Smart Contract Security Audit Report

Contents

1. Executive Summary

On Nov. 25, 2020, the SlowMist security team received the WePiggy team's security audit application for WePiggy phase II code, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerabilities	Medium vulnerability will affect the operation of DeFi project. It is recommended to fix medium-risk vulnerabilities.

Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack

- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background

3.1 Project Introduction

WePiggy is an open source, non-custodial crypto asset lending market protocol. In WePiggy's market, users can deposit their crypto assets to earn interest, or borrow others by paying interests.

Project website:

https://wepiggy.com

Audit version code:

https://github.com/WePiggy/wepiggy-contracts/tree/826534cee92b01665a6f7b8258e33b5e612dee59

Fixed version code:

https://github.com/WePiggy/wepiggy-contracts/tree/844a23c63ce6e9347bc6cfafd6fb1c40de172672

4. Code Overview

4.1 Contracts Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

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	-

ATokenMigrator			
Function Name	Visibility	Mutability	Modifiers
migrate	Public	Can modify state	-
_getTokenBalance	Internal	Can modify state	-
Receive	External	Payable	-
compareStrings	Internal	-	-

ATokenMigrator			
Function Name	Visibility	Mutability	Modifiers
replaceMigrate	External	Payable	-
migrate	External	Payable	-
receive	External	Payable	-

CErc20Migrator			
Function Name	Visibility	Mutability	Modifiers
replaceMigrate	External	Can modify state	-
migrate	External	Can modify state	-

CEthMigrator			
Function Name	Visibility	Mutability	Modifiers
replaceMigrate	External	Payable	-
migrate	External	Payable	-
receive	External	Payable	-

Comptroller			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
enterMarkets	Public	Can modify state	-
addToMarketInternal	Internal	Can modify state	-
exitMarket	External	Can modify state	-
getAssetsIn	External	-	-

checkMembership	External	-	-
mintAllowed	External	Can modify state	-
mintVerify	External	Can modify state	-
redeemAllowed	External	Can modify state	-
redeemAllowedInternal	Internal	-	-
redeemVerify	External	Can modify state	-
borrowAllowed	External	Can modify state	-
borrowVerify	External	Can modify state	-
repayBorrowAllowed	External	Can modify state	-
repayBorrowVerify	External	Can modify state	-
liquidateBorrowAllowed	External	Can modify state	-
liquidateBorrowVerify	External	Can modify state	-
seizeAllowed	External	Can modify state	-
seizeVerify	External	Can modify state	-
transferAllowed	External	Can modify state	-
transferVerify	External	Can modify state	-
getAccountLiquidity	Public	-	-
getAccountLiquidityInternal	Internal	-	-
getHypotheticalAccountLiquidity	Public	-	-
getHypotheticalAccountLiquidityInternal	Internal	-	-
liquidateCalculateSeizeTokens	External	-	-
_setPriceOracle	Public	Can modify state	onlyOwner
_setCloseFactor	External	Can modify state	onlyOwner
_setCollateralFactor	External	Can modify state	onlyOwner
_setMaxAssets	External	Can modify state	onlyOwner
_setLiquidationIncentive	External	Can modify state	onlyOwner
_supportMarket	External	Can modify state	onlyOwner
_addMarketInternal	Internal	Can modify state	onlyOwner
_setMarketBorrowCaps	External	Can modify state	-
_setBorrowCapGuardian	External	Can modify state	onlyOwner
_setPauseGuardian	Public	Can modify state	onlyOwner
_setMintPaused	Public	Can modify state	-
_setBorrowPaused	Public	Can modify state	-
_setTransferPaused	Public	Can modify state	-
_setSeizePaused	Public	Can modify state	-
_setDistributeWpcPaused	Public	Can modify state	-
_setPiggyDistribution	Public	Can modify state	onlyOwner
getAllMarkets	Public	-	-

isMarketMinted	Public	-	-
isMarketListed	Public	-	-
_setMarketMinted	Public	Can modify state	-

SimplePriceOracle			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
getUnderlyingPrice	Public	-	-
setUnderlyingPrice	Public	Can modify state	onlyOwner
setPrice	Public	Can modify state	onlyOwner
getPrice	External	-	-
get	External	-	-
compareStrings	Internal	-	-

WePiggyPriceOracleV1			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
getPrice	External	-	-
setPrice	External	Can modify state	onlyOwner
setTokenConfig	Public	Can modify state	onlyOwner

WePiggyPriceProviderV1			
Function Name	Visibility	Mutability	Modifiers
getUnderlyingPrice	External	-	-
_getUnderlyingPriceInternal	Internal	-	-
_getCustomerPriceInternal	Internal	-	-
_getCompoundPriceInternal	Internal	-	-
_getChainlinkPriceInternal	Internal	-	-
addTokenConfig	Public	Can modify state	onlyOwner
addOrUpdateTokenConfigSource	Public	Can modify state	onlyOwner
updateTokenConfigBaseUnit	Public	Can modify state	onlyOwner
updateTokenConfigFixedUsd	Public	Can modify state	onlyOwner
getOracleSourcePrice	Public	-	-
compareStrings	Internal	-	-
oracleLength	Public	-	-

PiggyDistribution			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
distributeMintWpc	Public	Can modify state	-
distributeRedeemWpc	Public	Can modify state	-
distributeBorrowWpc	Public	Can modify state	-
distributeRepayBorrowWpc	Public	Can modify state	-
distributeSeizeWpc	Public	Can modify state	-
distributeTransferWpc	Public	Can modify state	-
_stakeTokenToPiggyBreeder	Public	Can modify state	onlyOwner
_claimWpcFromPiggyBreeder	Public	Can modify state	onlyOwner
_refreshWpcSpeeds	Public	Can modify state	onlyOwner
refreshWpcSpeedsInternal	Internal	Can modify state	-
updateWpcSupplyIndex	Internal	Can modify state	-
updateWpcBorrowIndex	Internal	Can modify state	-
distributeSupplierWpc	Internal	Can modify state	-
distributeBorrowerWpc	Internal	Can modify state	-
transferWpc	Internal	Can modify state	-
claimWpc	Public	Can modify state	-
claimWpc	Public	Can modify state	-
claimWpc	Public	Can modify state	-
_setWpcRate	Public	Can modify state	onlyOwner
_addWpcMarkets	Public	Can modify state	onlyOwner
_addWpcMarketInternal	Internal	Can modify state	-
_dropWpcMarket	Public	Can modify state	onlyOwner
transferALLWPC	Public	Can modify state	onlyOwner

BaseJumpRateModel			
Function Name	Visibility	Mutability	Modifiers
updateJumpRateModel	External	Can modify state	-
utilizationRate	Public	-	-
getBorrowRateInternal	Internal	-	-
getBorrowRate	External	-	-
getSupplyRateInternal	Internal	-	-
getSupplyRate	External	-	-
updateJumpRateModelInternal	Internal	Can modify state	onlyOwner

DAIInterestRateModel			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
updateDAIJumpRateModel	External	Can modify state	-
getSupplyRate	External	-	-
dsrPerBlock	Public	-	-
poke	Public	Can modify state	-

JumpRateModel			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer

PERC20			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
mint	External	Can modify state	-
mintForMigrate	External	Can modify state	-
redeem	External	Can modify state	-
redeemUnderlying	External	Can modify state	-
borrow	External	Can modify state	-
repayBorrow	External	Can modify state	-
repayBorrowBehalf	External	Can modify state	-
liquidateBorrow	External	Can modify state	-
_addReserves	External	Can modify state	-
getCashPrior	Internal	-	-
doTransferIn	Internal	Can modify state	-
doTransferOut	Internal	Can modify state	-

PEther			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can modify state	initializer
mint	External	Payable	-
mintForMigrate	External	Payable	-
redeem	External	Can modify state	-

redeemUnderlying	External	Can modify state	-
borrow	External	Can modify state	-
repayBorrow	External	Payable	-
repayBorrowBehalf	External	Payable	-
liquidateBorrow	External	Payable	-
	External	Payable	-
getCashPrior	Internal	-	-
doTransferIn	Internal	Can modify state	-
doTransferOut	Internal	Can modify state	-
require-Error	Internal	-	-

PToken			
Function Name	Visibility	Mutability	Modifiers
init	Public	Can modify state	onlyOwner
transferTokens	Internal	Can modify state	-
transfer	External	Can modify state	nonReentrant
transferFrom	External	Can modify state	nonReentrant
approve	External	Can modify state	-
allowance	External	-	-
balanceOf	External	-	-
balanceOfUnderlying	External	Can modify state	-
getAccountSnapshot	External	-	-
getBlockNumber	Internal	-	-
borrowRatePerBlock	External	-	-
supplyRatePerBlock	External	-	-
totalBorrowsCurrent	External	Can modify state	nonReentrant
borrowBalanceCurrent	External	Can modify state	nonReentrant
borrowBalanceStored	Public	-	-
borrowBalanceStoredInternal	Internal	-	-
borrowInterestBalancePriorInternal	Internal	-	-
exchangeRateCurrent	Public	-	-
exchangeRateStored	Public	-	-
exchangeRateStoredInternal	Internal	-	-
getCash	External	-	-
accrueInterestSnapshot	Public	-	-
accrueInterest	Public	Can modify state	-
mintInternal	Internal	Can modify state	nonReentrant

mintInternalForMigrate	Internal	Can modify state	nonReentrant
mintFresh	Internal	Can modify state	-
redeemInternal	Internal	Can modify state	nonReentrant
redeemUnderlyingInternal	Internal	Can modify state	nonReentrant
redeemFresh	Internal	Can modify state	-
borrowInternal	Internal	Can modify state	nonReentrant
borrowFresh	Internal	Can modify state	-
repayBorrowInternal	Internal	Can modify state	nonReentrant
repayBorrowBehalfInternal	Internal	Can modify state	nonReentrant
repayBorrowFresh	Internal	Can modify state	-
liquidateBorrowInternal	Internal	Can modify state	nonReentrant
liquidateBorrowFresh	Internal	Can modify state	-
seize	External	Can modify state	nonReentrant
seizeInternal	Internal	Can modify state	-
_setComptroller	Public	Can modify state	onlyOwner
_setReserveFactor	External	Can modify state	nonReentrant
_setReserveFactorFresh	Internal	Can modify state	onlyOwner
_addReservesInternal	Internal	Can modify state	nonReentrant
_addReservesFresh	Internal	Can modify state	-
_reduceReserves	External	Can modify state	nonReentrant
_reduceReservesFresh	Internal	Can modify state	onlyOwner
_setInterestRateModel	Public	Can modify state	-
_setInterestRateModelFresh	Internal	Can modify state	onlyOwner
_setMigrator	Public	Can modify state	onlyOwner
_setMinInterestAccumulated	Public	Can modify state	onlyOwner
getCashPrior	Internal	-	-
doTransferIn	Internal	Can modify state	-
doTransferOut	Internal	Can modify state	-

4.2 Contract Information

The first phase has audited the following contracts and has been deployed to the main network.

The contract address in main network:

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

The contract for the second phase audit has been deployed to the main network，the deployment on the main network will transfer the contract's owner authority to the timelock contract, The current Owner is a multiSign contract the PToken deployment adopts an upgradeable model, and the admin of the AdminUpgradeabilityProxy contract is managed by the ordinary address of the project party.

The contract address in main network：

Contract Name	Contract Address
AWT1	0x12D803497D1e58dD4D4A4F455D754f1d0F937C8b
WPC_AIRDROP_FUNDING_MANAGER	0x2dd8FFA7923a17739F70C34759Af7650e44EA3BE
WP_PRICE_PROVIDER_V1	0xe212829Ca055eD63279753971672c693C6C6d088
WE_PIGGY_PRICE_ORACLE_V1	0xE4A1E73157EB4b58b1347E2BE2df7ac83467b288
COMPTROLLER	0x0C8c1ab017c3C0c8A48dD9F1DB2F59022D190f0b
STABLECOIN_JUMP_RATE_MODEL	0x8158B34fF8A36dD9E4519d62C52913C24ad5554b
BTC_ETH_JUMP_RATE_MODEL	0xA0a75821220bfC74f8012d5D5745FE472F510075
P_DAI	0x85166b72c87697a6acfF24101B43Fd54fE28a179
P_USDT	0x5cFad792C4Df1323188180778AeC58E00eAcE32a
P_USDC	0xf8E5b9738BF63ADFFf36a849F9b9C9617c8D8c1f
P_WBTC	0xc12B9D620bFCB48be3e0CCbf0ea80C717333b46F
P_ETH	0x27A94869341838D5783368a8503FdA5fbCd7987c
MAX_IMILLION	0x97F3763F8C0bE87Cab0e99Ee4b7806acA772FeDA
C_ETH_MIGRATOR	0xe70ADe95E3D038398eb0ACD17534200b0c87A7c4
C_DAI_MIGRATOR	0x836490DcF3C63da80A8BEE9C666Fbfe294D80c69
C_USDT_MIGRATOR	0x8ac4cdf74ce4B4691591384122f3eFeAae351C25

C_USDC_MIGRATOR	0x271e498c2FCaa9f2dbB4f6b9046ba65Fa2c79dBE
C_WBTC_MIGRATOR	0xab80D132eDd440DD7E80ADF77BE91a7445921334
A_TOKEN_MIGRATOR	0x894c9FB45aeCc4F757C2d07A692504E8e29A5747
PIGGY_DISTRIBUTION	0xfe584fc2b820aef5e4976beb9379ecd22aed0c6c

4.3 Code Audit

4.3.1 High-risk vulnerabilities

4.3.1.1 Missing Permission Verification

The distributeTransferWpc function does not have permission control, allowing any user to distribute rewards by himself. It is recommended to add permission control to distributeTransferWpc.

- contracts/farm/PiggyDistribution.sol

```
function distributeTransferWpc(address pToken, address src, address dst, bool distributeAll) public override(IPiggyDistribution) {
        updateWpcSupplyIndex(pToken);
        distributeSupplierWpc(pToken, src, distributeAll);
        distributeSupplierWpc(pToken, dst, distributeAll);
    }
```

Fix Status: The issues has been fixed in this commit:

192d06294bb48b2c8fc555b3dff010d4136d197e

4.3.1.2 Race Conditions Issues

The setTargetToken function can set the address of the targetToken contract to be migrated. There is a race condition issues. When the migration function is called normally, the attacker uses a higher gas price to set the targetToken as a malicious contract. After the migration is completed, the underlying assets can be set take away from the malicious contract.

- contracts/farm/migrator/CErc20Migrator.sol

```
function setTargetToken(address _targetToken) public {
        targetToken = _targetToken;
    }
```

Fix Status: The issues has been fixed in this commit:

ee167fa88735065350b9880a5402ede923ae7b8d

4.3.2 Medium-risk vulnerabilities

4.3.2.1 Business Logic Issues

The ATokenMigrator contract receives Ether through receive. If eth is entered into the ATokenMigrator contract but the replaceMigrate function has not been called, In this case, when the replaceMigrate function is called, the underlying assets and the eth of the malicious entry into the contract are migrated, but the lp used when the mintForMigrate is finally called is obtained through oldLpToken.balanceOf(breeder); without recalculation , This will use more than expected eth for migration, but lp still uses the old value, this will cause the final migration data out of expectations.

- contracts/farm/migrator/ATokenMigrator.sol

```
function replaceMigrate(ATokenInterface oldLpToken) external payable returns (PToken, uint){
        require(msg.sender == breeder, "not from breeder");
        require(block.number >= notBeforeBlock, "too early to migrate");
        address self = address(this);
        uint256 lp = oldLpToken.balanceOf(breeder);
        require(lp > 0, "balance must bigger than 0");
        oldLpToken.transferFrom(breeder, self, lp);
        oldLpToken.redeem(lp);
        address underlyingAssetAddress = oldLpToken.underlyingAssetAddress();
        if (underlyingAssetAddress == address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE)) {
            PEther newLpToken = PEther(targetToken);
            // 获得赎回了多少代币
            uint redeemBal = self.balance;
```

```
            // 将赎回的代币，抵押到 wePiggy 中，生成 pToken
            newLpToken.mintForMigrate{value : redeemBal}(lp);
            // 获得抵押生成的 pToken 有多少
            uint mintBal = newLpToken.balanceOf(self);
            //将余额转到挖矿合约
            newLpToken.transferFrom(self, breeder, mintBal);
            //返回占比
            return (newLpToken, mintBal);
        } else {
            PERC20 newLpToken = PERC20(targetToken);
            require(underlyingAssetAddress == newLpToken.underlying(), "not match");
            //获得赎回了多少代币
            uint redeemBal = 0;
            IERC20 token = IERC20(underlyingAssetAddress);
            redeemBal = token.balanceOf(self);
            // 将赎回的代币，抵押到 wePiggy 中，生成 pToken
            token.approve(address(newLpToken), redeemBal);
            newLpToken.mintForMigrate(redeemBal, lp);
            // 获得抵押生成的 pToken 有多少
            uint mintBal = newLpToken.balanceOf(self);
            //将余额转到挖矿合约
            newLpToken.transferFrom(self, breeder, mintBal);
            return (newLpToken, mintBal);
        }
    }
```

Fix Status: The issues has been Incomplete fixed in this commit:

ee167fa88735065350b9880a5402ede923ae7b8d

4.3.2.2 Floating Points and Numerical Precision

Divide first, then subtract, and finally multiply. Because of the feature of "solidity", decimals are not retained, resulting in a large deviation in arithmetic accuracy.

- contracts/token/PToken.sol

```
if (borrowSnapshot.interestIndex == 0) {
            return (MathError.NO_ERROR, 0);
        }
```

```
(mathErr, interestTimesIndex) = divUInt(borrowIndex, borrowSnapshot.interestIndex);
if (mathErr != MathError.NO_ERROR) {
            return (mathErr, 0);
        }
(mathErr, principalTimesIndex) = subUInt(interestTimesIndex, 1);
if (mathErr != MathError.NO_ERROR) {
            return (mathErr, 0);
        }
(mathErr, interestAmountPrior) = mulUInt(principalTimesIndex, borrowSnapshot.principal);
if (mathErr != MathError.NO_ERROR) {
            return (mathErr, 0);
        }
```

Fix Status: The issues has been Incomplete fixed in this commit:

222159f4dbcf54ac254185fbadd00a52ab46a2ae

4.3.3 Low-risk vulnerabilities

4.3.3.1 Excessive authority auditing

The method of feeding prices uses chainlink's Oracle, compound's Oracle, and a centralized method. The token configuration determines which method to use to feed the price. The Owner can configure the token's feed price method, The Owner in the SimplePriceOracle contract can set the price arbitrarily, and there is a risk of excessive authority.

- contracts/oracle/SimplePriceOracle.sol

```
    function setUnderlyingPrice(PToken pToken, uint price) public onlyOwner {
        address asset = _pETHUnderlying;
        if (!compareStrings(pToken.symbol(), "pETH")) {
            asset = address(PERC20(address(pToken)).underlying());
        }
        uint bt = block.timestamp;
        data[asset] = Datum(bt, price);
        emit PricePosted(asset, data[asset].price, price, price, bt);
    }
```

```
    function setPrice(address asset, uint price) public onlyOwner {
        uint bt = block.timestamp;
        emit PricePosted(asset, data[asset].price, price, price, bt);
        data[asset] = Datum(bt, price);
    }
```

Fix Status: The implementation of the code has been modified in this commit:

03b8b4d744e53436c6c78b25384f1d2a257b1cc8

The Owner of WePiggyPriceOracleV1 can arbitrarily set the token price and set the token configuration, there is a risk of excessive authority.

- contracts/oracle/WePiggyPriceOracleV1.sol

```
function setPrice(address token, uint price, bool force) external override(WePiggyPriceOracleInterface) onlyOwner {
        Datum storage datum = data[token];
        if (force) {
            datum.value = price;
            datum.timestamp = block.timestamp;
        } else {
            TokenConfig storage config = configs[token];
            require(config.token == token, "bad params");
            uint upper = datum.value.mul(config.upperBoundAnchorRatio).div(1e2);
            uint lower = datum.value.mul(config.lowerBoundAnchorRatio).div(1e2);
            require(price.sub(lower) >= 0, "the price must greater than the old*lowerBoundAnchorRatio");
            require(upper.sub(price) >= 0, "the price must less than the old*upperBoundAnchorRatio");
            datum.value = price;
            datum.timestamp = block.timestamp;
        }
        emit PriceUpdated(token, price);
    }
function setTokenConfig(address token, string memory symbol, uint upperBoundAnchorRatio, uint lowerBoundAnchorRatio)
public onlyOwner {
        require(minLowerBoundAnchorRatio <= lowerBoundAnchorRatio, "lowerBoundAnchorRatio must greater or equal to
minLowerBoundAnchorRatio");
        require(maxUpperBoundAnchorRatio >= upperBoundAnchorRatio, "upperBoundAnchorRatio must Less than or equal to
maxUpperBoundAnchorRatio");
        TokenConfig storage config = configs[token];
        config.token = token;
        config.symbol = symbol;
        config.upperBoundAnchorRatio = upperBoundAnchorRatio;
```

```
        config.lowerBoundAnchorRatio = lowerBoundAnchorRatio;
        emit ConfigUpdated(token, symbol, upperBoundAnchorRatio, lowerBoundAnchorRatio);
    }
```

Fix Status: The implementation of the code has been modified in commit:

72a028a8ea34765d73eef654ab3f48a1c575191b

The Owner of WePiggyPriceProviderV1 can add and update the token configuration, there is a risk of excessive authority.

- contracts/oracle/WePiggyPriceProviderV1.sol

```
function addTokenConfig(address pToken, address underlying, string memory underlyingSymbol, uint256 baseUnit, bool fixedUsd,
        address[] memory sources, PriceOracleType[] calldata sourceTypes) public onlyOwner {
        require(sources.length == sourceTypes.length, "sourceTypes.length must equal than sources.length");
        // add TokenConfig
        TokenConfig storage tokenConfig = tokenConfigs[pToken];
        require(tokenConfig.pToken == address(0), "bad params");
        tokenConfig.pToken = pToken;
        tokenConfig.underlying = underlying;
        tokenConfig.underlyingSymbol = underlyingSymbol;
        tokenConfig.baseUnit = baseUnit;
        tokenConfig.fixedUsd = fixedUsd;
        // add priceOracles
        require(oracles[pToken].length < 1, "bad params");
        for (uint i = 0; i < sources.length; i++) {
            PriceOracle[] storage list = oracles[pToken];
            list.push(PriceOracle({
            source : sources[i],
            sourceType : sourceTypes[i]
            }));
        }
        emit ConfigUpdated(pToken, underlying, underlyingSymbol, baseUnit, fixedUsd);
        emit PriceOracleUpdated(pToken, oracles[pToken]);
    }
function addOrUpdateTokenConfigSource(address pToken, uint256 index, address source, PriceOracleType _sourceType) public onlyOwner {
        PriceOracle[] storage list = oracles[pToken];
        if (list.length > index) {//will update
            PriceOracle storage oracle = list[index];
```

```
                oracle.source = source;
                oracle.sourceType = _sourceType;
            } else {//will add
                list.push(PriceOracle({
                source : source,
                sourceType : _sourceType
                }));
            }
        }
    function updateTokenConfigBaseUnit(address pToken, uint256 baseUnit) public onlyOwner {
            TokenConfig storage tokenConfig = tokenConfigs[pToken];
            require(tokenConfig.pToken != address(0), "bad params");
            tokenConfig.baseUnit = baseUnit;
            emit ConfigUpdated(pToken, tokenConfig.underlying, tokenConfig.underlyingSymbol, baseUnit, tokenConfig.fixedUsd);
        }
    function updateTokenConfigFixedUsd(address pToken, bool fixedUsd) public onlyOwner {
            TokenConfig storage tokenConfig = tokenConfigs[pToken];
            require(tokenConfig.pToken != address(0), "bad params");
            tokenConfig.fixedUsd = fixedUsd;
            emit ConfigUpdated(pToken, tokenConfig.underlying, tokenConfig.underlyingSymbol, tokenConfig.baseUnit, fixedUsd);
        }
```

The owner of PiggyDistribution, PToken and Comptroller contracts is not set to the timelock contract. It is recommended to transfer the owner authority of PiggyDistribution, PToken and Comptroller to the timelock contract.

Fix Status: Waiting for fix.

4.3.4 Enhancement Suggestions

4.3.4.1 Enhancement Point of DelegateBySig Function

The nonce in the delegateBySig function is input by the user. When the user input a larger nonce, the current transaction cannot be success but the relevant signature data will still remain on the chain, causing this signature to be available for some time in the future. It is recommended to fix it

according to EIP-2612.

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

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

```
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);
```

Fix Status: This issues has been ignore.

5. Audit Result

5.1 Conclusion

Audit Result : Low Risk

Audit Number : 0X002012140003

Audit Date : Dec. 14, 2020

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, 2 high-risk, 2 medium-risk, 1 low-risk vulnerabilities were found during the audit, the high-risk, medium-risk have been fixed, the owner authority has not been transferred to the timelock contract. There is an enhancement suggestion has been ignore.

6. Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance this report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist