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：知道创宇云安全服务团队

文档信息

文档名称	文档 MD5	文档版本
moonswap 合约审计报告	e311f92e817d88b2eb477fde9030e000	V1.0

版本说明

修订人	修订内容	修订时间	保密级别
知道创宇云安全服务团队	moonswap 合约审计报告	2020.09.22	项目组公开

版权说明

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

Knownsec

目录

Knownsec

Knownsec

Knownsec

1. 综述

本次报告有效测试时间是从 2020 年 9 月 21 日开始到 2020 年 9 月 22 日结束，在此期间针对 moonswap 智能合约代码的安全性和规范性进行审计并以此作为报告统计依据。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，发现存在存在逻辑设计缺陷风险，用户可将地址设置为空地址，可能导致非预期的情况发生，但是由于利用难度很大，用户一般不会把地址设置成空地址，故综合评定为**通过**。

本次智能合约安全审计结果：通过

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

本次测试的目标信息

Token 名称	**moonswap**
代币地址	**https://github.com/moon-migration/moonswap-eth-contract**
代码类型	**代币代码**
代码语言	**Solidity**
合约地址	□ https://github.com/moon-migration/moonswap-eth-contract/blob/master/contracts/Migrator.sol □ https://github.com/moon-migration/moonswap-core/blob/master/contracts/UniswapV2Factory.sol □ https://github.com/moon-migration/moonswap-eth-

Knownsec

	contract/blob/master/contracts/uniswapv2/CrosschainPair.sol □ https://github.com/moon-migration/moonswap-eth-contract/blob/master/contracts/uniswapv2/CrosschainFactory.sol □ https://github.com/moon-migration/moonswap-core/blob/master/contracts/MigratorFactory.sol □ https://github.com/moon-migration/moonswap-core/blob/master/contracts/MigratorPair.sol □ https://github.com/moon-migration/moonswap-eth-contract/blob/master/contracts/MoonFund.sol □ https://github.com/moon-migration/moonswap-core/blob/master/contracts/ConfluxStar.sol

合约文件及哈希

合约文件	MD5 值
Migrator.sol	035703e3fb403829094f52470ec932ae
UniswapV2Factory.sol	25c9372fcfb98d714c94c1da417f8442
CrosschainPair.sol	8f837c3764c8a1282219505f539f4703
CrosschainFactory.sol	c267bb3b7453fc8d115c900a7f40d042
MigratorFactory.sol	3d49b488c46b7a27147e9f5aa6324b24
MigratorPair.sol	5348a9d72bc3b22287f81a6a2e0b8d11
MoonFund.sol	1bcc007556b6d8436cbb2a7a845aab6e
ConfluxStar.sol	946f9fa44b0b33fb8f27f193f6d77f53

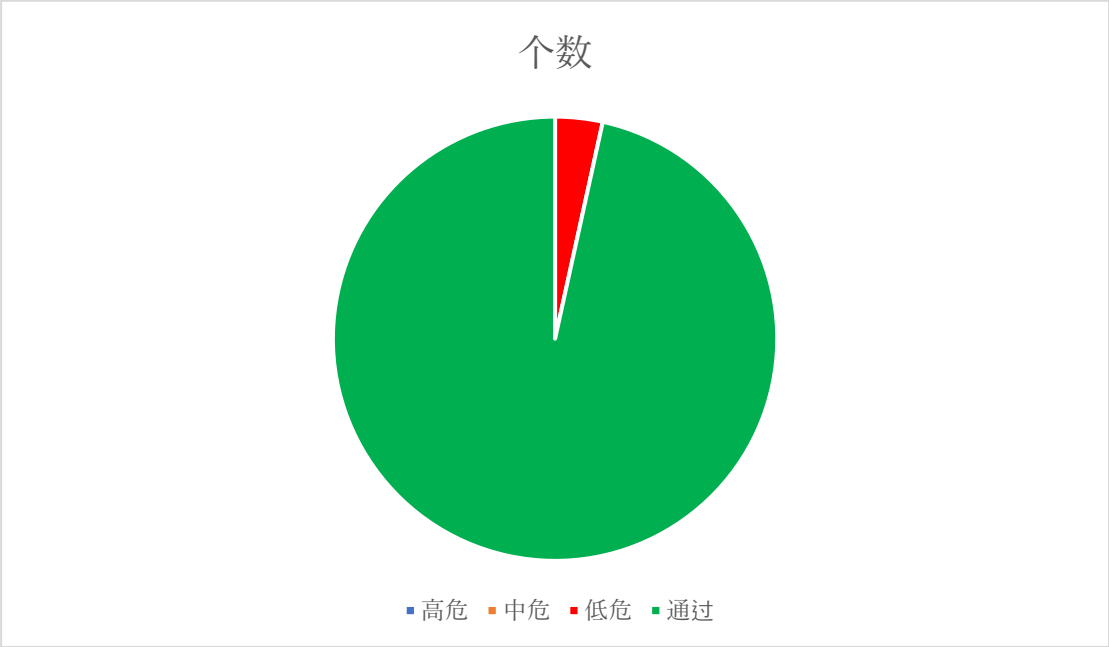
Knownsec

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：

漏洞风险等级个数统计表			
高危	中危	低危	通过
0	0	1	28



2.2. 审计结果汇总说明

智能合约审计结果			
测试序号	测试内容	测试结果	测试描述
1	重入攻击检测	通过	经检测，不存在该安全问题
2	重放攻击检测	通过	经检测，不存在该安全问题
3	重排攻击检测	通过	经检测，不存在该安全问题

4	数值溢出检测	通过	经检测，不存在该安全问题
5	算数精度误差	通过	经检测，不存在该安全问题
6	访问控制缺陷检测	通过	经检测，不存在该安全问题
7	tx.progin 身份验证	通过	经检测，不存在该安全问题
8	call 注入攻击	通过	经检测，不存在该安全问题
9	返回值调用验证	通过	经检测，不存在该安全问题
10	未初始化的存储指针	通过	经检测，不存在该安全问题
11	错误使用随机数检测	通过	经检测，不存在该安全问题
12	交易顺序依赖检测	通过	经检测，不存在该安全问题
13	拒绝服务攻击检测	通过	经检测，不存在该安全问题
14	逻辑设计缺陷检测	（低危） 通过	经检测，存在该安全问题，用户可将地址设置为空地址，可能导致非预期的情况发生，但是由于利用难度很大，用户一般不会把地址设置成空地址
15	假充值漏洞检测	通过	经检测，不存在该安全问题
16	增发代币漏洞检测	通过	经检测，不存在该安全问题
17	冻结账户绕过检测	通过	经检测，不存在该安全问题
18	编译器版本安全	通过	经检测，不存在该安全问题
19	不推荐的编码方式	通过	经检测，不存在该安全问题
20	冗余代码	通过	经检测，不存在该安全问题
21	安全算数库的使用	通过	经检测，不存在该安全问题

Knownsec

22	require/assert 的使用	通过	经检测，不存在该安全问题
23	gas 消耗	通过	经检测，不存在该安全问题
24	fallback 函数的使用	通过	经检测，不存在该安全问题
25	owner 权限控制	通过	经检测，不存在该安全问题
26	低级函数安全	通过	经检测，不存在该安全问题
27	变量覆盖	通过	经检测，不存在该安全问题
28	时间戳依赖攻击	通过	经检测，不存在该安全问题
29	不安全的接口使用	通过	经检测，不存在该安全问题

Knownsec

3. 代码审计结果分析

3.1. 重入攻击检测【通过】

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

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

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.2. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

3.3. 重排攻击检测【通过】

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

检测结果:经检测，智能合约代码中不存在相关洞。

Knownsec

安全建议:无。

3.4. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.5. 算术精度误差【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.6. 访问控制检测【通过】

Knownsec

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。

3.7. tx.origin 身份验证【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.8. call 注入攻击【通过】

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

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

3.9. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状

Knownsec

态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

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

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.10. 未初始化的储存指针【通过】

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

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

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.11. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者

Knownsec

比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.12. 交易顺序依赖【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。

3.13. 拒绝服务攻击【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.14. 逻辑设计缺陷【低危】

检查智能合约代码中与业务设计相关的安全问题。

Knownsec

检测结果：经检测，合约中的 setMigrator 函数和 safeTokenTransfer 等函数中存在逻辑设计缺陷风险，用户可将地址设置为空地址，可能导致非预期的情况发生，代码如下：

ConfluxStar.sol | CrosschainFactory.sol | CrosschainPair.sol | Migrator.sol | MigratorFactory.sol

```
48
49          getPair[token0][token1] = pair;
50          getPair[token1][token0] = pair; // populate mapping in the reverse direction
51          allPairs.push(pair);
52
53          emit PairCreated(token0, token1, pair, allPairs.length);
54      }
55
56      function setMigrator(address _migrator) external override {//knownsec//未验证空地址
57          require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
58          migrator = _migrator;
59      }
```

ConfluxStar.sol

```
203         pool.lpToken.safeTransfer(address(msg.sender), _amount);
204         emit EmergencyWithdraw(msg.sender, _pid, _amount);
205     }
206
207     function moonBalance() external view returns(uint256) {
208       return IERC20(cMoonToken).balanceOf(address(this));
209     }
210
211     function safeTokenTransfer(address _to, uint256 _amount) internal {//knownsec//未验证_to是否为空地址
212         uint256 tokenBal = IERC20(cMoonToken).balanceOf(address(this));
213         require(_amount <= tokenBal, "ConfluxStar: Balance insufficient");
214         IERC20(cMoonToken).transfer(_to, _amount);
```

ConfluxStar.sol

```
217     function _getPoolReward(uint256 _poolLastRewardTime) internal view returns(uint256) {
218         uint256 _timestamp = block.timestamp;
219         return _timestamp.sub(_poolLastRewardTime).mul(tokenPerSecond);
220     }
221
222     // custodian deposit
223     function tokensReceived(address operator, address from, address to, uint amount,
224             bytes calldata userData,
225             bytes calldata operatorData) external {//knownsec//未验证to是否为空地址
226
227             emit TokenTransfer(msg.sender, from, to, amount);
228     }
229
```

CrosschainPair.sol

```
76      // called once by the factory at time of deployment
77      function initialize(address _token0, address _token1) external {
78          require(msg.sender == factory, 'Moonswap: FORBIDDEN'); // sufficient check
79          token0 = _token0;
80          token1 = _token1;
81      }
82
83      //
84      function mint(address to) external lock returns (uint liquidity) {//knownsec//未验证to是否为空地址
85          uint balance0 = IERC20(token0).balanceOf(address(this));
86          uint balance1 = IERC20(token1).balanceOf(address(this));
87          uint amount0 = balance0;
88          uint amount1 = balance1;
89
90          address migrator = ICrosschainFactory(factory).migrator();
91          require(msg.sender == migrator, "Moonswap: FORBIDDEN");
92          liquidity = IMigrator(migrator).desiredLiquidity();
93          require(liquidity > 0, 'Moonswap: INSUFFICIENT_LIQUIDITY_MINTED');
94          _mint(to, liquidity);
95
96          pairMigration.migrateLiquidity = pairMigration.migrateLiquidity.add(liquidity);
```

MigratorPair.sol

```
191
192     function getUserLpAmount(address to) external view returns(uint) {
193       return userLpAmount[to];
194     }
195
196     // custodian deposit
197     function tokensReceived(address operator, address from, address to, uint amount,
198           bytes calldata userData,
199           bytes calldata operatorData) external {//knownsec//未验证to是否为空地址
200
201           address cMoonLpToken = IMigratorFactory(factory).cMoonLpToken(address(this));
202           if(cMoonLpToken == msg.sender) {
203              // Exchange Moonswap Liquidity
204              _exchangeLp(from, amount);
205           }
206
207           emit TokenTransfer(msg.sender, from, to, amount);
208     }
209
```

UniswapV2Factory.sol

```
47          IUniswapV2Pair(pair).initialize(token0, token1);
48          getPair[token0][token1] = pair;
49          getPair[token1][token0] = pair; // populate mapping in the reverse direction
50          allPairs.push(pair);
51
52          emit PairCreated(token0, token1, pair, allPairs.length);
53      }
54
55      function setFeeTo(address _feeTo) external {//knownsec//未验证空地址
56          require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
57          feeTo = _feeTo;
58      }
59
60      function setFeeToSetter(address _feeToSetter) external {//knownsec//未验证空地址
61          require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
62          feeToSetter = _feeToSetter;
63      }
64
65      function setMigrator(address _migrator) external {//knownsec//未验证空地址
66          require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
67          migrator = _migrator;
```

安全建议：以函数 setMigrator 为例限制地址不为空地址，添加如下代码：

require(_migrator != address(0));

3.15. 假充值漏洞【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.16. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.17. 冻结账户绕过【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.18. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

knownsec

检测结果：经检测，智能合约代码中制定了编译器版本 0.5.8 以上，不存在该安全问题。

安全建议：无。

3.19. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.20. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.21. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

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

安全建议：无。

3.22. require/assert 的使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

Knownsec

3.23. gas 消耗【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.24. fallback 函数使用【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.25. owner 权限控制【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.26. 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞 call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.27. 变量覆盖【通过】

Knownsec

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题

安全建议：无。

3.28. 时间戳依赖攻击【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.29. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

Knownsec

4. 附录 A：合约代码

本次测试代码来源：

4.1. ConfluxStar.sol

```
pragma solidity =0.5.16;//knownsec//指定编译器版本，符合推荐做法

import  "@openzeppelin/contracts/math/SafeMath.sol";//knownsec//使用了安全的数值计算函数

import "@openzeppelin/contracts/introspection/IERC1820Registry.sol";
import "@openzeppelin/contracts/token/ERC777/IERC777Recipient.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/ownership/Ownable.sol";
import "@openzeppelin/contracts/token/ERC777/IERC777.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "./Pauseable.sol";
import './SponsorWhitelistControl.sol';
import './libraries/Math.sol';
/**
 * Interstellar migration
 * Welcome to Conflux Star ##Farm
 *
 */
contract ConfluxStar is Ownable, Pauseable, IERC777Recipient {//knownsec//ConfluxStar 继承自 Ownable, Pauseable, IERC777Recipient
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    SponsorWhitelistControl constant public SPONSOR = SponsorWhitelistControl(address(0x0888000000000000000000000000000000000001));
    IERC1820Registry private _erc1820 = IERC1820Registry(0x866aCA87FF33a0ae05D2164B3D999A804F583222);
    // keccak256("ERC777TokensRecipient")
    bytes32 constant private TOKENS_RECIPIENT_INTERFACE_HASH =

0xb281fc8c12954d22544db45de3159a39272895b169a852b314f9cc762e44c53b;
```

Knownsec

```
struct UserInfo {
    uint256 amount;
    uint256 rewardDebt;
}
struct PoolInfo {
    IERC20 lpToken;                 // Address of LP token contract.
    uint256 allocPoint;             // How many allocation points assigned to this pool. Moons to distribute per block.
    uint256 lastRewardTime;    //
    uint256 accTokenPerShare; //
}
// Moon crosschain atom mapping
address public cMoonToken;
uint256 public tokenPerSecond;
uint256 public startFarmTime;
// 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 Token mining starts.
uint256 public startBlock;
mapping(address => uint256) public poolIndexs;
mapping (address => bool) private _accountCheck;
address[] private _accountList;
event TokenTransfer(address indexed tokenAddress, address indexed from, address to, uint value);
event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

constructor(//knownsec//构造函数
        address _cMoon,
        uint256 _tokenPerSecond,
        uint256 _startTime
    ) public {
        cMoonToken = _cMoon;
        tokenPerSecond = _tokenPerSecond; // Calculate the production rate according to the mining situation
        startFarmTime = _startTime;
        _erc1820.setInterfaceImplementer(address(this), TOKENS_RECIPIENT_INTERFACE_HASH, address(this));
```

Knownsec

```
        // register all users as sponsees
        address[] memory users = new address[](1);
        users[0] = address(0);
        SPONSOR.add_privilege(users);
    }
    function poolLength() external view returns (uint256) {
        return poolInfo.length;
    }
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        require(poolIndexs[address(_lpToken)] < 1, "LpToken exists");
        uint256 lastRewardTime = block.timestamp > startFarmTime ? block.timestamp : startFarmTime;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardTime: lastRewardTime,
            accTokenPerShare: 0
        }));
        poolIndexs[address(_lpToken)] = poolInfo.length;
    }
    function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }
    function setTokenPerSecond(uint256 _tokenPerSecond) public onlyOwner {
        massUpdatePools();
        tokenPerSecond = _tokenPerSecond;
    }
    function pendingToken(uint256 _pid, address _user) external view returns (uint256) {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accTokenPerShare = pool.accTokenPerShare;
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
```

Knownsec

```
            if (block.timestamp > pool.lastRewardTime && lpSupply != 0) {
                uint256 tokenReward = _getPoolReward(pool.lastRewardTime);
                accTokenPerShare =
accTokenPerShare.add(tokenReward.mul(1e12).div(lpSupply));
            }

            return
user.amount.mul(accTokenPerShare).div(1e12).sub(user.rewardDebt);
        }
        function massUpdatePools() public whenPaused {
            uint256 length = poolInfo.length;
            for (uint256 pid = 0; pid < length; ++pid) {
                updatePool(pid);
            }
        }
        function updatePool(uint256 _pid) public whenPaused {
            PoolInfo storage pool = poolInfo[_pid];
            if (block.timestamp <= pool.lastRewardTime) {
                return;
            }
            uint256 lpSupply = pool.lpToken.balanceOf(address(this));
            if (lpSupply == 0) {
                pool.lastRewardTime = block.timestamp;
                return;
            }
            uint256 tokenReward = _getPoolReward(pool.lastRewardTime);
            pool.accTokenPerShare =
pool.accTokenPerShare.add(tokenReward.mul(1e12).div(lpSupply));
            pool.lastRewardTime = block.timestamp;
        }
        //
        function deposit(uint256 _pid, uint256 _amount, address to) public
whenPaused {
            if(to == address(0)){
                to = address(msg.sender);
            }
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][to];
            updatePool(_pid);
            if (user.amount > 0) {
                uint256 pending =
user.amount.mul(pool.accTokenPerShare).div(1e12).sub(user.rewardDebt);
                safeTokenTransfer(to, pending);
            }
```

Knownsec

```
            pool.lpToken.safeTransferFrom(to, address(this), _amount);
            user.amount = user.amount.add(_amount);
            user.rewardDebt = user.amount.mul(pool.accTokenPerShare).div(1e12);
            // data migration
            if (!_accountCheck[to]) {
                _accountCheck[to] = true;
                _accountList.push(to);
            }
            emit Deposit(to, _pid, _amount);
        }
        function withdraw(uint256 _pid, uint256 _amount) public whenPaused {
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];
            require(_amount > 0, "user amount is zero");
            require(user.amount >= _amount, "withdraw: not good");
            updatePool(_pid);
            uint256 pending =
user.amount.mul(pool.accTokenPerShare).div(1e12).sub(user.rewardDebt);
            safeTokenTransfer(msg.sender, pending);
            user.amount = user.amount.sub(_amount);
            user.rewardDebt = user.amount.mul(pool.accTokenPerShare).div(1e12);
            pool.lpToken.safeTransfer(address(msg.sender), _amount);
            emit Withdraw(msg.sender, _pid, _amount);
        }
        function emergencyWithdraw(uint256 _pid) public whenPaused {
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];
            uint256 _amount = user.amount;
            require(_amount > 0, "user amount is zero");
            user.amount = 0;
            user.rewardDebt = 0;
            pool.lpToken.safeTransfer(address(msg.sender), _amount);
            emit EmergencyWithdraw(msg.sender, _pid, _amount);
        }
        function moonBalance() external view returns(uint256) {
          return IERC20(cMoonToken).balanceOf(address(this));
        }
        function safeTokenTransfer(address _to, uint256 _amount) internal
{//knownsec//未验证_to 是否为空地址
            uint256 tokenBal = IERC20(cMoonToken).balanceOf(address(this));
            require(_amount <= tokenBal, "ConfluxStar: Balance insufficient");
            IERC20(cMoonToken).transfer(_to, _amount);
        }
```

Knownsec

```
        function _getPoolReward(uint256 _poolLastRewardTime) internal view returns(uint256) {
            uint256 _timestamp = block.timestamp;
            return _timestamp.sub(_poolLastRewardTime).mul(tokenPerSecond);
        }
        // custodian deposit
        function tokensReceived(address operator, address from, address to, uint amount,
            bytes calldata userData,
            bytes calldata operatorData) external {//knownsec//未验证 to 是否为空地址
            emit TokenTransfer(msg.sender, from, to, amount);
        }
        //---------------- Data Migration ----------------------
        function accountTotal() public view returns (uint256) {
            return _accountList.length;
        }
        function accountList(uint256 begin) public view returns (address[100] memory) {
            require(begin >= 0 && begin < _accountList.length, "MoonSwap: accountList out of range");
            address[100] memory res;
            uint256 range = Math.min(_accountList.length, begin.add(100));
            for (uint256 i = begin; i < range; i++) {
                res[i-begin] = _accountList[i];
            }
            return res;
        }
    }
```

4.2. Migrator.sol

```
pragma solidity 0.6.12;//knownsec//指定编译器版本，符合推荐做法
import "./uniswapv2/interfaces/IUniswapV2Pair.sol";
import "./uniswapv2/interfaces/IUniswapV2Factory.sol";
import "./uniswapv2/interfaces/ICrosschainPair.sol";
import "./uniswapv2/interfaces/ICrosschainFactory.sol";
contract Migrator {
    address public master;
    ICrosschainFactory public factory;
    uint256 public notBeforeBlock;
```

Knownsec

```
    uint256 public desiredLiquidity = uint256(-1);
    mapping(address => bool) public originalFactories;

    constructor(//knownsec//构造函数

        address _master,
        address[] memory _oldFactories,
        ICrosschainFactory _factory,
        uint256 _notBeforeBlock
    ) public {
        master = _master;
        factory = _factory;
        notBeforeBlock = _notBeforeBlock;
        uint range = _oldFactories.length;
        require(range > 0, "Migrate: oldFactory Empty");
        for (uint i = 0; i < range; i++) {
            originalFactories[_oldFactories[i]] = true;
        }
    }
    function migrate(IUniswapV2Pair orig) public returns (ICrosschainPair) {
        require(msg.sender == master, "not from master access");
        require(block.number >= notBeforeBlock, "too early to migrate");
        require(originalFactories[orig.factory()], "not from old factory");
        address token0 = orig.token0();
        address token1 = orig.token1();
        ICrosschainPair pair = ICrosschainPair(factory.getPair(token0, token1));
        if (pair == ICrosschainPair(address(0))) {
            pair = ICrosschainPair(factory.createPair(token0, token1));
        }
        uint256 lp = orig.balanceOf(msg.sender);
        if (lp == 0) return pair;
        desiredLiquidity = lp;
        orig.transferFrom(msg.sender, address(orig), lp);
        orig.burn(address(pair));
        pair.mint(msg.sender);
        desiredLiquidity = uint256(-1);
        return pair;
    }
}
```

4.3. UniswapV2Factory.sol

```
pragma solidity >=0.5.16;//knownsec//指定编译器版本，符合推荐做法

import './interfaces/IUniswapV2Factory.sol';
```

knownsec

```
import './UniswapV2Pair.sol';
import './SponsorWhitelistControl.sol';
import "./Pauseable.sol";
contract UniswapV2Factory is IUniswapV2Factory, Pauseable
{//knownsec//UniswapV2Factory 继承自 IUniswapV2Factory, Pauseable
    address public feeTo;
    address public feeToSetter;
    address public migrator; // migratorFactory
    mapping(address => mapping(address => address)) public getPair;
    address[] public allPairs;
    SponsorWhitelistControl constant public SPONSOR =
SponsorWhitelistControl(address(0x0888000000000000000000000000000000000001));
    mapping(address => bool) public tokenBlacklist; // create pair blacklist
    event PairCreated(address indexed token0, address indexed token1, address
pair, uint);

    constructor(address _feeToSetter)//knownsec//构造函数

      Pauseable()
      public {
        feeToSetter = _feeToSetter;
        // register all users as sponsees
        address[] memory users = new address[](1);
        users[0] = address(0);
        SPONSOR.add_privilege(users);
    }
    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }
    function createPair(address tokenA, address tokenB) external whenPaused
returns (address pair) {
        require(tokenA != tokenB, 'UniswapV2: IDENTICAL_ADDRESSES');
        require(!tokenBlacklist[tokenA] && !tokenBlacklist[tokenB],
"createPair: not allow token");
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) :
(tokenB, tokenA);
        require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'UniswapV2:
PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(UniswapV2Pair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
```

Knownsec

```
        }
        IUniswapV2Pair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setFeeTo(address _feeTo) external {//knownsec//未验证空地址
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {//knownsec//未验证空地址
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }

    function setMigrator(address _migrator) external {//knownsec//未验证空地址
        require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
        migrator = _migrator;
    }
    function setTokenBlacklist(address tokenAddr, bool _status) external {
      require(msg.sender == feeToSetter, 'Factory: FORBIDDEN');
      tokenBlacklist[tokenAddr] = _status;
    }
}
```

4.4. CrosschainPair.sol

```
pragma solidity =0.6.12;//knownsec//指定编译器版本，符合推荐做法

import './UniswapV2ERC20.sol';

import './libraries/Math.sol';//knownsec//使用了安全的数值计算函数

import './libraries/UQ112x112.sol';
import './interfaces/IERC20.sol';
import './interfaces/ICrosschainFactory.sol';
import './interfaces/IUniswapV2Callee.sol';
```

Knownsec

```
import './interfaces/IWETH.sol';
interface IMigrator {
    // Return the desired amount of liquidity token that the migrator wants.
    function desiredLiquidity() external view returns (uint256);
}

contract CrosschainPair is UniswapV2ERC20 {//CrosschainPair 继承自 UniswapV2ERC20
    using SafeMath  for uint;
    using UQ112x112 for uint224;
    address public WETH;
    uint public constant MINIMUM_LIQUIDITY = 10**3;
    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));
    address public factory;
    address public token0;
    address public token1;
    uint112 private reserve0;           // uses single storage slot, accessible via getReserves
    uint112 private reserve1;           // uses single storage slot, accessible via getReserves
    uint32  private blockTimestampLast; // uses single storage slot, accessible via getReserves
    struct PairMigration {
      uint migrateLiquidity;       // migrator total liquidity
      uint amount0;
      uint amount1;
    }
    PairMigration public pairMigration;
    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'UniswapV2: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }
    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'Moonswap: TRANSFER_FAILED');
    }
    function _safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
```

Knownsec

```
        require(success, 'Moonswap: ETH_TRANSFER_FAILED');
    }
    receive() external payable {
        assert(msg.sender == WETH); // only accept ETH via fallback from the
WETH contract
    }
    function getReserves() public view returns (uint112 _reserve0, uint112
_reserve1, uint32 _blockTimestampLast) {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }
    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address
indexed to);

    constructor() public {//knownsec//构造器

        factory = msg.sender;
        WETH = ICrosschainFactory(factory).WETH();
    }
    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'Moonswap: FORBIDDEN'); //
sufficient check
        token0 = _token0;
        token1 = _token1;
    }
    //
    function mint(address to) external lock returns (uint liquidity) {//knownsec//
未验证 to 是否为空地址
        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0;
        uint amount1 = balance1;
        address migrator = ICrosschainFactory(factory).migrator();
        require(msg.sender == migrator, "Moonswap: FORBIDDEN");
        liquidity = IMigrator(migrator).desiredLiquidity();
        require(liquidity > 0, 'Moonswap:
INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);
        pairMigration.migrateLiquidity =
pairMigration.migrateLiquidity.add(liquidity);
        pairMigration.amount0 = pairMigration.amount0.add(amount0);
```

knownsec

```
                pairMigration.amount1 = pairMigration.amount1.add(amount1);
                // move asset for crosschain safe address audit the process
                address _receiveAddress = ICrosschainFactory(factory).getCfxReceiveAddr(address(this));
                require(_receiveAddress != address(0), 'Moonswap: receive is ZERO_ADDRESS');//knownsec//验证空地址
                if(token0 == WETH){
                    IWETH(WETH).withdraw(amount0);
                    _safeTransferETH(_receiveAddress, amount0);
                }else{
                    _safeTransfer(token0, _receiveAddress, amount0);
                }
                if(token1 == WETH){
                    IWETH(WETH).withdraw(amount1);
                    _safeTransferETH(_receiveAddress, amount1);
                }else{
                    _safeTransfer(token1, _receiveAddress, amount1);
                }
                emit Mint(msg.sender, amount0, amount1);
            }
        }
```

4.5. CrosschainFactory.sol

```
    pragma solidity =0.6.12;//knownsec//指定编译器版本，符合推荐做法

    import './interfaces/ICrosschainFactory.sol';
    import './interfaces/ICrosschainPair.sol';
    import './CrosschainPair.sol';
    contract CrosschainFactory is ICrosschainFactory {//knownsec//CrosschainFactory 继承自 ICrosschainFactory
        address public override migrator;
        address public override feeToSetter;
        address public override WETH;

        mapping(address => mapping(address => address)) public override getPair;
        address[] public override allPairs;
        mapping(address => address) public override getCfxReceiveAddr;

        event PairCreated(address indexed token0, address indexed token1, address pair, uint);
```

