智能合约安全审计报告

Smart contract security audit report

审计编号：20220725002

审计合约名称:	审计合约地址:	审计合约链接地址:
FIRE	0xF8c13eBDD4E7a78603ECA2c3043E4761D93074b1	https://bscscan.com/address/0xF8c13eBDD4E7a78603ECA2c3043E4761D93074b1#code

报告查询名称：FIRE

合约审计开始日期：2022.07.25

合约审计完成日期：2022.07.27

审计结果：通过

审计团队：链盾科技

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

免责声明: 本次审计仅针对本报告载明的审计类型及结果表中给定的审计类型范围进行审计，其他未知安全漏洞不在本次审计责任范围之内。成都链盾科技仅根据本报告出具前已经存在或发生的攻击或 漏洞出具本报告，对于出具以后存在或发生的新的攻击或漏洞，成都链盾科技无法判断其对智能合约安全状况可能的影响，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，仅基于合约提供 者在本报告出具前已向成都链盾科技提供的文件和资料，且该部分文件和资料不存在任何缺失、被篡改、删减或隐瞒的前提下作出的;如提供的文件和资料存在信息缺失、被篡改、删减、隐瞒或反映的 情况与实际情况不符等情况或提供文件和资料在本报告出具后发生任何变动的，成都链盾科技对由此而导致的损失和不利影响不承担任何责任。成都链盾科技出具的本审计报告系根据合约提供者提供的 文件和资料依靠成都链盾科技现掌握的技术而作出的，由于任何机构均存在技术的局限性。

成都链盾科技作出的本审计报告仍存在无法完整检测出全部风险的可能性，成都链盾科技对由此产生的损失不承担任何责任。

本声明最终解释权归成都链盾科技所有。

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 FIRE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，FIRE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。

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

BSC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

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 BSC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 BSC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 BEP20 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 BEP20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

BSC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

代币信息审计

(1) FIRE代币基本信息

代币名称	FIRE
代币简称	FIRE
代币精度	18
代币总量	5000000枚
代币类型	BEP20

表格 1 代币基本信息

(2) BEP20 代币标准函数

业务描述:FIRE合约所实现的是一个标准的 BEP20代币，其相关函数符合BEP20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, burn

安全建议:无

审计结果:通过

(3) 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。setGovernance 用于转移管理员权限，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 {
    require(msg.sender == governance, "!governance");
    minters[_minter] = true;
}

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

图 1 addMinter，removeMinter 函数源码

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

图 2 setGovernance 函数源码

相关函数:addMinter，removeMinter，setGovernance

安全建议:合约没有 mint 函数，不可铸币，minter 权限无用，addMinter 和 removeMinter 函数属于冗余代码，建议删除

修复结果:忽略

审计结果:通过

结论

（成都链盾)FIRE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，FIRE项目的智能合约的总体审计结果是通过。

电子邮箱：support@linkeytech.com