function swapExactETHForTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);
}
```

USDT.sol

```
pragma solidity ^0.6.0;

interface USDT {
    function approve(address guy, uint256 wad) external;

    function transfer(address _to, uint256 _value) external;
}
```

6. 附录 B：安全风险评级标准

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

7. 附录 C：智能合约安全审计工具简介

6.1 Manticore

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

6.2 Oyente

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

6.3 securify.sh

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

6.4 Echidna

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

6.5 MAIAN

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

约的字节码，并尝试建立一系列交易以找出并确认错误。

6.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7 ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

6.8 Remix-ide

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

6.9 知道创宇区块链安全审计人员专用工具包

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

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509