HAECHI AUDIT

Lixir Finance

Smart Contract Security Analysis

Published on : Oct 27, 2021

Version v1.3

SMART CONTRACT HAECHI AUDIT AUDIT CERTIFICATE

HAECHI AUDIT

Smart Contract Audit Certificate

Lixir Finance

Security Report Published by HAECHI AUDIT
v1.0 Jul 13, 2021
v1.3 Oct 27, 2021

Auditor : Jasper Lee

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	-	-	-	-	-
Minor	1	-	-	1	-
Tips	3	2	-	1	-

TABLE OF CONTENTS

4 Issues (0 Critical, 0 Major, 1 Minor, 3 Tips) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring. We have the vision to empower the next generation of finance. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Sushiswap,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io
Website: audit.haechi.io

INTRODUCTION

This report was prepared to audit the security of yAxis v3 smart contract created by Lixir Finance project team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by Lixir Finance project team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

CRITICAL Critical issues must be resolved as critical flaws that can harm a wide range of users.

MAJOR Major issues require correction because they either have security problems or are implemented not as intended.

MINOR Minor issues can potentially cause problems and therefore require correction.

TIPS Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends the Lixir Finance project team improve all issues discovered.

The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, Sample.sol:20 points to the 20th line of Sample.sol file, and Sample#fallback() means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found at GitHub (https://github.com/Lixir-Team/lixir-contracts). The last commit of the code used for this Audit is "1423c9d8112f103d782bec2bff0558c6be1ca4cc".

Issues

HAECHI AUDIT found 0 critical issues, 0 major issues, and 1 minor issue. There are 3 Tips issues explained that would improve the code's usability or efficiency upon modification.

Update

[v.1.3] In update commit 68bd5cdcc442ae1f42435947f7185f89ec7c0493, 0 Minor issues, and 2 Tips issues were resolved.

Severity	Issue	Status
○ MINOR	Possible Reinitialization in *LixirStrategySimpleGWAP#initializeVault()*	(Found - v1.0) (Acknowledged - v1.3)
TIPS	Possible inoperability for Nonstandard ERC20 token	(Found - v1.0) (Acknowledged - v1.3)
TIPS	Duplicated code in *LixirStrategySimpleGWAP#checkTick*	(Found - v1.0) (Resolved - v1.1)
TIPS	Duplicated code in *LixirVault*	(Found - v1.0) (Resolved - v1.1)

OVERVIEW

Contracts subject to audit

- LixirBase
- LixirFactory
- LixirRegistry
- LixirStrategySimpleGWAP
- LixirVault
- LixirVaultETH
- LixirVaultToken
- EIP712
- IERC20Permit
- ILixirStrategy
- ILixirVault
- ILixirVaultToken
- LixirError
- SafeCast
- LixirRoles
- SqrtPriceMath

Lixir Finance smart contract has the following privileges.

- Gov
- Strategist
- Pauser
- Keeper
- Deployer
- FeeSetter

Each privilege can access the following functions.

Role	Functions
Gov	❖ *LixirVault#setStrategist()* ❖ *LixirVault#unpause()*

Role	Functions
Strategist	❖ *LixirStrategySimpleGWAP#setTickShortDuration()* ❖ *LixirStrategySimpleGWAP#setMaxTickDiff()* ❖ *LixirStrategySimpleGWAP#setSpreads()* ❖ *LixirStrategySimpleGWAP#configureVault()* ❖ *LixirVault#setKeeper()*
Pauser	❖ *LixirVault#emergencyExit()*
Keeper	❖ *LixirStrategySimpleGWAP#rebalance()*
Deployer	❖ *LixirFactory#createVault()* ❖ *LixirFactory#createVaultETH()*
FeeSetter	❖ *LixirVault#setPerformanceFee()*

FINDINGS

MINOR

Possible Reinitialization in *LixirStrategySimpleGWAP#initializeVault()*

(Found - v.1.0) (Acknowledged - v. 1.3)

```
function initializeVault(ILixirVault _vault, bytes memory data)
   external
   override
   onlyRole(LixirRoles.factory_role)
 {
   (
     uint24 fee,
     uint32 TICK_SHORT_DURATION,
     int24 MAX_TICK_DIFF,
     int24 mainSpread,
     int24 rangeSpread
   ) = abi.decode(data, (uint24, uint32, int24, int24, int24));
   require(_vault.strategy() == address(this), 'Incorrect vault strategy');
   _configureVault(
     _vault,
     fee,
     TICK_SHORT_DURATION,
     MAX_TICK_DIFF,
     mainSpread,
     rangeSpread
   );
   VaultData storage vaultData = vaultDatas[address(_vault)];
   uint32[] memory secondsAgos = new uint32[](1);
   secondsAgos[0] = 0;
   IUniswapV3Pool pool = _vault.activePool();
   (int56[] memory ticksCumulative, ) = pool.observe(secondsAgos);
   vaultData.timestamp = uint32(block.timestamp);
   vaultData.tickCumulative = ticksCumulative[0];
 }
```

[https://github.com/Lixir-Team/lixir-contracts/blob/1423c9d8112f103d782bec2bff0558c6be1ca4cc/contracts/LixirStrategySimpleGWAP.sol#L28-L56]

Issue

LixirStrategySimpleGWAP is a rebalancing strategy in Lixir Fiance that can affect the primary state of vault. *LixirStrategySimpleGWAP#initializeVault()* does not have an initialized flag, it can be re-initialized by users with a factory role, which has a significant impact on the system.

.

Recommendation

Please add the initialized flag as below.

```
bool public initialized;
function initializeVault(ILixirVault _vault, bytes memory data)
   external
   override
   onlyRole(LixirRoles.factory_role)
 {
   (
     uint24 fee,
     uint32 TICK_SHORT_DURATION,
     int24 MAX_TICK_DIFF,
     int24 mainSpread,
     int24 rangeSpread
   ) = abi.decode(data, (uint24, uint32, int24, int24, int24));
   require(!initialized, 'Already initialized');
   require(_vault.strategy() == address(this), 'Incorrect vault strategy');
   _configureVault(
     _vault,
     fee,
     TICK_SHORT_DURATION,
     MAX_TICK_DIFF,
     mainSpread,
     rangeSpread
   );
   VaultData storage vaultData = vaultDatas[address(_vault)];
   uint32[] memory secondsAgos = new uint32[](1);
   secondsAgos[0] = 0;
   IUniswapV3Pool pool = _vault.activePool();
   (int56[] memory ticksCumulative, ) = pool.observe(secondsAgos);
   vaultData.timestamp = uint32(block.timestamp);
   vaultData.tickCumulative = ticksCumulative[0];
   initialized = true;
 }
```

[https://github.com/Lixir-Team/lixir-contracts/blob/1423c9d8112f103d782bec2bff0558c6be1ca4cc/contracts/LixirStrategySimpleGWAP.sol#L28-L56]

Update

[v1.3] Lixir Finance Team confirmed that reinitialization of multiple parameters is intended by design.

💡 **TIPS**

Possible inoperability for Nonstandard ERC20 token

(Found - v.1.0) (Acknowledged - v.1.3)

Issue

LixirVault’s deposit or draw relies on the transfer function of the token, assuming that the actual token transfer amount is the same as the transfer amount parameter provided for the transfer function. If the implementation of the transfer is deflationary or maliciously implemented, *LixirVault* may not operate properly.

Recommendation

When adding a token pair, please make sure that the token is not maliciously implemented or deflationary.

Update

[v1.3] Lixir Finance Team confirmed the vulnerability. Lixir Finance Team confirmed that they have no plan to support deflationary tokens at the moment, nor ERC777 or ones that may have malicious reentrancy.

TIPS

Duplicated code in *LixirStrategySimpleGWAP#checkTick()*

(Found - v.1.0) (Resolved - v.1.3)

```
int24 diff =
  expectedTick >= tick ? expectedTick - tick : tick - expectedTick;
require(
  diff <= MAX_TICK_DIFF && diff <= MAX_TICK_DIFF,
  'Tick diff to great'
);
```

[https://github.com/Lixir-Team/lixir-contracts/blob/1423c9d8112f103d782bec2bff0558c6be1ca4cc/contracts/LixirStrategySimpleGWAP.sol#L194-L199]

Issue

If statement in *LixirStrategySimpleGWAP#checkTick()* has the same condition twice.

Recommendation

Please remove duplication as below.

```
int24 diff =
  expectedTick >= tick ? expectedTick - tick : tick - expectedTick;
require(
  diff <= MAX_TICK_DIFF,
  'Tick diff to great'
);
```

[https://github.com/Lixir-Team/lixir-contracts/blob/1423c9d8112f103d782bec2bff0558c6be1ca4cc/contracts/LixirStrategySimpleGWAP.sol#L194-L199]

Update

[v1.3] In update in commit 68bd5cdcc442ae1f42435947f7185f89ec7c0493, Lixir Finance Team fixed the issue.

💡 **TIPS**

Duplicated code in *LixirVault*

(Found - v.1.0) (Resolved - v.1.3)

```
import '@openzeppelin/contracts/utils/Pausable.sol';
import '@openzeppelin/contracts/utils/Pausable.sol';
import '@openzeppelin/contracts/token/ERC20/IERC20.sol';
import '@openzeppelin/contracts/math/Math.sol';
```

[https://github.com/Lixir-Team/lixir-contracts/blob/1423c9d8112f103d782bec2bff0558c6be1ca4cc/contracts/LixirVault.sol#L17-L20]

Issue

If statement in *LixirStrategySimpleGWAP#checkTick()* has the same condition twice.

Recommendation

Please remove duplication as below.

```
import '@openzeppelin/contracts/utils/Pausable.sol';
import '@openzeppelin/contracts/token/ERC20/IERC20.sol';
import '@openzeppelin/contracts/math/Math.sol';
```

[https://github.com/Lixir-Team/lixir-contracts/blob/1423c9d8112f103d782bec2bff0558c6be1ca4cc/contracts/LixirVault.sol#L17-L20]

Update

[v1.3] In update in commit 68bd5cdcc442ae1f42435947f7185f89ec7c0493, Lixir Finance Team fixed the issue.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects in Ethereum and Solidity. To write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results are unit test results that cover the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to having issues.

```
LixirFinance
  LixirRegistry
    #constructor()
      ✓ Should construct LixirRegistry properly
    #isGovOrDelegate()
      ✓ Should return FALSE if account parameter is not governance address or delegate address
      ✓ Should return TRUE if account parameter is governance address or delegate address
    #setFeeTo()
      ✓ Should fail if caller is not governance or delegate
      ✓ Should set fee properly
  LixirVault
    #initialize()
      ✓ Should fail if _strategist doesn't have strategist role
      ✓ Should fail if _keeper doesn't have keeper role
      ✓ Should fail if _strategy doesn't have strategy role
      ✓ Should fail if anomaly tries to reinitialize LixirVault
      ✓ Should fail if _token0 >= _token1
      ✓ Should initialize LixirVault properly
    #rebalance()
      ✓ Should fail if called by non-strategy entity
      ✓ Should fail if when paused
      ✓ Should fail if mainTickLower is smaller then TickMath.MIN_TICK
      ✓ Should fail if mainTickUpper is larger then TickMath.MAX_TICK
      ✓ Should fail if mainTickLower is larger then mainTickUpper
      ✓ Should fail if rangeTickLower0 is smaller then TickMath.MIN_TICK
      ✓ Should fail if rangeTickUpper0 is larger then TickMath.MAX_TICK
      ✓ Should fail if rangeTickLower0 is larger then rangeTickUpper0
      ✓ Should fail if rangeTickLower1 is smaller then TickMath.MIN_TICK
      ✓ Should fail if rangeTickUpper1 is larger then TickMath.MAX_TICK
      ✓ Should fail if rangeTickLower1 is larger then rangeTickUpper1
      ✓ Should rebalance properly
    #deposit()
      ✓ Should fail if when paused
      ✓ Should fail if deposit is expired
      ✓ Should fail if output amount is less than minimum amount
      ✓ Should fail if max supply is less than total supply after minting
      ✓ Should deposit properly
    #withdrawFrom()
      ✓ Should fail if withdrawal amount is larger than approved amount
      ✓ Should fail if withdrawal expired
    ✓ Should fail if balance is insufficient
    ✓ Should withdraw properly
  #withdraw()
    ✓ Should fail if withdrawal expired
    ✓ Should fail if balance is insufficient
    ✓ Should withdraw properly
  #setPerformanceFee()
    ✓ Should fail if called by non-fee_setter
    ✓ Should fail if newFee is larger than PERFORMANCE_FEE_PRECISION
    ✓ Should set performanceFee properly
  #setMaxSupply()
    ✓ Should fail if called by non-strategist
    ✓ Should set maxSupply properly
  #setKeeper()
    ✓ Should fail if called by non-strategist
    ✓ Should fail if _keeper is not granted as keeper
    ✓ Should set keeper properly
  #setStrategist()
    ✓ Should fail if called by non-governance or delegate
    ✓ Should fail if _keeper is not granted as keeper
    ✓ Should set strategist properly
  #unpause()
    ✓ Should fail if called by non-governance or delegate
    ✓ Should set unpause properly
  #calculateTotals()
    ✓ Should calculate totals properly
LixirVaultETH
  #initialize()
    ✓ Should fail if there is no weth in tokens
    ✓ Should initialize LixirVault properly
  #depositETH()
    ✓ Should fail if when paused
    ✓ Should fail if deposit is expired
    ✓ Should fail if output amount is less than minimum amount
    ✓ Should fail if max supply is less than total supply after minting
    ✓ Should deposit properly
  #withdrawETHFrom()
    ✓ Should fail if withdrawal amount is larger than approved amount
    ✓ Should fail if withdrawal expired
    ✓ Should fail if balance is insufficient
    ✓ Should withdraw properly
  #withdrawETH()
    ✓ Should fail if withdrawal expired
    ✓ Should fail if balance is insufficient
    ✓ Should withdraw properly
LixirStrategySimpleGWAP
  #initializeVault()
    ✓ Should fail if called by non-factory entity
    ✓ Should fail if vault.strategy is not consistent with current contract
    ✓ Should initialize vault properly
  #setTickShortDuration()
    ✓ Should fail if called by non-strategist entity
    ✓ Should fail if _vault is not granted as vault
    ✓ Should fail if TICK_SHORT_DURATION is smaller than 30
    ✓ Should set TickShortDuration properly
  #setMaxTickDiff()
    ✓ Should fail if called by non-strategist entity
    ✓ Should fail if _vault is not granted as vault
    ✓ Should fail if MaxTickDiff is smaller than 0
    ✓ Should set MaxTickDiff properly
  #setSpreads()
    ✓ Should fail if called by non-strategist entity
    ✓ Should fail if _vault is not granted as vault
    ✓ Should fail if caller is non-strategist entity for vault
    ✓ Should fail if mainSpread is smaller than 0
    ✓ Should fail if rangeSpread is smaller than 0
    ✓ Should set Spreads properly
  #configureVault()
    ✓ Should fail if called by non-strategist entity
    ✓ Should fail if _vault is not granted as vault
    ✓ Should fail if TICK_SHORT_DURATION is smaller than 30
    ✓ Should fail if MaxTickDiff is smaller than 0
    ✓ Should fail if mainSpread is smaller than 0
    ✓ Should fail if rangeSpread is smaller than 0
    ✓ Should set Spreads properly
  #rebalance()
    ✓ Should fail if called by non-keeper entity
    ✓ Should rebalance properly
LixirFactory
  #createVault()
    ✓ Should fail if called by non-deployer entity
    ✓ Should fail if strategy is not granted strategy_role
    ✓ Should fail if vaultImplementation is not granted vault_implementation_role
    ✓ Should fail if token0 or token1 is weth
    ✓ Should create vault properly
  #createVaultETH()
    ✓ Should fail if called by non-deployer entity
    ✓ Should fail if strategy is not granted strategy_role
    ✓ Should fail if vaultImplementation is not granted vault_implementation_role
    ✓ Should fail if there is no weth in tokens
    ✓ Should create vault properly
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
LixirBase.sol	100	100	100	100	
LixirFactory.sol	100	100	100	100	
LixirRegistry.sol	100	100	100	100	
LixirStrategySimpleGWAP.sol	100	100	100	100	
LixirVault.sol	100	100	100	100	
LixirVaultETH.sol	100	100	100	100	
LixirVaultToken.sol	100	100	100	100	
contracts/interfaces					
EIP712.sol	100	100	100	100	
IERC20Permit.sol	100	100	100	100	
ILixirStrategy.sol	100	100	100	100	
ILixirVault.sol	100	100	100	100	
ILixirVaultToken.sol	100	100	100	100	
contracts/libraries					
LixirError.sol	100	100	100	100	
LixirRoles.sol	100	100	100	100	
SafeCast.sol	100	100	100	100	
SqrtPriceMath.sol	100	100	100	100	

[Table 1] Test Case Coverage

End of Document