Knownsec

```
    constructor() public {//knownsec//构造函数
        feeToSetter = msg.sender;

        uint chainId;
        assembly {
            chainId := chainid()
        }

        WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
        if( chainId == 4 ){ // rinkeby
            WETH = 0xc778417E063141139Fce010982780140Aa0cD5Ab;
        }
    }

    function allPairsLength() external override view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external override
returns (address pair) {
        require(tokenA != tokenB, 'MoonSwap: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) :
(tokenB, tokenA);
        require(token0 != address(0), 'MoonSwap: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'MoonSwap:
PAIR_EXISTS'); // single check is sufficient//knownsec//验证空地址
        bytes memory bytecode = type(CrosschainPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }

        ICrosschainPair(pair).initialize(token0, token1);

        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse
direction
        allPairs.push(pair);

        emit PairCreated(token0, token1, pair, allPairs.length);
    }
```

Knownsec

```
    function setMigrator(address _migrator) external override {//knownsec//未
```

验证空地址

```
        require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
        migrator = _migrator;
    }

    // add safe conflux fund contract associated address
    function addCfxReceiveAddr(address token0, address token1, address
_receiveAddr) external {
        require(msg.sender == feeToSetter, 'Moonswap: FORBIDDEN');
        require(_receiveAddr != address(0), "MoonSwap: Receive Address is
zero");
        address pair = getPair[token0][token1];
        require(pair != address(0), "MoonSwap: Pair no exists");

        getCfxReceiveAddr[pair] = _receiveAddr;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
    function setWETH(address _weth) external {
        require(msg.sender == feeToSetter, 'MoonSwap: FORBIDDEN');
        require(_weth != address(0), "MoonSwap: weth is zero");
        WETH = _weth;
    }
}
```

4.6. MigratorFactory.sol

```
pragma solidity ^0.5.16;//knownsec//指定编译器版本，符合推荐做法


import './SponsorWhitelistControl.sol';
import "@openzeppelin/contracts/utils/Address.sol";
import './MigratorPair.sol';

import "@openzeppelin/contracts/math/SafeMath.sol";//knownsec//使用了安全
```

的数值计算函数

```
import "./Pauseable.sol";
```

Knownsec

```
// migrator get crosschain address, then listen transfer convert erc777 cToken
// register tokenReceive get the asset


contract MigratorFactory is Pauseable {//knownsec//MigratorFactory 继承自Pauseable
        using Address for address;
        using SafeMath    for uint;

        address public operatorAddr;

        mapping(address => mapping(address => address)) public getPair;
        address[] public allPairs;
        SponsorWhitelistControl constant public SPONSOR = SponsorWhitelistControl(address(0x0888000000000000000000000000000000000001));
        mapping(address => address) public cMoonLpToken;
        address public swapFactory;
        mapping(address => uint) public desiredLiquidity;
        mapping(address => uint) public getInflationPair; // Lp Decimals diff

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


        constructor(address _operatorAddr)//knownsec//构造函数

          Pauseable()
          public {
            operatorAddr = _operatorAddr;
            // register all users as sponsees
            address[] memory users = new address[](1);
            users[0] = address(0);
            SPONSOR.add_privilege(users);
        }

        function allPairsLength() external view returns (uint) {
            return allPairs.length;
        }

        function createPair(address tokenA, address tokenB) external whenPaused returns (address pair) {
            require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
            require(tokenA != tokenB, 'MoonSwap: IDENTICAL_ADDRESSES');
```

Knownsec

```
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) :
(tokenB, tokenA);
        require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'MoonSwap:
PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(MigratorPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        MigratorPair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse
direction
        allPairs.push(pair);

        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setMoonLpToken(address token0, address token1, address
_moonLpToken) external {
        require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
        address pair = getPair[token0][token1];
        require(pair != address(0), "MoonSwap: pair no create");
        cMoonLpToken[pair] = _moonLpToken;
    }

    function setSwapFactory(address _swapFactory) external {
        require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
        swapFactory = _swapFactory;
    }

    function setOperatorAddr(address _operatorAddr) external {
        require(msg.sender == operatorAddr, 'MoonSwap: FORBIDDEN');
        operatorAddr = _operatorAddr;
    }

    function setDesiredLiquidity(address token0, address token1, uint
_desiredLiquidity, uint _multiplier) external {
        require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
        address pair = getPair[token0][token1];
        require(pair != address(0), "MoonSwap: pair no create");
        require(_multiplier > 0, "MoonSwap: multiplier must setting");
        desiredLiquidity[pair] = _desiredLiquidity.mul(_multiplier);
```

Knownsec

```
            getInflationPair[pair] = _multiplier;
        }

        function getDesiredLiquidity(address token0, address token1) public view returns(uint){
            address pair = getPair[token0][token1];
            require(pair != address(0), "MoonSwap: pair no create");

            return desiredLiquidity[pair];
        }
    }
```

4.7. MigratorPair.sol

```
    pragma solidity ^0.5.16;//knownsec//指定编译器版本，符合推荐做法

    import './SponsorWhitelistControl.sol';
    import './libraries/Math.sol';
    import "@openzeppelin/contracts/math/SafeMath.sol";
    import "@openzeppelin/contracts/introspection/IERC1820Registry.sol";
    import "@openzeppelin/contracts/token/ERC777/IERC777Recipient.sol";
    import "@openzeppelin/contracts/utils/Address.sol";
    import "@openzeppelin/contracts/ownership/Ownable.sol";
    import "@openzeppelin/contracts/token/ERC777/IERC777.sol";
    import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
    import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";//knownsec//使用了安全的数值计算函数

    interface IMigratorFactory {
        function operatorAddr() external view returns (address);
        function swapFactory() external view returns (address);
        function cMoonLpToken(address pair) external view returns (address);
        function getInflationPair(address pair) external view returns (uint);
        function getPause() external view returns (bool);
        function confluxStar() external view returns (address);
    }

    interface IUniswapV2Factory{
        function getPair(address tokenA, address tokenB) external view returns (address pair);
    }
```

Knownsec

```
interface IUniswapV2Pair {
    function mint(address to) external returns (uint);
    function transfer(address to, uint value) external returns (bool);
    function approve(address spender, uint value) external returns (bool);
}

interface IConfluxStar {
    function deposit(uint256 _pid, uint256 _amount, address to) external;
    function poolIndexs(address lpToken) external returns(uint256);
}

contract MigratorPair is IERC777Recipient {//knownsec//MigratorPair 继承自
IERC777Recipient
    using SafeMath    for uint;
    using Address for address;
    using SafeERC20 for IERC20;

    address public factory;
    address public token0;
    address public token1;

    mapping (address => bool) private _accountCheck;
    address[] private _accountList;

    // operator upload user shareAmount for airdrop FC
    uint totalShareAmount;
    mapping(address => uint) userShareAmount;
    uint totalLpAmount;
    mapping(address => uint) userLpAmount;

    IERC1820Registry private _erc1820 =
IERC1820Registry(0x866aCA87FF33a0ae05D2164B3D999A804F583222);
    // keccak256("ERC777TokensRecipient")
    bytes32 constant private TOKENS_RECIPIENT_INTERFACE_HASH =

0xb281fc8c12954d22544db45de3159a39272895b169a852b314f9cc762e44c53b;

    SponsorWhitelistControl constant public SPONSOR =
SponsorWhitelistControl(address(0x0888000000000000000000000000000000000000
1));
    event TokenTransfer(address indexed tokenAddress, address indexed from,
address to, uint value);
```

Knownsec

```
    event ExchangeLpToken(address indexed swapPair, address indexed from, uint
value);
    event AddLiquidityEvent(address indexed swapPair, address indexed from, uint
balance0, uint balance1);


    constructor()//knownsec//构造函数
        public
    {
        factory = msg.sender;
        _erc1820.setInterfaceImplementer(address(this),
TOKENS_RECIPIENT_INTERFACE_HASH, address(this));

        // register all users as sponsees
        address[] memory users = new address[](1);
        users[0] = address(0);
        SPONSOR.add_privilege(users);
    }

    modifier whenPaused() {
        require(!IMigratorFactory(factory).getPause(), "Pauseable: MoonSwap
paused");
        _;
    }

    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'MoonSwap: FORBIDDEN'); // sufficient
check
        token0 = _token0;
        token1 = _token1;
    }

    // user cMoonLpToken Exchange
    function exchangeLp() external {
        address cMoonLpToken =
IMigratorFactory(factory).cMoonLpToken(address(this));
        require(cMoonLpToken != address(0), "Moonswap: cMoonLpToken is
ZERO_ADDRESS");
        uint amount = IERC20(cMoonLpToken).balanceOf(msg.sender);
        require(amount > 0, "MoonSwap: no balance");
        IERC20(cMoonLpToken).safeTransferFrom(msg.sender, address(this),
amount);
```

Knownsec

```
        _exchangeLp(msg.sender, amount);
    }

    function _exchangeLp(address from, uint amount) internal {
        address swapFactory = IMigratorFactory(factory).swapFactory();
        address confluxStar = IMigratorFactory(factory).confluxStar();
        address swapPair = IUniswapV2Factory(swapFactory).getPair(token0,
token1);
        require(swapPair != address(0), "MoonSwap: no swap pair");//knownsec//
```

验证空地址

```
        uint _multiplier =
IMigratorFactory(factory).getInflationPair(address(this));
        require(_multiplier > 0, "Moonswap: multiplier is zero");
        amount = amount.mul(_multiplier); // diff decimals Inflation Amount


        if(confluxStar != address(0)){//knownsec//验证空地址
          IUniswapV2Pair(swapPair).approve(confluxStar, amount);
          uint256 _pIndex = IConfluxStar(confluxStar).poolIndexs(swapPair);
          if(_pIndex > 0){
              IConfluxStar(confluxStar).deposit(_pIndex.sub(1), amount,
from);//knownsec//在 deposit 函数里面验证了空地址
          }else{
              IUniswapV2Pair(swapPair).transfer(from, amount);
          }

        }else{
          IUniswapV2Pair(swapPair).transfer(from, amount);
        }
        userLpAmount[from] = userLpAmount[from].add(amount);

        // data migration
        if (!_accountCheck[from]) {
            _accountCheck[from] = true;
            _accountList.push(from);
        }

        emit ExchangeLpToken(swapPair, from, amount);
    }

    // Add liquidity by Operator
    function addLiquidity() external {
```

Knownsec

```
        address operatorAddr = IMigratorFactory(factory).operatorAddr();
        require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
        address swapFactory = IMigratorFactory(factory).swapFactory();
        // when finish addLiquidity close the method
        require(swapFactory != address(0), "MoonSwap: no swap factory");//knownsec//验证空地址

        address swapPair = IUniswapV2Factory(swapFactory).getPair(token0, token1);
        require(swapPair != address(0), "MoonSwap: no swap pair");//knownsec//验证空地址

        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        require(balance0 > 0 && balance1 > 0, "MoonSwap: balance is zero!");

        IERC20(token0).transfer(swapPair, balance0);
        IERC20(token1).transfer(swapPair, balance1);

        uint liquidity = IUniswapV2Pair(swapPair).mint(address(this));

        totalLpAmount = liquidity;

        emit AddLiquidityEvent(swapPair, msg.sender, balance0, balance1);
    }

    function setTotalShareAmount(uint _shareAmount) external {
        require(_shareAmount > 0, "MoonSwap: shareAmount is zero");
        address operatorAddr = IMigratorFactory(factory).operatorAddr();
        require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
        totalShareAmount = _shareAmount;
    }

    function uploadUserShares(address[] calldata _users, uint[] calldata _shareAmounts) external {
        address operatorAddr = IMigratorFactory(factory).operatorAddr();
        require(msg.sender == operatorAddr, "MoonSwap: FORBIDDEN");
        uint range = _users.length;
        require(range == _shareAmounts.length, "length is no match");

        for (uint i = 0; i < range; i++) {
            address _user = _users[i];
```

knownsec

```
            uint _shareAmount = _shareAmounts[i];
            userShareAmount[_user] = _shareAmount;
        }
    }

    // upload user ethereum stake data Time dimension
    function getTotalShareAmount() external view returns(uint){
        return totalShareAmount;
    }

    // interface
    function getShareAmount(address to) external view returns (uint) {
        return userShareAmount[to];
    }

    function getTotalLpAmount() external view returns(uint){
        return totalLpAmount;
    }

    function getUserLpAmount(address to) external view returns(uint) {
        return userLpAmount[to];
    }

    // custodian deposit
    function tokensReceived(address operator, address from, address to, uint
amount,
                bytes calldata userData,
                bytes calldata operatorData) external {//knownsec//未验证 to 是否为
空地址

                address cMoonLpToken =
IMigratorFactory(factory).cMoonLpToken(address(this));
                if(cMoonLpToken == msg.sender) {
                    // Exchange Moonswap Liquidity
                    _exchangeLp(from, amount);
                }

                emit TokenTransfer(msg.sender, from, to, amount);
    }

    //---------------- Data Migration ----------------------
        function accountTotal() public view returns (uint256) {
```

Knownsec

```
            return _accountList.length;
        }

        function accountList(uint256 begin) public view returns (address[100] memory) {
            require(begin >= 0 && begin < _accountList.length, "MigratorPair: accountList out of range");
            address[100] memory res;
            uint256 range = Math.min(_accountList.length, begin.add(100));
            for (uint256 i = begin; i < range; i++) {
                res[i-begin] = _accountList[i];
            }
            return res;
        }
    }
```

4.8. MoonFund.sol

```
pragma solidity 0.6.12;//knownsec//指定编译器版本，符合推荐做法

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";

import "@openzeppelin/contracts/math/SafeMath.sol";//knownsec//使用了安全的数值计算函数

import "@openzeppelin/contracts/access/Ownable.sol";
interface IMasterStar {
    function deposit(uint256 _pid, uint256 _amount) external;
}
interface IMoonFansToken {
    function mint(address _to, uint256 _amount) external;
}

/**
 * Moon transfer conflux blockchain by MoonFund
 * Moon => cMoon process
 */

contract MoonFund is Ownable {//knownsec//MoonFund 继承自 Ownable

    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    // receive user deposit Moon
```

Knownsec

```
        mapping(address => uint256) reserves;
        address public fansToken;
        address public masterStar;
        address public moonToken;
        address public confluxStarAddr; // Conflux star Address associated etherum address
        uint256 stakePid; //
        event StakeEvent(address indexed user, uint256 indexed pid, uint256 amount);
        event MintcTokenEvent(address indexed user, uint256 indexed pid, address _to, uint256 amount);
        event SaveEvent(address indexed user, address _to, uint256 amount);
        event RetrieveEvent(address indexed user, uint256 amount);

        constructor(//knownsec//构造函数
            address _fansToken,
            address _mastarStar,
            address _moonToken)
            public
        {
            fansToken = _fansToken;
            masterStar = _mastarStar;
            moonToken = _moonToken;
        }
        // before stake, create pool at MasterStar
        // when stake, this contract get FansToken mint operator
        function stake(uint256 _pid, uint256 _amount) external onlyOwner {
            IMoonFansToken(fansToken).mint(address(this), _amount);
            IERC20(fansToken).safeApprove(masterStar, _amount);
            IMasterStar(masterStar).deposit(_pid, _amount);
            stakePid = _pid;
            emit StakeEvent(msg.sender, _pid, _amount);
        }
        function setConfluxStarAddr(address _addr) external onlyOwner {
            require(_addr != address(0), "MoonFund: addr is zero address");//knownsec//验证空地址
            confluxStarAddr = _addr;
        }
        // the step deposit 0 masterStar,
        function mintcToken() external {
            uint256 _pid = stakePid;
            address _to = confluxStarAddr;
```

Knownsec

```
            require(_to != address(0), "MoonFund: addr is zero address");//knownsec//验证空地址
            uint256 beforeBalance = IERC20(moonToken).balanceOf(address(this));
            IMasterStar(masterStar).deposit(_pid, 0);
            uint256 _diffAmount = IERC20(moonToken).balanceOf(address(this)).sub(beforeBalance);
            IERC20(moonToken).safeTransfer(address(_to), _diffAmount);
            emit MintcTokenEvent(msg.sender, _pid, _to, _diffAmount);
        }
        function onlyHarvest() external onlyOwner {
            uint256 _pid = stakePid;
            IMasterStar(masterStar).deposit(_pid, 0);
        }
        function save(uint256 _amount) external {
            address _to = confluxStarAddr;
            require(_amount > 0, "MoonFund: amount is zero");
            require(_to != address(0), "MoonFund: addr is zero address");//knownsec//验证空地址
            IERC20(moonToken).safeTransferFrom(address(msg.sender), address(this), _amount);
            IERC20(moonToken).safeTransfer(address(_to), _amount);
            reserves[msg.sender] = reserves[msg.sender].add(_amount);
            emit SaveEvent(msg.sender, _to, _amount);
        }
        function retrieve(uint256 _amount) external {
            require(_amount > 0, "MoonFund: amount is zero");
            uint256 balance = IERC20(moonToken).balanceOf(address(this));
            require(_amount < balance, "MoonFund: Balance insufficient");
            uint256 _userAmount = reserves[msg.sender];
            require(_amount <= _userAmount, "MoonFund: retrieve amount overflow");
            reserves[msg.sender] = reserves[msg.sender].sub(_amount);
            IERC20(moonToken).safeTransfer(address(msg.sender), _amount);
            emit RetrieveEvent(msg.sender, _amount);
        }
        function balanceOf() external view returns(uint256){
           return IERC20(moonToken).balanceOf(address(this));
        }
    }
```

Knownsec

5. 附录 B：漏洞风险评级标准

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

Knownsec

6. 附录 C：漏洞测试工具简介

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

Knownsec

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 处合理约的字节码，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. 知道创宇渗透测试人员专用工具包

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。

Knownsec

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