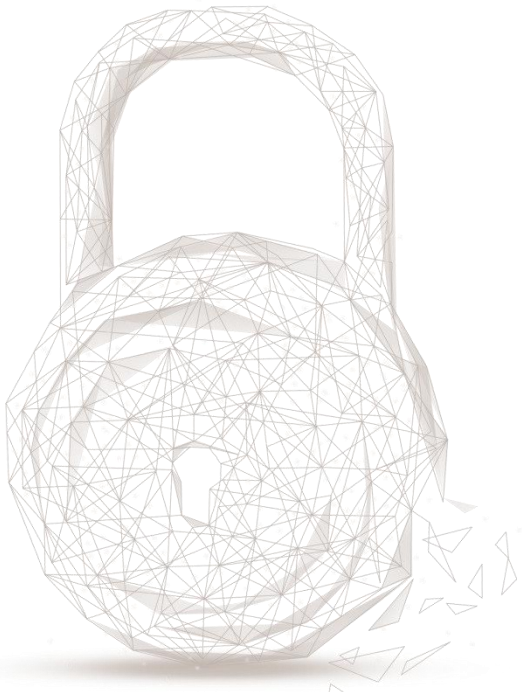
成都链安
BEOSIN

智能合约安全审计报告



审计编号：202102261438

报告查询名称：XF

审计合约名称	审计合约地址	审计合约链接
xFarmer	0xe0fe25eefcfcaddef844fe30b8be1d68ac6b7af3	https://hecoinfo.com/address/0xe0fe25eefcfcaddef844fe30b8be1d68ac6b7af3#code
StakingRewards(XF-XF)	0xA7b4E0c598305FC695b837A3D5E8Cfe121DED34b	https://hecoinfo.com/address/0xA7b4E0c598305FC695b837A3D5E8Cfe121DED34b#code
StakingRewards(XF-USDT)	0xb2D4688598aAd83Be3bF9243487817125E1dE95C	https://hecoinfo.com/address/0xb2D4688598aAd83Be3bF9243487817125E1dE95C#code
StakingRewards(XF-HT)	0x64C9fc836c5EBe8483814F669f0E34085d1cdF4f	https://hecoinfo.com/address/0x64C9fc836c5EBe8483814F669f0E34085d1cdF4f#code
StakingRewards(XF-HUSD)	0x848236841886459a308Fe180FDAF2C3dd83843c6	https://hecoinfo.com/address/0x848236841886459a308Fe180FDAF2C3dd83843c6#code
StakingRewards(XF-HBTC)	0xcb3440B517533c08b84CaBD8Cf8B620A0d131F29	https://hecoinfo.com/address/0xcb3440B517533c08b84CaBD8Cf8B620A0d131F29#code

合约审计开始日期：2021.02.26

合约审计完成日期：2021.02.26

审计结果：通过

审计团队：成都链安科技有限公司

审计类型及结果：

序号	审计类型	审计子项	审计结果
1	代码规范审计	编译器版本安全审计	通过
		弃用项审计	通过
		冗余代码审计	通过
		require/assert 使用审计	通过
		gas 消耗审计	通过
2	通用漏洞审计	整型溢出审计	通过

		重入攻击审计	通过
		伪随机数生成审计	通过
		交易顺序依赖审计	通过
		拒绝服务攻击审计	通过
		函数调用权限审计	通过
		call/delegatecall 安全审计	通过
		返回值安全审计	通过
		tx.origin 使用安全审计	通过
		重放攻击审计	通过
		变量覆盖审计	通过
3	业务审计	业务逻辑审计	通过
		业务实现审计	通过

备注：审计意见及建议请见代码注释。

审计结果说明：

本公司采用形式化验证，静态分析，动态分析，典型案例测试和人工审核的方式对XF项目智能合约代码规范性，安全性以及业务逻辑三个方面进行多维度全面的安全审计。**经审计，XF项目智能合约通过所有检测项，合约审计结果为通过。**以下为本合约详细审计信息。

1. 代码规范审计

1. 编译器版本安全审计

老版本的编译器可能会导致各种已知的安全问题，建议开发者在代码中指定合约代码采用最新的编译器版本，并消除编译器告警。

- **安全建议：无**
- **审计结果：通过**

2. 弃用项审计

Solidity智能合约开发语言处于快速迭代中，部分关键字已被新版本的编译器弃用，如throw，years等，为了消除其可能导致的隐患，合约开发者不应该使用当前编译器版本已弃用的关键字。

- **安全建议：无**
- **审计结果：通过**

3. 冗余代码审计

智能合约中的冗余代码会降低代码可读性，并可能需要消耗更多的gas用于合约部署，建议消除冗余代码。

- **安全建议：无**
- **审计结果：通过**

4. require/assert 使用审计

Solidity使用状态恢复异常来处理错误。这种机制将会撤消对当前调用(及其所有子调用)中的状态所做的所有更改，并向调用者标记错误。函数assert和require可用于检查条件并在条件不满足时抛出异常。assert函数只能用于测试内部错误，并检查非变量。require函数用于确认条件有效性，例如输入变量，或合约状态变量是否满足条件，或验证外部合约调用的返回值。

- **安全建议：无**
- **审计结果：通过**

5. gas 消耗审计

Heco虚拟机执行合约代码需要消耗gas，当gas不足时，代码执行会抛出out of gas异常，并撤销所有状态变更。合约开发者需要控制代码的gas消耗，避免因为gas不足导致函数执行一直失败。

- **安全建议：无**
- **审计结果：通过**

2. 通用漏洞审计

1. 整型溢出审计

整型溢出是很多语言都存在的安全问题，它们在智能合约中尤其危险。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uint类型时，0减去1会下溢得到最大数字值。溢出情况会导致不正确的结果，特别是如果其可能的结果未被预期，可能会影响程序的可靠性和安全性。

- **安全建议：无**
- **审计结果：通过**

2. 重入攻击审计

重入漏洞是最典型的智能合约漏洞，该漏洞原因是Solidity中的call.value()函数在被用来发送HT的时候会消耗它接收到的所有gas，当调用call.value()函数发送HT的逻辑顺序存在错误时，就会存在重入攻击的风险。

- **安全建议：无**
- **审计结果：通过**

3. 伪随机数生成审计

智能合约中可能会使用到随机数，在solidity下常见的是用block区块信息作为随机因子生成，但是这样使用是不安全的，区块信息是可以被矿工控制或被攻击者在交易时获取到，这类随机数在一定程度上是可预测或可碰撞的，比较典型的例子就是fomo3d的airdrop随机数可以被碰撞。

- **安全建议：无**
- **审计结果：通过**

4. 交易顺序依赖审计

在Heco的交易打包执行过程中，面对相同难度的交易时，矿工往往会选择gas费用高的优先打包，因此用户可以指定更高的gas费用，使自己的交易优先被打包执行。

- **安全建议：无**
- **审计结果：通过**

5. 拒绝服务攻击审计

拒绝服务攻击，即Denial of Service，可以使目标无法提供正常的服务。在Heco智能合约中也会存在此类问题，由于智能合约的不可更改性，该类攻击可能使得合约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有很多种，包括在作为交易接收方时的恶意revert，代码设计缺陷导致gas耗尽等等。

- **安全建议：无**
- **审计结果：通过**

6. 函数调用权限审计

智能合约如果存在高权限功能，如：铸币，自毁，change owner等，需要对函数调用做权限限制，避免权限泄露导致的安全问题。

- **安全建议：无**
- **审计结果：通过**

7. call/delegatecall安全审计

Solidity中提供了call/delegatecall函数来进行函数调用，如果使用不当，会造成call注入漏洞，例如call的参数如果可控，则可以控制本合约进行越权操作或调用其他合约的危险函数。

- **安全建议：无**
- **审计结果：通过**

8. 返回值安全审计

在Solidity中存在transfer()，send()，call.value()等方法中，transfer转账失败交易会回滚，而send和call.value转账失败会return false，如果未对返回做正确判断，则可能会执行到未预期的逻辑;另外在TRC20 Token的transfer/transferFrom功能实现中，也要避免转账失败return false的情况，以免造成假充值漏洞。

- **安全建议：无**
- **审计结果：通过**

9. tx.origin使用安全审计

在Heco智能合约的复杂调用中，tx.origin表示交易的初始创建者地址，如果使用tx.origin进行权限判断，可能会出现错误;另外，如果合约需要判断调用方是否为合约地址时则需要使用tx.origin，不能使用extcodesize。

- **安全建议：无**
- **审计结果：通过**

10. 重放攻击审计

重放攻击是指如果两份合约使用了相同的代码实现，并且身份鉴权在传参中，当用户在向一份合约中执行一笔交易，交易信息可以被复制并且向另一份合约重放执行该笔交易。

- **安全建议：无**
- **审计结果：通过**

11. 变量覆盖审计

Heco存在着复杂的变量类型，例如结构体，动态数组等，如果使用不当，对其赋值后，可能导致覆盖已有状态变量的值，造成合约执行逻辑异常。

- **安全建议：无**
- **审计结果：通过**

3. 业务审计

3.1 xFarmer代币合约审计

(1) XF代币基本信息

代币名称	xFarmer
代币简称	XF
代币精度	18
代币总量	初始总量1万，不可销毁，可铸币，总量上限为2亿
代币类型	HRC20

表格 1 代币基本信息

(2) HRC20 代币标准函数

- **业务描述：**xFarmer 合约所实现的是一个标准的 HRC20 代币，其相关函数符合 HRC20 代币标准规范。需要注意的是，用户可以使用 approve 函数设置对指定地址的授权值，但为了避免多重授权，建议在需要修改授权值时，不要直接使用 approve 函数进行修改，而是使用 increaseAllowance 和 decreaseAllowance 函数对当前授权值进行增加和减少。
- **相关函数：**name, symbol, decimals, totalSupply, balanceOf, allowance, transfer, transferFrom, approve, increaseAllowance, decreaseAllowance
- **安全建议：无**
- **审计结果：通过**

(3) mint 函数

- **业务描述：**如下图所示，合约的 minter 可调用此函数向指定地址铸币，代币总量上限为 2.1 亿。

```
261     function mint(address _account, uint256 _amount) public {
262         require(minters[msg.sender], "XF: !minter");
263         _mint(_account, _amount);
264     }
```

图 1 mint 函数源码

- **相关函数：**mint，_mint，_beforeTokenTransfer
- **安全建议：无**
- **审计结果：通过**

(4) 相关治理函数

- **业务描述：**合约实现了 addMinter，removeMinter，setPendingGov，acceptGov 等函数用于合约治理。setPendingGov 用于设置预备管理员，acceptGov 用于预备管理员接收管理员权限；合约的管理员可以通过 addMinter 和 removeMinter 增加或移除合约的 minter。

```
function addMinter(address _minter) public onlyGov {
    minters[_minter] = true;
}

/**
 * Remove minter
 * @param _minter minter
 */
function removeMinter(address _minter) public onlyGov {
    minters[_minter] = false;
}
```

图 2 addMinter，removeMinter 函数源码

```
function setPendingGov(address _pendingGov)
    external
    onlyGov
{
    address oldPendingGov = pendingGov;
    pendingGov = _pendingGov;
    emit NewPendingGov(oldPendingGov, _pendingGov);
}

/**
 * Lets msg.sender accept governance
 */
function acceptGov()
    external {
    require(msg.sender == pendingGov, "XF: !pending");
    address oldGov = governance;
    governance = pendingGov;
    pendingGov = address(0);
    emit NewGov(oldGov, governance);
}
```

图 3 setPendingGov，acceptGov 函数源码

- **相关函数：**addMinter，removeMinter，setPendingGov，acceptGov
- **安全建议：**无
- **审计结果：**通过

3.2 StakingReward 合约审计

项目所有抵押奖励池代码相同，仅在合约部署时设置的抵押贷币不同

(1) 抵押初始化

- **业务描述：**合约的“抵押-奖励”模式需要初始化相关参数（奖励比例rewardRate，首次更新时间lastUpdateTime，阶段完成时间periodFinish），通过指定的奖励分配管理员地址rewardDistribution调用notifyRewardAmount函数，输入初始用于计算奖励比例的奖励数值reward，初始化抵押与奖励相关参数。

```
651     function notifyRewardAmount(uint256 reward) external onlyRewardsDistribution updateReward(address(0)) {
652         if (block.timestamp >= periodFinish) {
653             rewardRate = reward.div(rewardsDuration);
654         } else {
655             uint256 remaining = periodFinish.sub(block.timestamp);
656             uint256 leftover = remaining.mul(rewardRate);
657             rewardRate = reward.add(leftover).div(rewardsDuration);
658         }
659 
660         // Ensure the provided reward amount is not more than the balance in the contract.
661         // This keeps the reward rate in the right range, preventing overflows due to
662         // very high values of rewardRate in the earned and rewardsPerToken functions;
663         // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
664         uint balance = rewardsToken.balanceOf(address(this));
665         require(rewardRate <= balance.div(rewardsDuration), "Provided reward too high");
666 
667         lastUpdateTime = block.timestamp;
668         periodFinish = block.timestamp.add(rewardsDuration);
669         emit RewardAdded(reward);
670     }
```

图 4 notifyRewardAmount 函数源码截图

- **相关函数：**notifyRewardAmount
- **安全建议：无**
- **审计结果：通过**

(2) 抵押代币

- **业务描述：**合约实现了stake函数用于抵押代币，用户预先授权该合约地址，通过调用合约中的transferFrom函数，每次调用该函数抵押代币时通过修饰器updateReward更新奖励相关数据。

```
619     function stake(uint256 amount, address account) external nonReentrant updateReward(msg.sender) {
620         require(amount > 0, "Cannot stake 0");
621         _totalSupply = _totalSupply.add(amount);
622         _balances[msg.sender] = _balances[msg.sender].add(amount);
623         stakingToken.safeTransferFrom(msg.sender, address(this), amount);
624         emit Staked(msg.sender, amount);
625     }
```

图 5 stake 函数源码截图

- **相关函数：**stake，updateReward
- **安全建议：stake函数中参数account并未使用，冗余代码，建议删除**
- **修复结果：忽略**
- **审计结果：通过**

(3) 提取抵押代币

- **业务描述：**合约实现了withdraw函数用于提取已抵押的代币，通过调用合约中的transfer函数，合约地址将指定数量的代币转至函数调用者（用户）地址；每次调用该函数提取代币时通过修饰器updateReward更新奖励相关数据。

```
627     function withdraw(uint256 amount) public nonReentrant updateReward(msg.sender) {
628         require(amount > 0, "Cannot withdraw 0");
629         _totalSupply = _totalSupply.sub(amount);
630         _balances[msg.sender] = _balances[msg.sender].sub(amount);
631         stakingToken.safeTransfer(msg.sender, amount);
632         emit Withdrawn(msg.sender, amount);
633     }
```

图 6 withdraw 函数源码截图

- **相关函数：**withdraw，updateReward
- **安全建议：无**
- **审计结果：通过**

(4) 领取抵押奖励

- **业务描述：**合约实现了getReward函数用于领取抵押奖励，通过调用合约中的transfer函数，合约地址将指定数量（用户的全部抵押奖励）的代币转至函数调用者（用户）地址；每次调用该函数抵押代币时通过修饰器updateReward更新奖励相关数据。

```
635     function getReward() public nonReentrant updateReward(msg.sender) {
636         uint256 reward = rewards[msg.sender];
637         if (reward > 0) {
638             rewards[msg.sender] = 0;
639             rewardsToken.safeTransfer(msg.sender, reward);
640             emit RewardPaid(msg.sender, reward);
641         }
642     }
```

图 7 getReward 函数源码截图

- **相关函数：**getReward，updateReward
- **安全建议：无**
- **审计结果：通过**

(5) 提取所有代币并领取奖励

- **业务描述：**合约实现了exit函数用于调用者提取所有代币并领取奖励，调用withdraw函数提取全部已抵押的代币，调用getReward函数领取完调用者的抵押奖励，结束“抵押-奖励”模式参与。此时用户地址由于其已抵押代币数量为空，无法获得新的抵押奖励。

```
644     function exit() external {
645         withdraw(_balances[msg.sender]);
646         getReward();
647     }
```

图 8 exit 函数源码截图

- **相关函数：**exit，withdraw，getReward
- **安全建议：无**
- **审计结果：通过**

(6) 奖励相关数据查询功能

- **业务描述：**合约用户可通过调用lastTimeRewardApplicable函数查询当前时间戳与阶段完成时间中最早的时间戳；调用rewardPerToken函数可查询每个抵押代币可获得的抵押奖励；调用earned函数可查询指定地址所获取的总抵押奖励。

```
595     function lastTimeRewardApplicable() public view returns (uint256) {
596         return Math.min(block.timestamp, periodFinish);
597     }
598
599     function rewardPerToken() public view returns (uint256) {
600         if (_totalSupply == 0) {
601             return rewardPerTokenStored;
602         }
603         return
604         rewardPerTokenStored.add(
605             lastTimeRewardApplicable().sub(lastUpdateTime).mul(rewardRate).mul(1e18).div(_totalSupply)
606         );
607     }
608
609     function earned(address account) public view returns (uint256) {
610         return _balances[account].mul(rewardPerToken().sub(userRewardPerTokenPaid[account])).div(1e18).add(rewards[account]);
611     }
```

图 9 相关函数源码

- **相关函数：**lastTimeRewardApplicable，rewardPerToken，earned
- **安全建议：无**
- **审计结果：通过**

4. 结论

Beosin(成都链安)对 XF 项目的智能合约的设计和代码实现进行了详细的审计。审计团队在审计过程中发现的问题均已告知项目方并就修复结果达成一致，XF 项目的智能合约的总体审计结果是**通过。**

成都链安

BEOSIN

官方网址

https://lianantech.com

电子邮箱

vaas@lianantech.com

微信公众号