Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.10.22, the SlowMist security team received the HOTCROSS team's security audit application for HOTCROSS, 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

Project address:

https://github.com/hotcrosscom/cross-yield

commit: 657ff774cf7cb0cd4c2c9b20558c6af50c863cb4

https://github.com/hotcrosscom/initial-hotcross-offering

commit: 7b9f182604fd6cc6246c31043a6e45d46af93aa7

https://github.com/hotcrosscom/cross-send

commit: fa5852811321b7647029fcf9ed6415137445ef7f

https://github.com/hotcrosscom/hotdrop

commit: b615770da4b5a2f6b3612a4096352e0bbb8e3463

Fixed Version

https://github.com/hotcrosscom/cross-send/tree/87d0bf21a91079d9e21591fea7c0aff332d9929d

https://github.com/hotcrosscom/hotdrop/tree/f9157551c01b34e310412f6d5d3568af00421809

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Redundant code	Gas Optimization Audit	Suggestion	Ignored
N2	Gas optimization	Gas Optimization Audit	Suggestion	Ignored
N3	Risk of allowance amount abuse	Authority Control Vulnerability	Low	Fixed
N4	Price manipulation issue	Others	Medium	Ignored
N5	Sandwich attacks issue	Reordering Vulnerability	Medium	Ignored
N6	Missing slippage check	Reordering Vulnerability	Low	Ignored
N7	Permission check Missing	Authority Control Vulnerability	Low	Ignored
N8	Excessive authority issue	Authority Control Vulnerability	Medium	Confirmed
N9	Repeatable claims issue	Design Logic Audit	Critical	Fixed
N10	Round plan security reminder	Others	Suggestion	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:

FeeManagerBase			
Function Name	Visibility	Mutability	Modifiers
__FeeManagerBase_init	Public	Can Modify State	initializer
updateAdmin	Public	Can Modify State	onlyOwner
_updateAdmin	Private	Can Modify State	-
updateFees	Public	Can Modify State	onlyOwnerOrAdmin
_updateFees	Private	Can Modify State	-
updateFeeCollector	Public	Can Modify State	onlyOwner
_updateFeeCollector	Private	Can Modify State	-
readState	Public	-	-

RecipientCountFee			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
distributeNativeFee	Public	Payable	-

RecipientCountFee			
distributeTokenFee	Public	Can Modify State	-

Guard			
Function Name	Visibility	Mutability	Modifiers
__Guard_init	Internal	Can Modify State	initializer
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner

CrossSend			
Function Name	Visibility	Mutability	Modifiers
<Fallback>	External	Payable	-
<Receive Ether>	External	Payable	-
initialize	Public	Can Modify State	initializer
updateFeeManager	Public	Can Modify State	onlyOwner
_updateFeeManager	Private	Can Modify State	-
updateMaxRecipients	Public	Can Modify State	onlyOwner
_updateMaxRecipients	Private	Can Modify State	-
isNative	Private	-	-
send	External	Payable	whenNotPaused nonReentrant
sendToken	Private	Can Modify State	-
sendNative	Private	Can Modify State	-

CrossSend			
rescueToken	External	Can Modify State	onlyOwner

StrategyCake			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
balanceOf	Public	-	-
balanceOfFarmingToken	Public	-	-
stakedInFarmingProtocol	Public	-	-
setFeeProcessor	Public	Can Modify State	onlyOwner
harvest	Public	Can Modify State	whenNotPaused
getPendingFees	Public	-	-
collectFees	Internal	Can Modify State	-
beforeDeposit	External	Can Modify State	-
deposit	Public	Can Modify State	whenNotPaused
withdraw	Public	Can Modify State	-
emergency	Public	Can Modify State	onlyOwner
retireStrategy	External	Can Modify State	-
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner
setAllowances	Internal	Can Modify State	-

StrategyCake			
removeAllowances	Internal	Can Modify State	-

StrategyCakeLp			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
balanceOf	Public	-	-
balanceOfFarmingToken	Public	-	-
stakedInFarmingProtocol	Public	-	-
setFeeProcessor	Public	Can Modify State	onlyOwner
harvest	External	Can Modify State	whenNotPaused
getPendingFees	Public	-	-
collectFees	Internal	Can Modify State	-
addLiquidity	Internal	Can Modify State	-
deposit	Public	Can Modify State	whenNotPaused
withdraw	Public	Can Modify State	-
emergency	Public	Can Modify State	onlyOwner
retireStrategy	External	Can Modify State	-
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner
setAllowances	Internal	Can Modify State	-

StrategyCakeLp			
removeAllowances	Internal	Can Modify State	-

GasPrice			
Function Name	Visibility	Mutability	Modifiers
setMaxGasPrice	External	Can Modify State	onlyOwner

RewardVault			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
safeRewardTransfer	Public	Can Modify State	onlyOwner

CrossStake			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
getCurrentReward	Private	-	-
getRewardForThisPeriod	Private	-	-
getAccRewardPerShare	Private	-	-
pendingRewards	External	-	-
updatePool	Private	Can Modify State	-
releasePending	Private	Can Modify State	-
deposit	External	Can Modify State	nonReentrant

CrossStake			
withdraw	External	Can Modify State	nonReentrant
emergencyWithdraw	Public	Can Modify State	nonReentrant

CrossYield			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
farmingToken	Public	-	-
totalFarmingTokenBalance	Public	-	-
farmingTokenBalance	Public	-	-
farmingTokenPutToWork	Public	-	-
putFundsToWork	Public	Can Modify State	-
IOUPrice	Public	-	-
deposit	Public	Can Modify State	whenNotPaused nonReentrant
depositAll	External	Can Modify State	-
withdraw	Public	Can Modify State	nonReentrant
withdrawAll	External	Can Modify State	-
emergencyRescue	External	Can Modify State	onlyOwner
proposeStrategy	Public	Can Modify State	onlyOwner
upgradeStrategy	Public	Can Modify State	onlyOwner

StrategyBase

StrategyBase			
Function Name	Visibility	Mutability	Modifiers
__StrategyBase_init	Public	Can Modify State	initializer
setVault	External	Can Modify State	onlyOwner
beforeDeposit	External	Can Modify State	-

CrossYieldZap			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
<Receive Ether>	External	Payable	-
estimateSwap	Public	-	-
zapInBnb	External	Payable	-
zapIn	External	Can Modify State	-
getPair	Private	-	-
swapAndDeposit	Private	Can Modify State	-
setAllowance	Private	Can Modify State	-

HotDrop			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721 Authn
supportsInterface	Public	-	-
_baseURI	Internal	-	-

HotDrop			
baseURI	Public	-	-
userTickets	Public	-	-
startLottery	External	Can Modify State	onlyOwner
calcCost	Private	-	-
mint	Public	Can Modify State	nonReentrant
burn	Public	Can Modify State	-
finalize	External	Can Modify State	nonReentrant
drawWinner	External	Can Modify State	nonReentrant
claim	External	Can Modify State	nonReentrant

RandomNumberGenerator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	VRFConsumerBase
setHotdrop	External	Can Modify State	onlyOwner
setFee	External	Can Modify State	onlyOwner
setKeyHash	External	Can Modify State	onlyOwner
withdrawTokens	External	Can Modify State	onlyOwner
getRandomNumber	External	Can Modify State	-
fulfillRandomness	Internal	Can Modify State	-

Zap

Zap			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
addLiquidity	Internal	Can Modify State	-
estimateSwap	Internal	-	-
estimateBNBSwap	External	-	-
estimateBUSDSwap	External	-	-
contributeWithBUSD	External	Can Modify State	-
contributeWithBNB	External	Payable	-

IHO			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
zapBNB	Public	Payable	whenNotPaused nonReentrant
zapBUSD	Public	Payable	whenNotPaused nonReentrant

BaseIHO			
Function Name	Visibility	Mutability	Modifiers
__BaseIHO_init	Public	Can Modify State	initializer
<Fallback>	External	Payable	-
<Receive Ether>	External	Payable	-

BaseIHO			
_contribute	Internal	Can Modify State	-
contribute	Public	Can Modify State	whenNotPaused nonReentrant
claim	Public	Can Modify State	whenNotPaused nonReentrant
getUserShare	Private	-	-
getOfferingAndRefundingAmounts	Public	-	-
readState	Public	-	-
finalize	Public	Can Modify State	onlyOwner
rescueToken	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Suggestion] Redundant code

Category: Gas Optimization Audit

Content

Ether is not accepted by default, which is a redundant code.

- cross-send/contracts/CrossSend.sol#L41-L47
- initial-hotcross-offering/contracts/BaseIHO.sol#L67-L73

```
fallback () external payable {
  revert("cannot directly accept currency transfers");
}

receive () external payable {
  revert("cannot directly receive currency transfers");
```

```
}
```

Solution

It is recommended to delete redundant code to reduce the size of the contract, thereby saving gas.

Status

Ignored; The project team response: This is intentional. We want explicit error messages.

[N2] [Suggestion] Gas optimization

Category: Gas Optimization Audit

Content

If one of the batch transfers fails, the transaction that was previously transferred normally will be reverted, but Gas has been consumed. It is a gas optimization issue here.

- cross-send/contracts/CrossSend.sol#L175-L210

```
function sendToken(
   IERC20 token,
   address[] memory recipients,
   uint256[] memory amounts
 ) private returns(uint256) {
   uint256 total = 0;

   for (uint256 i = 0; i < recipients.length ; i++) {
     require(_msgSender() != recipients[i], "sender != recipient");
     token.safeTransferFrom(_msgSender(), recipients[i], amounts[i]);
     total += amounts[i];
   }

   return total;
 }
 function sendNative(
   address[] memory recipients,
   uint256[] memory amounts
 ) private returns(uint256) {
   uint256 total = 0;
```

```
    for (uint256 i = 0; i < recipients.length ; i++) {
      total += amounts[i];

      (bool success, ) = payable(recipients[i]).call{value: amounts[i]}("");
      require(success, "native transfer failed");
    }

    return total;
  }
```

Solution

It is recommended to use if instead of require to judge the return value. If the return value is false, record the failed transfer. The front end displays the failed transfer record to the user.

Status

Ignored; The project team response: this is an complex way to achieve what we're trying to achieve so we will keep it the way it is for the reasons explained above

[N3] [Low] Risk of allowance amount abuse

Category: Authority Control Vulnerability

Content

There have an allowance amount in the contract, but the distributeTokenFee function does not limit the caller, and there is an issue of being called arbitrarily, but it can only transfer the balance of the sender to the feeCollector.

- cross-send/contracts/fee-managers/RecipientCountFee.sol#L65-L83

```
function distributeTokenFee(
    uint256 txRecipientCount,
    uint256,
    uint256,
    uint256,
    uint256,
    address sender
  ) public override {
    uint256 payableFee = txRecipientCount * tokenFeePerRecipient;
```

```
    if(payableFee < minTokenFee) {
      payableFee = minTokenFee;
    }
    else if (payableFee > maxTokenFee) {
      payableFee = maxTokenFee;
    }

    token.safeTransferFrom(sender, feeCollector, payableFee);
  }
```

Solution

It is recommended to only allow the CrossSend contract to call the distributeTokenFee function.

Status

Fixed; The issue has been fixed in commit https://github.com/hotcrosscom/cross-send/commit/87d0bf21a91079d9e21591fea7c0aff332d9929d.

[N4] [Medium] Price manipulation issue

Category: Others

Content

The IOUPrice is calculated using totalFarmingTokenBalance. Attackers can control totalFarmingTokenBalance to manipulate IOUPrice.

The current code does not find the location where IOUPrice is used.

Reference：https://slowmist.medium.com/cream-hacked-analysis-us-130-million-hacked-95c9410320ca

- cross-yield/contracts/core/CrossYield.sol#L95-L101

```
function IOUPrice() public view returns (uint256) {
    uint256 IOUSupply = totalSupply();

    return IOUSupply == 0
      ? 1e18
      : (totalFarmingTokenBalance() * 1e18) / IOUSupply;
  }
```

- cross-yield/contracts/core/CrossYield.sol#L65-L67

```
function totalFarmingTokenBalance() public view returns (uint256) {
  return farmingToken().balanceOf(address(this)) + strategy.balanceOf();
}
```

Solution

It is recommended not to use IOUPrice for other contracts as the calculation of the asset price on-chain.

Status

Ignored; The project team response: We know about this but there is a very important difference here though; Cross Yield interest-bearing tokens, at least in the current, version is not planned to be used as collateral in any lending platforms; thus manipulation of the price is irrelevant in the context in which Cross Yield is gonna be used. Also the IOUPrice is purely and informative function that is used to display values in the UI.

[N5] [Medium] Sandwich attacks issue

Category: Reordering Vulnerability

Content

There is no slippage check during swap, and there is a risk of sandwich attack.

`IPancakeRouter02(pcsRouter).swapExactTokensForTokens(cakeBalance, 0,cakeToBaseRoute, address(this), block.timestamp);`

`IPancakeRouter02(pcsRouter) .swapExactTokensForTokens(toSwap, 0, route, address(this), block.timestamp);`

Reference：

https://medium.com/coinmonks/demystify-the-dark-forest-on-ethereum-sandwich-attacks-5a3aec9fa33e

- cross-yield/contracts/libs/OptimalSwap.sol#L14-L63

```
function prepareLiquidity(
   address cake,
   address[2] memory lpTokens,
```

```
    address wbnb,
    address busd,
    address farmingToken,
    address pcsRouter,
    uint256 fee
  ) external {
    uint256 cakeBalance = IERC20(cake).balanceOf(address(this));
    bool isCakeInLp = lpTokens[0] == cake || lpTokens[1] == cake;
    bool isWbnbBased = lpTokens[0] == wbnb || lpTokens[1] == wbnb;
    address baseToken = isWbnbBased ? wbnb : busd;

    // if cake is not part of the lp token, swap all cake for the base token
    if (!isCakeInLp) {
      address[] memory cakeToBaseRoute = new address[](2);
      cakeToBaseRoute[0] = cake;
      cakeToBaseRoute[1] = baseToken;

      IPancakeRouter02(pcsRouter)
        .swapExactTokensForTokens(cakeBalance, 0, cakeToBaseRoute, address(this),
block.timestamp);
    }

    (uint256 reserve0, uint256 reserve1,) = IPancakePair(farmingToken).getReserves();
    address[] memory route = new address[](2);
    uint256 lp0Bal = IERC20(lpTokens[0]).balanceOf(address(this));
    uint256 lp1Bal = IERC20(lpTokens[1]).balanceOf(address(this));
    uint256 toSwap;

    // The possible cases here are:
    //   - Cake was not part of the LP token, so we swap for baseToken which could
either be lpToken0 or lpToken
    //   - Cake was part of the LP token, so it can either be lpTokens[0] or
lpTokens[1]
    // So, depending on which token we have the highest balance for, we swap for the
other one.
    if (lp0Bal > lp1Bal) {
      toSwap = SwapAmount.getSwapAmount(lp0Bal, reserve0, fee);

      route[0] = lpTokens[0];
      route[1] = lpTokens[1];
    } else {
      toSwap = SwapAmount.getSwapAmount(lp1Bal, reserve1, fee);

      route[0] = lpTokens[1];
      route[1] = lpTokens[0];
```

```
    }

    // Perform the swap
    IPancakeRouter02(pcsRouter)
      .swapExactTokensForTokens(toSwap, 0, route, address(this), block.timestamp);
  }
```

`IPancakeRouter02(pcsRouter).swapExactTokensForTokens(toWbnb, 0, cakeToWbnbRoute, address(this), block.timestamp);`

- cross-yield/contracts/strategies/StrategyCake.sol#L131

```
  function collectFees() internal {
    // a percentant of the cake we get from the masterchef will be used to buy BNB and transfer to this address
    // this is the total fees that will be shared amongst the protocol, stategy dev and the harvester
    uint256 toWbnb = feeManager.calculateTotalFee(IERC20(farmingToken).balanceOf(address(this)));
    IPancakeRouter02(pcsRouter).swapExactTokensForTokens(toWbnb, 0, cakeToWbnbRoute, address(this), block.timestamp);
    uint256 wbnbBal = IERC20(wbnb).balanceOf(address(this));

    // distribute hervester fee
    uint256 harvesterFeeAmount = feeManager.calculateHarvestFee(wbnbBal);
    // collectFees is called indirectly from the CrossYield contract via the beforeDeposit hook.
    // This means that _msgSender() is the CrossYield contract and not the actuall EOA account
    // that send the transaction
    IERC20(wbnb).safeTransfer(tx.origin, harvesterFeeAmount);

    // distribute protocol fee
    uint256 protocolFeeAmount = feeManager.calculateProtocolFee(wbnbBal);
    // IERC20(wbnb).safeTransfer(protocolFeeRecipient, protocolFeeAmount);
    feeProcessor.process(protocolFeeAmount);

    // distribute strategy dev fee
    uint256 strategyDevFeeAmount = feeManager.calculateStrategyDevFee(wbnbBal);
    IERC20(wbnb).safeTransfer(feeManager.strategyDev(), strategyDevFeeAmount);

    emit FeeCollected(
      toWbnb,
```

```
    harvesterFeeAmount,
    protocolFeeAmount,
    strategyDevFeeAmount,
    _msgSender()
  );
}
```

- cross-yield/contracts/strategies/StrategyCakeLP.sol#L174

```
function collectFees() internal {
  // a percentant of the cake we get from the masterchef will be used to buy BNB and
transfer to this address
  // this is the total fees that will be shared amongst the protocol, stategy dev
and the harvester
  uint256 toWbnb =
feeManager.calculateTotalFee(IERC20(cake).balanceOf(address(this)));
  IPancakeRouter02(pcsRouter).swapExactTokensForTokens(toWbnb, 0, cakeToWbnbRoute,
address(this), block.timestamp);
  uint256 wbnbBal = IERC20(wbnb).balanceOf(address(this));

  // distribute hervester fee
  uint256 harvesterFeeAmount = feeManager.calculateHarvestFee(wbnbBal);
  IERC20(wbnb).safeTransfer(_msgSender(), harvesterFeeAmount);

  // distribute protocol fee
  uint256 protocolFeeAmount = feeManager.calculateProtocolFee(wbnbBal);
  feeProcessor.process(protocolFeeAmount);

  // distribute strategy dev fee
  uint256 strategyDevFeeAmount = feeManager.calculateStrategyDevFee(wbnbBal);
  IERC20(wbnb).safeTransfer(feeManager.strategyDev(), strategyDevFeeAmount);

  emit FeeCollected(
    toWbnb,
    harvesterFeeAmount,
    protocolFeeAmount,
    strategyDevFeeAmount,
    _msgSender()
  );
}
```

- cross-yield/contracts/strategies/StrategyCake.sol

```
function harvest() public override whenNotPaused {
   // claim rewards
   IMasterChef(masterchef).leaveStaking(0);

   // because harvest can automatically be called after each user deposit
   // we might end up having multiple deposits in the same block and only one
   // would return rewards from masterchef so the rest will have 0 cake so
   // we don't need to waste gas to collect fees and call deposit
   uint256 farmingTokenBalance = balanceOfFarmingToken();
   if(farmingTokenBalance > 0) {
     collectFees();
     deposit();

     emit HarvestTriggered(_msgSender(), farmingTokenBalance);
   }
 }
```

- cross-yield/contracts/strategies/StrategyCakeLP.sol#l137-L148

```
 function harvest() external override whenNotPaused {
   // claim rewards
   IMasterChef(masterchef).deposit(poolId, 0);

   uint256 harvestedAmount = IERC20(cake).balanceOf(address(this));

   collectFees();
   addLiquidity();
   deposit();

   emit HarvestTriggered(_msgSender(), harvestedAmount);
 }
```

Solution

It is recommended to add a check for slippage to ensure that amountOutMin meets user expectations. or use chainlink oracle to check the price.

Status

Ignored; The project team response: Slippage checks and sandwich attacks are not very relevant since this tool is

based on compounding and thus people will be harvesting quite often thus the accumulated rewards will not reach the point where a sandwich attacks can take place.

[N6] [Low] Missing slippage check

Category: Reordering Vulnerability

Content

The addLiquidity function without slippage check, it doesn't have an impermanent loss check.

- cross-yield/contracts/strategies/StrategyCakeLP.sol#L218

```
function addLiquidity() internal {
   OptimalSwap.prepareLiquidity(
     cake,
     [lpToken0, lpToken1],
     wbnb,
     busd,
     farmingToken,
     pcsRouter,
     swapFee
   );

   // add liquidity to AMM on PCS
   uint256 lp0Bal = IERC20(lpToken0).balanceOf(address(this));
   uint256 lp1Bal = IERC20(lpToken1).balanceOf(address(this));

   IPancakeRouter02(pcsRouter)
     .addLiquidity(lpToken0, lpToken1, lp0Bal, lp1Bal, 0, 0, address(this),
block.timestamp);

   emit LiquidityAdded(lpToken0, lpToken1, lp0Bal, lp1Bal, _msgSender());
 }
```

Solution

It is recommended to check the slippage when adding liquidity to manage the impermanence loss.

Status

Ignored

[N7] [Low] Permission check Missing

Category: Authority Control Vulnerability

Content

There is no permission check for deposit function. The function is called by the CrossYieid contract.

- cross-yield/contracts/strategies/StrategyCakeLP.sol#L225-L231

```
function deposit() public override whenNotPaused {
  uint256 farmingTokenBalance = balanceOfFarmingToken();

  if (farmingTokenBalance > 0) {
    IMasterChef(masterchef).deposit(poolId, farmingTokenBalance);
  }
}
```

- cross-yield/contracts/strategies/StrategyCake.sol#L169-L175

```
function deposit() public override whenNotPaused {
  uint256 farmingTokenBalance = balanceOfFarmingToken();

  if (farmingTokenBalance > 0) {
    IMasterChef(masterchef).enterStaking(farmingTokenBalance);
  }
}
```

Solution

It is recommended to add a permission check, and it is clear that it can only be called by the CrossYieid contract.

Status

Ignored; The project team response: Directly calling the deposit function will most likely result in no action as there will be no farming token balance unless it's called via the harvest method. Also, the function is also called by harvest method so can allow only the CrossYield to call it

[N8] [Medium] Excessive authority issue

Category: Authority Control Vulnerability

Content

Owner can modify the address of the strategy. The new strategy may have security risks if it is not audited, which will affect the user's funds. If the private key is leaked, it will affect the user's funds.

- cross-yield/contracts/core/CrossYield.sol#L224-L236

```
function upgradeStrategy() public onlyOwner {
   require(stratCandidate.strategy != address(0), "No proposal exists");
   require(block.number > stratCandidate.proposedBlock + stratUpgradableAfter,
"Strategy cannot be replaced yet");

   emit NewStrategy(stratCandidate.strategy);

   strategy.retireStrategy();
   strategy = IStrategy(stratCandidate.strategy);
   stratCandidate.strategy = address(0);
   stratCandidate.proposedBlock = 0;

   putFundsToWork();
 }
```

- cross-yield/contracts/core/CrossYield.sol#L211-L220

```
 function proposeStrategy(address _strategy) public onlyOwner {
   require(address(this) == IStrategy(_strategy).vault(), "Invalid new strategy");

   stratCandidate = StrategyCandidate({
     strategy: _strategy,
     proposedBlock: block.number
   });

   emit NewStratCandidate(_strategy);
 }
```

Solution

It is recommended to transfer the authority of the owner to the governance contract or timelock contract, and at least

use a multi-sign contract to manage the private key.

Status

Confirmed

[N9] [Critical] Repeatable claims issue

Category: Design Logic Audit

Content

After the claim, lottery.status is not set as claimed, and it is also necessary to check whether lotteryId has been claimed when the claim is executed.

- hotdrop/contracts/HotDrop.sol#L261-L274

```
 function claim(uint256 lotteryId) external nonReentrant {
   Lottery storage lottery = lotteries[lotteryId];
   require(lottery.status == Status.WinnerDrawn, "winner not drawn");

   // if there is no winner transfer the total amount to the treasury
   if(tickets[lotteryId][lottery.winningNumber] == address(0)) {
     lottery.purchaseToken.safeTransfer(treasury,
lottery.purchaseToken.balanceOf(address(this)));
   } else if(tickets[lotteryId][lottery.winningNumber] == _msgSender()) {
     // if the ticket is the winning ticket and belongs to the user then split the
pot
     uint256 treasuryAmount = (lottery.totalRaised * lottery.treasuryFee) / FEE_BASE;
     lottery.purchaseToken.safeTransfer(treasury, treasuryAmount);
     lottery.purchaseToken.safeTransfer(_msgSender(), lottery.totalRaised -
treasuryAmount);
   }
 }
```

Solution

It is recommended to add Claimed for Status. and after the claim, lottery.status must set to claimed.

Status

Fixed; The issue has been fixed in commit

https://github.com/hotcrosscom/hotdrop/commit/26beff1e2c374e8cfebe623eaa3eb500c6ce07b8.

[N10] [Suggestion] Round plan security reminder

Category: Others

Content

And the new round of lottery can only be opened when the last round of lottery is in WinnerDrawn, otherwise randomGenerator.latestLotteryId will be updated, which will cause the old lottery round to fail to execute drawWinner function due to this check.

```
require(lotteryId == randomGenerator.latestLotteryId(), "numbers not drawn");
```

- hotdrop/contracts/HotDrop.sol#L44-L256

```
 function drawWinner(uint256 lotteryId) external nonReentrant {
   Lottery storage lottery = lotteries[lotteryId];

   require(lottery.status == Status.Close, "lottery still active");
   require(lotteryId == randomGenerator.latestLotteryId(), "numbers not drawn");

   // get the winning number based on the randomResult generated by ChainLink's
fallback
   uint256 winningNumber = randomGenerator.randomResult();
   lottery.winningNumber = winningNumber;
   lottery.status = Status.WinnerDrawn;

   emit LotteryNumberDrawn(lotteryId, winningNumber);
 }
```

Solution

It is recommended to make sure that the previous round is over, and then start a new round.

Status

Confirmed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002111020001	SlowMist Security Team	2021.10.22 - 2021.11.02	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 critical risk, 3 medium risk, 3 low risk, 3 suggestion vulnerabilities. And 1 medium risk vulnerabilities were confirmed and being fixed; 2 medium risk, 2 low risk, 2 suggestion vulnerabilities were ignored; 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist