智能合约安全审计报告

[2021]

目录

1 前言

慢雾安全团队于 2021.06.07，收到 MaskWallet 团队对 Mask InitialTwitterOffering 智能合约安全审计的申请，慢雾安全团队根据项目特点制定如下审计方案。

慢雾安全团队将采用“白盒为主，黑灰为辅”的策略，以最贴近真实攻击的方式，对项目进行安全审计。

慢雾科技项目测试方法：

测试方法	说明
黑盒测试	站在外部从攻击者角度进行安全测试。
灰盒测试	通过脚本工具对代码模块进行安全测试，观察内部运行状态，挖掘弱点。
白盒测试	基于项目的源代码，进行脆弱性分析和漏洞挖掘。

慢雾科技漏洞风险等级：

漏洞等级	说明
严重漏洞	严重漏洞会对项目的安全造成重大影响，强烈建议修复严重漏洞。
高危漏洞	高危漏洞会影响项目的正常运行，强烈建议修复高危漏洞。
中危漏洞	中危漏洞会影响项目的运行，建议修复中危漏洞。
低危漏洞	低危漏洞可能在特定场景中会影响项目的业务操作，建议项目方自行评估和考虑这些问题是否需要修复。
弱点	理论上存在安全隐患，但工程上极难复现。
增强建议	编码或架构存在更好的实践方法。

2 审计方法

慢雾安全团队智能合约安全审计流程包含两个步骤:

- 使用开源或内部自动化分析的工具对合约代码中常见的安全漏洞进行扫描和测试。
- 人工审计代码的安全问题，通过人工分析合约代码，发现代码中潜在的安全问题。

如下是合约代码审计过程中慢雾安全团队会重点审查的漏洞列表:

（其他未知的安全漏洞及审计项不包含在本次审计责任范围）

- 重入漏洞
- 重放漏洞
- 重排漏洞
- 短地址漏洞
- 拒绝服务漏洞
- 交易顺序依赖漏洞
- 条件竞争漏洞
- 权限控制漏洞
- 整数上溢/下溢漏洞
- 时间戳依赖漏洞
- 未声明的存储指针漏洞
- 算术精度误差漏洞
- tx.origin身份验证漏洞
- 假充值漏洞
- 变量覆盖漏洞
- Gas优化审计
- 恶意 Event 事件审计
- 冗余的回调函数
- 不安全的外部调用审计

- 函数状态变量可见性审计
- 业务逻辑缺陷审计
- 变量声明及作用域审计

3 项目概要

3.1 项目介绍

类型：Token + ITO Module

代码行数：565 (包含外部导入部分)

复杂度：中等

工作量：6 个工作日，智能合约安全审计标准、相关说明及审计后证书查询：https://www.slowmist.com/service-smart-contract-security-audit.html

项目地址：

https://github.com/DimensionDev/InitialTwitterOffering

Audit based branch: https://github.com/DimensionDev/InitialTwitterOffering/tree/master

Initial audit commit: 6b95eca79e0436ff16f01610c2543e7a227cbbb0

Last audit commit: e05d299e80c1910fadd90bf021c52ced8b13f891

请确认以上项目地址是否正确。

出具报告前需要项目方在区块浏览器开源验证，并给到相关合约地址。

3.2 漏洞信息

如下是本次审计发现的漏洞及漏洞的修复状态信息：

NO	标题	漏洞类型	漏洞等级	漏洞状态

NO	标题	漏洞类型	漏洞等级	漏洞状态
N1	swap时的password效验可被绕过	业务逻辑缺陷审计	建议	已确认

4 审计详情

4.1 合约基础信息

如下是合约主网地址：

https://etherscan.io/address/0xBe63b9ee7055Ea8A5691e93D31E949C03A3e2F54#code

4.2 函数可见性分析

在审计过程中，慢雾安全团队对核心合约的函数可见性进行分析，结果如下：

IQLF			
Function Name	Visibility	Mutability	Modifiers
supportsInterface	external	-	-
ifQualified	external	-	-
logQualified	external	can modify state	-
supportsInterface	external	-	-

HappyTokenPool			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-

HappyTokenPool			
fill_pool	public	payable	-
swap	public	payable	-
check_availability	external	-	-
claim	external	can modify state	-
setUnlockTime	public	can modify state	-
destruct	public	can modify state	-
withdraw	public	can modify state	-

Context			
Function Name	Visibility	Mutability	Modifiers
_msgSender	internal	-	-
_msgData	internal	-	-

Ownable			
Function Name	Visibility	Mutability	Modifiers
_msgSender	internal	-	-
_msgData	internal	-	-
constructor	internal	can modify state	-
owner	public	-	-
renounceOwnership	public	can modify state	onlyOwner
transferOwnership	public	can modify state	onlyOwner

QLF			
Function Name	Visibility	Mutability	Modifiers
constructor	internal	can modify state	-
owner	public	-	-
renounceOwnership	public	can modify state	onlyOwner
transferOwnership	public	can modify state	onlyOwner
_msgSender	internal	-	-
_msgData	internal	-	-
ifQualified	external	-	-
logQualified	external	can modify state	-
supportsInterface	external	-	-
supportsInterface	external	-	-
constructor	public	can modify state	-
get_start_time	public	-	-
set_start_time	public	can modify state	onlyOwner
ifQualified	public	-	-
logQualified	public	can modify state	-
supportsInterface	external	-	-

4.3 漏洞详情

[N1] [建议] swap时的password效验可被绕过

漏洞类型: 业务逻辑缺陷审计

详细内容

contracts/ito.sol#178-255,

```
function swap (bytes32 id,
               bytes32 verification,
               uint256 exchange_addr_i,
               uint128 input_total,
               bytes32[] memory data)
public payable returns (uint256 swapped) {

    Pool storage pool = pool_by_id[id];
    Packed1 memory packed1 = pool.packed1;
    Packed2 memory packed2 = pool.packed2;
    Packed3 memory packed3 = pool.packed3;
    {
        bool qualified;
        string memory errorMsg;
        (qualified, errorMsg) =
IQLF(packed1.qualification_addr).logQualified(msg.sender, data);
        require(qualified, errorMsg);
    }
    require (packed3.start_time + base_time < block.timestamp, "Not started.");
    require (packed3.end_time + base_time > block.timestamp, "Expired.");
    // sha3(sha3(passowrd)[:40] + msg.sender) so that the raw password will never
appear in the contract
    require (verification == keccak256(abi.encodePacked(uint256(packed1.password),
msg.sender)),
             'Wrong Password');

    // revert if the pool is empty
    require (packed2.total_tokens > 0, "Out of Stock");

    address exchange_addr = pool.exchange_addrs[exchange_addr_i];
    uint256 ratioA = pool.ratios[exchange_addr_i*2];
    uint256 ratioB = pool.ratios[exchange_addr_i*2 + 1];
    // check if the input is enough for the desired transfer
    if (exchange_addr == DEFAULT_ADDRESS) {
        require(msg.value == input_total, 'No enough ether.');
    }
```

```
    uint128 swapped_tokens =
SafeCast.toUint128(SafeMath.div(SafeMath.mul(input_total, ratioB), ratioA));
    require(swapped_tokens > 0, "Better not draw water with a sieve");

    if (swapped_tokens > packed2.limit) {
        // don't be greedy - you can only get at most limit tokens
        swapped_tokens = packed2.limit;
        input_total = SafeCast.toUint128(SafeMath.div(SafeMath.mul(packed2.limit,
ratioA), ratioB));           // Update input_total
    } else if (swapped_tokens > packed2.total_tokens ) {
        // if the left tokens are not enough
        swapped_tokens = packed2.total_tokens;
        input_total =
SafeCast.toUint128(SafeMath.div(SafeMath.mul(packed2.total_tokens , ratioA),
ratioB));    // Update input_total
        // return the eth
        if (exchange_addr == DEFAULT_ADDRESS)
            payable(msg.sender).transfer(msg.value - input_total);
    }
    require(swapped_tokens <= packed2.limit);
// make sure again
    pool.exchanged_tokens[exchange_addr_i] =
SafeCast.toUint128(SafeMath.add(pool.exchanged_tokens[exchange_addr_i],
                                                                input_total));
// update exchanged

    // penalize greedy attackers by placing duplication check at the very last
    require (pool.swapped_map[msg.sender] == 0, "Already swapped");

    // update the remaining tokens and swapped token mapping
    pool.packed2.total_tokens = SafeCast.toUint128(SafeMath.sub(packed2.total_tokens,
swapped_tokens));
    pool.swapped_map[msg.sender] = swapped_tokens;

    // transfer the token after state changing
    // ETH comes with the tx, but ERC20 does not - INPUT
    if (exchange_addr != DEFAULT_ADDRESS) {
        IERC20(exchange_addr).safeTransferFrom(msg.sender, address(this),
input_total);
    }

    // Swap success event
    emit SwapSuccess(id, msg.sender, exchange_addr, packed3.token_address,
input_total, swapped_tokens);
```

```
    // if unlock_time == 0, transfer the swapped tokens to the recipient address
(msg.sender) - OUTPUT
    // if not, claim() needs to be called to get the token
    if (packed3.unlock_time == 0) {
        IERC20(packed3.token_address).safeTransfer(msg.sender, swapped_tokens);
        emit ClaimSuccess(id, msg.sender, block.timestamp, swapped_tokens,
packed3.token_address);
    }

    return swapped_tokens;
}
```

其中关于密码的判断部分，

```
// sha3(sha3(passowrd)[:40] + msg.sender) so that the raw password will never appear
in the contract
require (verification == keccak256(abi.encodePacked(uint256(packed1.password),
msg.sender)),
         'Wrong Password');
```

packed1.password 可以在fill_pool之后被读取，从而构造出正确的verification，绕过此判断。

解决方案

此类判断只能防住使用Mask插件进行交易的参与者，对恶意的使用者无法有效防御。可以通过白名单等方式来增强防御。

漏洞状态

已确认；已使用白名单和随机数来增强防御。

5 审计结果

审计编号	审计团队	审计日期	审计结果
0X002106150003	SlowMist Security Team	2021.06.07 - 2021.06.15	通过

总结：

慢雾安全团队采用人工结合内部工具对代码进行分析，审计期间发现了 1 个增强建议。其中 1 个增强建议已确认；其它所有漏洞均已修复。

6 声明

厦门慢雾科技有限公司(下文简称 ”慢雾”) 仅就本报告出具前项目方已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后项目方发生或存在的未知漏洞及安全事件，慢雾无法判断其安全状况，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慢雾提供的文件和资料(简称“已提供资料”)。慢雾假设: 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，慢雾对由此而导致的损失和不利影响不承担任何责任，慢雾仅对该项目的安全情况进行约定内的安全审计并出具了本报告，慢雾不对该项目背景及其他情况进行负责。

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号