Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.09.16, the SlowMist security team received the Evrynet team's security audit application for Evrynet Phase 3, 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.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules 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.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using 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 Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit version:

https://github.com/Evry-Finance/evry-finance-amm-swap

commit: 35ce66036b7f5c6a2f54d5266a00e2591394979c (master branch)

https://github.com/Evry-Finance/evry-finance-dmm-swap

commit: 144843b7db62e6fc1cb764ade4ab02af08c8450d (master branch)

https://github.com/Evry-Finance/evry-finance-farm

commit: 2d194cdbce2621ce08c863579bce8a51bc7bafcd (master branch)

https://github.com/Evry-Finance/evry-finance-toolkit/blob/master/contracts/Timelock.sol

commit: 4120ed4afaf2ba01b222f469498922277d47cd73 (master branch)

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Missing event records	Others	Suggestion	Fixed
N2	Earnings update issue	Design Logic Audit	Low	Fixed
N3	Compatibility issue	Design Logic Audit	Suggestion	Fixed
N4	Risk of excessive authority	Authority Control Vulnerability	Medium	Fixed
N5	Redundant code	Others	Suggestion	Fixed
N6	Platform fee issue	Design Logic Audit	High	Fixed
N7	Proxy model issue	Others	Suggestion	Confirmed
N8	Minimum delay time issue	Design Logic Audit	Low	Confirmed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

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

Farms

Farms			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
setEvryPerBlock	External	Can Modify State	onlyOwner
addPool	External	Can Modify State	onlyOwner
setPoolAllocation	External	Can Modify State	onlyOwner
deposit	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	nonReentrant
harvestAll	External	Can Modify State	-
harvest	External	Can Modify State	-
poolLength	External	-	-
pendingReward	External	-	-
updatePool	Public	Can Modify State	-
_harvest	Internal	Can Modify State	-
isDuplicatedPool	Internal	-	-
isEvryPool	Internal	-	-

EVRYDistributor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
release	External	Can Modify State	nonReentrant onlyOwner

Timelock			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
pendingAdminConfirm	Public	Can Modify State	-
setPendingAdmin	Public	Can Modify State	-
queueTransaction	Public	Can Modify State	-
cancelTransaction	Public	Can Modify State	-
executeTransaction	Public	Payable	-
getPendingTransactions	External	-	-
getBlockTimestamp	Public	-	-
_getRevertMsg	Internal	-	-

EvryERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_approve	Private	Can Modify State	-
_transfer	Private	Can Modify State	-
approve	External	Can Modify State	-

EvryERC20			
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
permit	External	Can Modify State	-

EvryFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allPairsLength	External	-	-
createPair	External	Can Modify State	onlyAdmin
setFeeToPlatform	External	Can Modify State	onlyOwner
setPlatformFee	External	Can Modify State	onlyOwner
setLiquidityFee	External	Can Modify State	onlyOwner
transferAdmin	External	Can Modify State	onlyAdmin
getFeeConfiguration	External	-	-

EvryPair			
Function Name	Visibility	Mutability	Modifiers
getReserves	Public	-	-
_safeTransfer	Private	Can Modify State	-
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-

EvryPair			
_update	Private	Can Modify State	-
mint	External	Can Modify State	lock
burn	External	Can Modify State	lock
swap	External	Can Modify State	lock
sendFeeToPlatform	Private	Can Modify State	-
getBasisTotalFee	Internal	-	-
_sync	Private	Can Modify State	-
skim	External	Can Modify State	lock
sync	External	Can Modify State	lock

EvryRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
_addLiquidity	Internal	Can Modify State	-
addLiquidity	External	Can Modify State	ensure
addLiquidityETH	External	Payable	ensure
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityETH	Public	Can Modify State	ensure

EvryRouter			
removeLiquidityWithPermit	External	Can Modify State	-
removeLiquidityETHWithPermit	External	Can Modify State	-
removeLiquidityETHSupportingFeeOnTransferTokens	Public	Can Modify State	ensure
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	Can Modify State	-
_swap	Internal	Can Modify State	-
swapExactTokensForTokens	External	Can Modify State	ensure
swapTokensForExactTokens	External	Can Modify State	ensure
swapExactETHForTokens	External	Payable	ensure
swapTokensForExactETH	External	Can Modify State	ensure
swapExactTokensForETH	External	Can Modify State	ensure
swapETHForExactTokens	External	Payable	ensure
_swapSupportingFeeOnTransferTokens	Internal	Can Modify State	-
swapExactTokensForTokensSupportingFeeOnTransferTokens	External	Can Modify State	ensure
swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	ensure
swapExactTokensForETHSupportingFeeOnTransferTokens	External	Can Modify State	ensure
quote	Public	-	-
getAmountOut	Public	-	-

EvryRouter

getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

DaoRegistry

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
addPool	External	Can Modify State	onlyOwner
getPools	External	-	-

DMMRouter02

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ManageUser
<Receive Ether>	External	Payable	-
_setImplementation	Private	Can Modify State	-
_implementation	Internal	-	-
_delegate	Internal	Can Modify State	-
<Fallback>	External	Payable	-
_swap	Private	Can Modify State	-
swapExactTokensForTokens	Public	Can Modify State	ensure

DMMRouter02			
swapTokensForExactTokens	Public	Can Modify State	ensure
swapExactETHForTokens	External	Payable	ensure
swapTokensForExactETH	External	Can Modify State	ensure
swapExactTokensForETH	External	Can Modify State	ensure
swapETHForExactTokens	External	Payable	ensure
_swapSupportingFeeOnTransferTokens	Internal	Can Modify State	-
swapExactTokensForTokensSupportingFeeOnTransferTokens	Public	Can Modify State	ensure
swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	ensure
swapExactTokensForETHSupportingFeeOnTransferTokens	External	Can Modify State	ensure
quote	External	-	-
getAmountsOut	External	-	-
getAmountsIn	External	-	-
verifyPoolsPathSwap	Internal	-	-
verifyPoolAddress	Internal	-	-

DMMRouter02DelegateCall			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ManageUser
<Receive Ether>	External	Payable	-

DMMRouter02DelegateCall			
_addLiquidity	Internal	-	-
addLiquidity	Public	Can Modify State	ensure
addLiquidityNewPool	External	Can Modify State	_admin
addLiquidityETH	Public	Payable	ensure
addLiquidityNewPoolETH	External	Payable	_superAdmin
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityETH	Public	Can Modify State	ensure
removeLiquidityWithPermit	External	Can Modify State	-
removeLiquidityETHWithPermit	External	Can Modify State	-
removeLiquidityETHSupportingFeeOnTransferTokens	Public	Can Modify State	ensure
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	Can Modify State	-
verifyPoolAddress	Internal	-	-

DMMFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ManageUser
_setImplementation	Private	Can Modify State	-
_implementation	Internal	-	-
_delegate	Internal	Can Modify State	-

DMMFactory			
<Fallback>	External	Payable	-
createPool	External	Can Modify State	_admin
isPool	External	-	-
getFeeConfiguration	External	-	-

DMMFactoryDelegate			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ManageUser
setFeeConfiguration	External	Can Modify State	-
setFeeSetter	External	Can Modify State	-
allPoolsLength	External	-	-
getPools	External	-	-
getPoolsLength	External	-	-
getPoolAtIndex	External	-	-

DMMPool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20Permit VolumeTrendRecorder
initialize	External	Can Modify State	-
mint	External	Can Modify State	nonReentrant

DMMPool			
burn	External	Can Modify State	nonReentrant
swap	External	Can Modify State	nonReentrant
skim	External	Can Modify State	nonReentrant
sync	External	Can Modify State	nonReentrant
_sync	Private	Can Modify State	-
getTradeInfo	External	-	-
getReserves	External	-	-
name	Public	-	-
symbol	Public	-	-
verifyBalanceAndUpdateEma	Internal	Can Modify State	-
getFeeBeforeSwap	Public	-	-
_update	Internal	Can Modify State	-
_mintFee	Internal	Can Modify State	-
getReservesData	Internal	-	-
getK	Internal	-	-
safeUint112	Internal	-	-
feeForPlatform	Private	-	-
sendFeeToPlatform	Private	Can Modify State	-

ManageUser			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getRoleUser	Public	-	-

Farms			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
setEvryPerBlock	External	Can Modify State	onlyOwner
addPool	External	Can Modify State	onlyOwner
setPoolAllocation	External	Can Modify State	onlyOwner
deposit	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	nonReentrant
harvestAll	External	Can Modify State	-
harvest	External	Can Modify State	-
poolLength	External	-	-
pendingReward	External	-	-
updatePool	Public	Can Modify State	-
_harvest	Internal	Can Modify State	-
isDuplicatedPool	Internal	-	-
isEvryPool	Internal	-	-

ManageUserAddress			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getRole	Public	-	-
addAdmin	Public	Can Modify State	-
removeAdmin	Public	Can Modify State	-
transferSuperAdmin	Public	Can Modify State	-

VolumeTrendRecorder			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getVolumeTrendData	External	-	-
recordNewUpdatedVolume	Internal	Can Modify State	-
calculateRFactorByNewVolume	Internal	-	-
getRFactor	Internal	-	-
calculateRFactor	Internal	-	-
newEMA	Internal	-	-
safeUint128	Internal	-	-
safeUint128	Internal	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Missing event records

Category: Others

Content

- In the Farms contract, the owner can modify the evryPerBlock parameter through the setEvryPerBlock function, but the event is not recorded.
- In the EvryFactory contract, the owner role can modify the feeToPlatform, feePlatformBasis, and feeLiquidityBasis parameters through the setFeeToPlatform, setPlatformFee, and setLiquidityFee functions, respectively. The admin role can transfer ownership through the transferAdmin function. But none of the incidents were recorded.

Code location:

- farm/contracts/Farms.sol

```
function setEvryPerBlock(uint256 _evryPerBlock) external onlyOwner {
  evryPerBlock = _evryPerBlock;
}
```

- amm/contracts/EvryFactory.sol

```
  function setFeeToPlatform(address _feeToPlatform) external onlyOwner override {
      feeToPlatform = _feeToPlatform;
  }

  function setPlatformFee(uint256 feeBasis) external onlyOwner override {
      feePlatformBasis = feeBasis;
  }

  function setLiquidityFee(uint256 feeBasis) external onlyOwner override {
      feeLiquidityBasis = feeBasis;
  }

  function transferAdmin(address newAdmin) external onlyAdmin override {
      admin = newAdmin;
  }
```

Solution

It is recommended to record incidents when modifying sensitive parameters for follow-up self-examination or community review.

Status

Fixed

[N2] [Low] Earnings update issue

Category: Design Logic Audit

Content

In the farm contract, the owner can add a new pool through the addPool function, and modify the allocation point of the pool through the setPoolAllocation function. However, when the addPool and setPoolAllocation operations are performed, all the existing pools in the contract are not updated first, which will cause the revenue of the existing pools to change due to the change in the allocation points.

Code location: farm/contracts/Farms.sol

```
  function addPool(
    uint256 allocPoint,
    IERC20 _stakeToken,
    uint256 _startBlock
  ) external onlyOwner {
    require(!isDuplicatedPool(_stakeToken), "Farms::addPool:: stakeToken dup");
    uint256 lastRewardBlock = block.number > _startBlock ? block.number :
_startBlock;
    totalAllocPoint = totalAllocPoint.add(allocPoint);

    stakeTokens.push(_stakeToken);

    poolInfo.push(
      PoolInfo({ lpToken: _stakeToken, allocPoint: allocPoint, lastRewardBlock:
lastRewardBlock, accEVRYPerShare: 0 })
    );
    emit AddPool(stakeTokens.length.sub(1), allocPoint, _stakeToken);
  }
  function setPoolAllocation(uint256 _pid, uint256 _allocPoint) external onlyOwner {
    updatePool(_pid);

    // Remove current AP value of pool _pid from total AP, then add new one.
    totalAllocPoint =
totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);

    // Replace old AP value with new one.
    poolInfo[_pid].allocPoint = _allocPoint;

    emit SetPoolAllocation(_pid, _allocPoint);
  }
```

Solution

It is recommended to update all existing pools first when adding new pools or modifying pool allocation points.

Status

Fixed

[N3] [Suggestion] Compatibility issue

Category: Design Logic Audit

Content

In the farm contract, users can stake their tokens through the deposit function. It will directly record the `amount` parameter passed by the user into `user.amount`, and transfer the tokens to the contract through the safeTransferFrom function. If the contract receives deflationary tokens, the actual number of tokens received by the contract will not match the number of tokens recorded in the contract.

Code location: farm/contracts/Farms.sol

```
  function deposit(
    address _for,
    uint256 pid,
    uint256 amount
  ) external nonReentrant {
    PoolInfo memory pool = updatePool(pid);
    UserInfo storage user = userInfo[pid][_for];
    // Validation
    if (user.fundedBy != address(0)) require(user.fundedBy == msg.sender,
"Farms::deposit:: bad sof");

    // Effects
    _harvest(_for, pid);

    user.amount = user.amount.add(amount);
    user.rewardDebt = user.rewardDebt.add(amount.mul(pool.accEVRYPerShare) /
ACC_EVRY_PRECISION);
    if (user.fundedBy == address(0)) user.fundedBy = msg.sender;

    // Interactions
    stakeTokens[pid].safeTransferFrom(msg.sender, address(this), amount);
    if (isEvryPool(pool.lpToken)) evrySupply = evrySupply.add(amount);

    emit Deposit(msg.sender, pid, amount, _for);
  }
```

Solution

It is recommended to use the difference between the contract balance before and after the transfer to record the user's actual recharge amount.

Status

Fixed

[N4] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

In the EvryFactory contract, the owner can arbitrarily set the value of the feePlatformBasis and feeLiquidityBasis parameters. These two parameters will determine the amount of handling fees that users need to pay during the swap process. However, the value range of these two parameters is not restricted when setting these two parameters, which will lead to the risk of excessive owner authority.

Code location: amm/contracts/EvryFactory.sol

```
function setPlatformFee(uint256 feeBasis) external onlyOwner override {
    feePlatformBasis = feeBasis;
}

function setLiquidityFee(uint256 feeBasis) external onlyOwner override {
    feeLiquidityBasis = feeBasis;
}
```

Solution

It is recommended to limit the value range of feePlatformBasis and feeLiquidityBasis.

Status

Fixed

[N5] [Suggestion] Redundant code

Category: Others

Content

- The TWAP module was removed from the EvryPair contract, but the price0CumulativeLast and price1CumulativeLast parameters were not removed.
- In the VolumeTrendRecorder contract, the calculateRFactorByNewVolume function will calculate the rFactor value based on currentBlockVolume. When skipBlock is 0, it will directly return the calculation result of calculateRFactor without changing the currentBlockVolume value, so `uint256(currentBlockVolume).add(value)` is redundant.

Code location:

- amm/contracts/EvryPair.sol

```
uint public override price0CumulativeLast;
uint public override price1CumulativeLast;
```

- dmm/contracts/VolumeTrendRecorder.sol

```
function calculateRFactorByNewVolume(uint256 blockNumber, uint256 value)
    internal
    view
    returns (uint256) {

    uint256 skipBlock = blockNumber - lastTradeBlock;
    if (skipBlock == 0) {
        safeUint128(
            uint256(currentBlockVolume).add(value),
            "volume exceeds valid range"
        );
        return calculateRFactor(uint256(shortEMA), uint256(longEMA));
    }

    uint256 _shortEMA = newEMA(shortEMA, SHORT_ALPHA, currentBlockVolume);
    uint256 _longEMA = newEMA(longEMA, LONG_ALPHA, currentBlockVolume);
    // ema = ema * (1-aplha) ^(skipBlock -1)
    _shortEMA = _shortEMA.mulInPrecision(
        (PRECISION - SHORT_ALPHA).unsafePowInPrecision(skipBlock - 1)
    );
    _longEMA = _longEMA.mulInPrecision(
        (PRECISION - LONG_ALPHA).unsafePowInPrecision(skipBlock - 1)
    );
    return calculateRFactor(_shortEMA, _longEMA);

}
```

Solution

It is recommended to remove redundant code.

Status

Fixed

[N6] [High] Platform fee issue

Category: Design Logic Audit

Content

In the EvryPair contract, users can directly exchange tokens through the swap function, and finally send a fee to the feeToPlatform address through the sendFeeToPlatform function. However, the number of handling fees and feeToPlatform address can be directly passed in by the user, so users can pass in a handling fee of 0 to waive the handling fee that needs to be paid when the token is exchanged.

Similarly, the _swap and _swapSupportingFeeOnTransferTokens of the Router contract will not need to pass feeToPlatform, feePlatformBasis, and feeLiquidityBasis parameters.

Code location:

```
    function swap(
            uint[2] memory amountOut,
            address to,
            address feeToPlatform,
            uint feePlatformBasis,
            uint feeLiquidityBasis,
            bytes calldata data
        )
            external
            lock
            override
        {

        FeeConfiguration memory feeConfiguration = FeeConfiguration({
            feeToPlatform: feeToPlatform,
            feePlatformBasis: feePlatformBasis,
            feeLiquidityBasis: feeLiquidityBasis,
            amount0Out: amountOut[0],
            amount1Out: amountOut[1]
        });

        require(feeConfiguration.amount0Out > 0 || feeConfiguration.amount1Out > 0,
'Evry: INSUFFICIENT_OUTPUT_AMOUNT');

        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(feeConfiguration.amount0Out < _reserve0 &&
feeConfiguration.amount1Out < _reserve1, 'Evry: INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'Evry: INVALID_TO');
            if (feeConfiguration.amount0Out > 0) _safeTransfer(_token0, to,
feeConfiguration.amount0Out); // optimistically transfer tokens
            if (feeConfiguration.amount1Out > 0) _safeTransfer(_token1, to,
feeConfiguration.amount1Out); // optimistically transfer tokens
            if (data.length > 0) IEvryCallee(to).evryCall(msg.sender,
feeConfiguration.amount0Out, feeConfiguration.amount1Out, data);
            balance0 = IERC20(_token0).balanceOf(address(this));
            balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - feeConfiguration.amount0Out ?
balance0 - (_reserve0 - feeConfiguration.amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - feeConfiguration.amount1Out ?
balance1 - (_reserve1 - feeConfiguration.amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'Evry: INSUFFICIENT_INPUT_AMOUNT');

        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
            uint totalFee =
feeConfiguration.feePlatformBasis.add(feeConfiguration.feeLiquidityBasis);
            uint balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(totalFee));
            uint balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(totalFee));
            require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(10000**2), 'Evry: K');
        }

        _update(balance0, balance1);
        {
        // emit Swap(msg.sender, amount0In, amount1In, amountOut, to);
}
        if (amount0In > 0) {
            sendFeeToPlatform(token0, amount0In, feeConfiguration.feePlatformBasis,
feeConfiguration.feeToPlatform);
        } else {
            sendFeeToPlatform(token1, amount1In, feeConfiguration.feePlatformBasis,
feeConfiguration.feeToPlatform);
        }
        _sync();

    }
```

Solution

It is recommended to directly obtain the required parameters through the getFeeConfiguration function of the EvryFactory contract.

Status

Fixed

[N7] [Suggestion] Proxy model issue

Category: Others

Content

In the DMM module, DMMFactory is the proxy contract of the factory contract, and DMMFactoryDelegate is the implementation contract of the factory contract. However, part of the logic of the implementation contract is implemented in the agency contract, which causes confusion at the agency level. If the data structure of the implementation contract is modified in the future, the overall security will be affected.

The same is true for the DMMRouter02 contract.

Code location:

dmm/contracts/periphery/DMMRouter02.sol

dmm/contracts/periphery/DMMRouter02DelegateCall.sol

dmm/contracts/DMMFactory.sol

dmm/contracts/DMMFactoryDelegate.sol

Solution

It is recommended that the proxy contract only focus on the agent and other logic is placed in the implementation contract.

Status

Confirmed; After communicating with the project party, the project party stated that due to contract size limitation, we have to keep some functions in proxy.

[N8] [Low] Minimum delay time issue

Category: Design Logic Audit

Content

There is a minimumDelay parameter in the Timelock contract, which is used for the time interval between transaction execution and pending state. The minimumDelay will be passed in when the contract is initialized, but it does not check whether the minimum time interval is reasonable. If 0 or a very small number is passed in, the delay in executing the transaction will become meaningless.

Code location: toolkit/contracts/Timelock.sol

```
  constructor(address admin_, uint256 minimumDelay_, uint256 maximumDelay_) {
    require(minimumDelay_ <= maximumDelay_, "Timelock minimum delay must less than
maximum delay");

    admin = admin_;
    minimumDelay = minimumDelay_;
    maximumDelay = maximumDelay_;
  }
```

Solution

It is recommended to check that the minimumDelay parameter is greater than a reasonable delay time.

Status

Confirmed; After communicating with the project party, the project party stated that product team agreed to keep it as is due to some specific business case. Product team will keep issue in mind and review delay time of every change.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002109300003	SlowMist Security Team	2021.09.16 - 2021.09.30	Low Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 1 medium risk, 2 low risks, 4 suggestion vulnerabilities. And 1 high risk, 1 medium risk, 2 low risks, 4 suggestion vulnerabilities were confirmed and being fixed; All other findings were fixed. The code was not deployed to the mainnet.

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 based 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 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.

SLOWMIST

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist