智能合约安全审计报告

目录

1 前言

慢雾安全团队于 2020 年 11 月 04 日，收到 WePiggy 团队对 WePiggy 安全审计评估的申请，根据项目特点慢雾安全团队制定如下审计方案。

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 DeFi 项目测试方法：

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 DeFi 漏洞风险等级：

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:

（其他未知安全漏洞不包含在本次审计责任范围）

- 重入攻击
- 重放攻击
- 重排攻击
- 短地址攻击
- 拒绝服务攻击
- 交易顺序依赖
- 条件竞争攻击
- 权限控制攻击
- 整数上溢/下溢攻击
- 时间戳依赖攻击
- Gas 使用，Gas 限制和循环
- 冗余的回调函数
- 不安全的接口使用
- 函数状态变量的显式可见性
- 逻辑缺陷
- 未声明的存储指针
- 算术精度误差
- tx.origin 身份验证
- 假充值漏洞
- 变量覆盖

3. 项目背景

3.1 项目介绍

WePiggy 是一个开源，非托管的加密资产借贷市场协议。在 WePiggy 的市场上，用户可存入特定的加密资产赚取利息，也可以支付一定的利息借取某种加密资产。

项目官网地址:

https://wepiggy.com

审计版本代码:

https://github.com/WePiggy/wepiggy-contracts/tree/599e854a5cab44de2355a1bcda09b170c20

ccc35/contracts

已提供的文档：

https://www.yuque.com/zgryhn/fg3t76/qccll7

修复版本代码：

https://github.com/WePiggy/wepiggy-contracts/tree/38c1e240578e11621b665007592a21d2c8e611ec

4. 代码概述

4.1 合约可见性分析

在审计过程中，慢雾安全团队对核心合约的可见性进行分析，结果如下：

PiggyBreeder			
Function Name	Visibility	Mutability	Modifiers
poolLength	External	-	-
usersLength	External	-	-
setDevAddr	Public	Can modify state	-
setMigrator	Public	Can modify state	onlyOwner
setEnableClaimBlock	Public	Can modify state	onlyOwner
setReduceIntervalBlock	Public	Can modify state	onlyOwner
setAllocPoint	Public	Can modify state	onlyOwner
setReduceRate	Public	Can modify state	onlyOwner
setDevMiningRate	Public	Can modify state	onlyOwner
replaceMigrate	Public	Can modify state	onlyOwner
migrate	Public	Can modify state	onlyOwner
safePiggyTransfer	Internal	Can modify state	-
getPiggyPerBlock	Public	-	-
getMultiplier	Public	-	-
allPendingPiggy	External	-	-
pendingPiggy	External	-	-
_pending	Internal	-	-

massUpdatePools	Public	Can modify state	-
updatePool	Public	Can modify state	-
add	Public	Can modify state	onlyOwner
stake	Public	Can modify state	-
unStake	Public	Can modify state	-
claim	Public	Can modify state	-
emergencyWithdraw	Public	Can modify state	-

FundingHolder			
Function Name	Visibility	Mutability	Modifiers
transfer	Public	Can modify state	onlyOwner

FundingManager			
Function Name	Visibility	Mutability	Modifiers
safePiggyTransfer	Internal	Can modify state	-
addFunding	Public	Can modify state	onlyOwner
setFunding	Public	Can modify state	onlyOwner
getPendingBalance	Public	-	-
claim	Public	Can modify state	-

WePiggyToken			
Function Name	Visibility	Mutability	Modifiers
mint	Public	Can modify state	-
_transfer	Internal	Can modify state	-
delegates	External	-	-
delegate	External	Can modify state	-
delegateBySig	External	Can modify state	-
getCurrentVotes	External	-	-
getPriorVotes	External	-	-
_delegate	Internal	Can modify state	-
_moveDelegates	Internal	Can modify state	-
_writeCheckpoint	Internal	Can modify state	-

safe32	Internal	-	-
getChainId	Internal	-	-

Timelock			
Function Name	Visibility	Mutability	Modifiers
setDelay	Public	Can modify state	-
acceptAdmin	Public	Can modify state	-
setPendingAdmin	Public	Can modify state	-
queueTransaction	Public	Can modify state	-
cancelTransaction	Public	Can modify state	-
executeTransaction	Public	Payable	-
getBlockTimestamp	Internal	-	-
receive()	External	Payable	-

4.2 合约信息

Contract Name	Contract Address
WePiggyToken	0x4C78015679FabE22F6e02Ce8102AFbF7d93794eA
FundingHolder[InsurancePayment]	0xb205d0AeF84C666FBBe441C61DC04fEb844444E6
FundingHolder[ResourceExpansion]	0x7a603D06007fc09f896Fb75644365AB091A7b91a
FundingHolder[TeamVote]	0x8C14A40c488E16b055bc4e250563B5480047f01E
FundingHolder[TeamSpending]	0x209EE6f924a39BdDC9A57c0E263dd5E29CEAc78A
FundingHolder[CommunityRewards]	0x1fe0F15546858d9A1E84CE2E7908b160608267c5
FundingManager	0xf18D727C034f47AE2C0FE221C1cf4A15f0557b5F
PiggyBreeder	0x451032C55F813338b6e73c1c4B24217614165454
TimeLock	0x311aEA58Ca127B955890647413846E351df32554

4.3 代码审计

4.3.1 高危漏洞

4.3.1.1 治理双花漏洞

存在委托治理双花的漏洞，治理合约允许 Token 持有者将投票权转移给被委托人。但是这里有一个缺陷，使当 Token 持有者从钱包中转移代币时，投票权仍保留在被授权者的手中。在这种情况下，应该撤销被委托人相应的投票权，否则投票权可以通过委托和转让 Token 来增发。

- wepiggy-contracts/contracts/token/WePiggyToken.sol

```
// It is suggested to fix the delegation double spending bug by using the following code:
function _transfer(address sender, address recipient, uint256 amount) internal override virtual {
        super._transfer(sender, recipient, amount);
        _moveDelegates(_delegates[sender], _delegates[recipient], amount);
    }
```

修复状态: 该问题在 commit: da7b30e5098721119962eb5678a7d7b0f37434a4 中已修复。

4.3.1.2 业务逻辑错误

迁移的时候没有处理和保存用户的“rewardDebt”，并将用户的“rewardDebt”设置为 0。这是一个业务逻辑错误，会影响用户的历史“rewardDebt”。建议在迁移过程中保留用户的奖励。

- contracts/farm/PiggyBreeder.sol

```
function migrate(uint256 _pid, uint256 _targetPid) public onlyOwner {

        PoolInfo storage pool = poolInfo[_pid];
        IMigrator migrator = pool.migrator;
        require(address(migrator) != address(0), "migrate: no migrator");

        IERC20 lpToken = pool.lpToken;
        uint256 bal = lpToken.balanceOf(address(this));
        lpToken.safeApprove(address(migrator), bal);
        (IERC20 newLpToken, uint mintBal) = migrator.migrate(lpToken);
```

```
        PoolInfo storage targetPool = poolInfo[_targetPid];
        IERC20 targetToken = targetPool.lpToken;
        require(address(targetToken) == address(newLpToken), "migrate: bad");

        uint rate = mintBal.mul(1e12).div(bal);
        for (uint i = 0; i < userAddresses[_pid].length; i++) {

            updatePool(_targetPid);

            address addr = userAddresses[_pid][i];
            UserInfo storage user = userInfo[_pid][addr];
            UserInfo storage tUser = userInfo[_targetPid][addr];

            uint tmp = user.amount.mul(rate).div(1e12);

            tUser.amount = tUser.amount.add(tmp);
            targetPool.totalDeposit = targetPool.totalDeposit.add(tmp);
            pool.totalDeposit = pool.totalDeposit.sub(user.amount);
            user.rewardDebt = 0;
            user.amount = 0;
        }

    }
```

修复状态: 这个问题在 commit: 38c1e240578e11621b665007592a21d2c8e611ec 中已修复。

4.3.2 中危漏洞

4.3.2.1 DoS 问题

用户地址的数量没有限制。当地址数较大的时时候 for 循环将会超过 EVM 的调用深度，无法正常执行。建议通过分批量处理或使用算法来避免调用深度过大导致交易失败的问题。

- contracts/farm/PiggyBreeder.sol Line: 196

```
for (uint i = 0; i < userAddresses[_pid].length; i++) {

            updatePool(_targetPid);
```

```
            address addr = userAddresses[_pid][i];
            UserInfo storage user = userInfo[_pid][addr];
            UserInfo storage tUser = userInfo[_targetPid][addr];

            uint tmp = user.amount.mul(rate).div(1e12);

            tUser.amount = tUser.amount.add(tmp);
            targetPool.totalDeposit = targetPool.totalDeposit.add(tmp);
            pool.totalDeposit = pool.totalDeposit.sub(user.amount);
            user.rewardDebt = 0;
            user.amount = 0;
        }

    }
```

修复状态: 这个问题在 commit: 38c1e240578e11621b665007592a21d2c8e611ec 中进行了不完全的修复。

4.3.3 低危漏洞

4.3.3.1 权限过大问题

PiggyBreeder 合约中的 Owner 可以通过 add 函数添加一个新的 lpToken，但如果出现黑天鹅事件，比如恶意添加一个 lpToken，就能够通过无价值的 lpToken 来获得奖励。建议将 Owner 权限移交给治理合约或 timelock 合约进行管理。

- wepiggy-contracts/contracts/farm/PiggyBreeder.sol Line: 358-379

```
function add(uint256 _allocPoint, IERC20 _lpToken, IMigrator _migrator, bool _withUpdate) public onlyOwner {

        if (_withUpdate) {
            massUpdatePools();
        }

        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;

        //update totalAllocPoint
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
```

```
        // add poolInfo
        poolInfo.push(PoolInfo({
        lpToken : _lpToken,
        allocPoint : _allocPoint,
        lastRewardBlock : lastRewardBlock,
        accPiggyPerShare : 0,
        totalDeposit : 0,
        migrator : _migrator
        }));
    }
```

Owner 可以设置 devaddr，migrator，enable claimBlock 等变量的数据。建议将 Owner 的权限移交给治理合约或 timelock 合约进行管理。

- wepiggy-contracts/contracts/farm/PiggyBreeder.sol

```
    // Update dev address by the previous dev.
    function setDevAddr(address _devAddr) public onlyOwner {
        devAddr = _devAddr;
    }

    // Set the migrator contract. Can only be called by the owner.
    function setMigrator(uint256 _pid, IMigrator _migrator) public onlyOwner {
        poolInfo[_pid].migrator = _migrator;
    }

    // set the enable claim block
    function setEnableClaimBlock(uint256 _enableClaimBlock) public onlyOwner {
        enableClaimBlock = _enableClaimBlock;
    }

    // update reduceIntervalBlock
    function setReduceIntervalBlock(uint256 _reduceIntervalBlock, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        reduceIntervalBlock = _reduceIntervalBlock;
    }

    // Update the given pool's PIGGY allocation point. Can only be called by the owner.
    function setAllocPoint(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
```

```
        }
        //update totalAllocPoint
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);

        //update poolInfo
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // update reduce rate
    function setReduceRate(uint256 _reduceRate, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        reduceRate = _reduceRate;
    }

    // update dev mining rate
    function setDevMiningRate(uint256 _devMiningRate) public onlyOwner {
        devMiningRate = _devMiningRate;
    }

    // Migrate lp token to another lp contract.
    function replaceMigrate(uint256 _pid) public onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];
        IMigrator migrator = pool.migrator;
        require(address(migrator) != address(0), "migrate: no migrator");

        IERC20 lpToken = pool.lpToken;
        uint256 bal = lpToken.balanceOf(address(this));
        lpToken.safeApprove(address(migrator), bal);
        (IERC20 newLpToken, uint mintBal) = migrator.replaceMigrate(lpToken);

        require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");
        pool.lpToken = newLpToken;

        emit ReplaceMigrate(address(migrator), _pid, bal);
    }

    // Move lp token data to another lp contract.
    function migrate(uint256 _pid, uint256 _targetPid, uint256 begin) public onlyOwner {

        require(begin < userAddresses[_pid].length, "migrate: begin error");
```

```
        PoolInfo storage pool = poolInfo[_pid];
        IMigrator migrator = pool.migrator;
        require(address(migrator) != address(0), "migrate: no migrator");

        IERC20 lpToken = pool.lpToken;
        uint256 bal = lpToken.balanceOf(address(this));
        lpToken.safeApprove(address(migrator), bal);
        (IERC20 newLpToken, uint mintBal) = migrator.migrate(lpToken);

        PoolInfo storage targetPool = poolInfo[_targetPid];
        require(address(targetPool.lpToken) == address(newLpToken), "migrate: bad");

        uint rate = mintBal.mul(1e12).div(bal);
        for (uint i = begin; i < begin.add(20); i++) {

            if (i < userAddresses[_pid].length) {
                updatePool(_targetPid);

                address addr = userAddresses[_pid][i];
                UserInfo storage user = userInfo[_pid][addr];
                UserInfo storage tUser = userInfo[_targetPid][addr];

                if (user.amount <= 0) {
                    continue;
                }

                uint tmp = user.amount.mul(rate).div(1e12);

                tUser.amount = tUser.amount.add(tmp);
                tUser.rewardDebt = tUser.rewardDebt.add(user.rewardDebt.mul(rate).div(1e12));
                targetPool.totalDeposit = targetPool.totalDeposit.add(tmp);
                pool.totalDeposit = pool.totalDeposit.sub(user.amount);
                user.rewardDebt = 0;
                user.amount = 0;
            } else {
                break;
            }

        }

        emit Migrate(address(migrator), _pid, _targetPid, bal);

    }
```

修复状态: Owner 的权限已经移交给 timelock 合约.

权限移交的交易:

https://cn.etherscan.com/tx/0x902e0eaf5251249b03eb0ad354b7d058dbb8a4ee44407b640babd28d045a0ca2

4.3.4 增强建议

4.3.4.1 签名重放问题

delegateBySig 函数 nonce 是由用户自己传入的参数进行签名的，当用户传了一个较大的 nonce 时，当前交易无法通过校验但是相关的签名数据仍会留在链上，导致此签名可能在未来某个时间段可用。建议参考 eip-2612 进行修复。

参考：https://github.com/ethereum/EIPs/blob/master/EIPS/eip-2612.md#implementation

- wepiggy-contracts/contracts/token/WePiggyToken.sol Line: 105-146

```
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), "WePiggyToken::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "WePiggyToken::delegateBySig: invalid nonce");
        require(now <= expiry, "WePiggyToken::delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
```

修复状态：这个问题由于不直接影响项目的安全性，属于增强点，在保证签名的 nonce 准确的情况不会有该问题，因此暂时忽略.

5. 审计结果

5.1 结论

审计结果: 通过

审计编号: 0X002011110001

审计日期: 2020 年 11 月 11 日

审计团队: 慢雾安全团队

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 2 个高危漏洞，1 个中危漏洞，1 个低危漏洞，1 个增强建议。其中高危，中危，低危漏洞已经进行了修复，Owner 权限过大的问题通过 timelock 机制进行了缓解，timelock 的延迟时间为 48 小时，有 1 个增强建议暂时被忽略。

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。

慢雾科技
slow mist

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号