HashEx
Blockchain Security

Defi Helper

smart contracts final audit report

December 2021

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the DefiHelper team to perform an audit of their automate smart contracts. The audit was conducted between December 9 and December 12, 2021.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at GitHub at commit f6e0928. The scope of the audit was Solidity contracts in the specified folder.

Update: recheck was made at commit 3877d66.

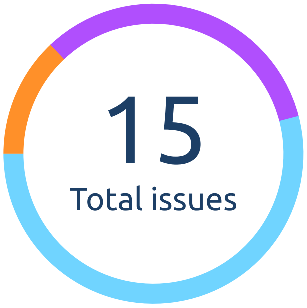
2.1 Summary

Project name	Defi Helper
URL	https://defihelper.io/
Platform	Ethereum, Avalanche Network
Language	Solidity

2.2 Contracts

Name	Address
SynthetixUniswapLpRestake	0xe890Dbb2EA4dd17ec0D1F9a2DD6a756D2C637f97
MasterChefJoeLpRestake	0xb19C5dd2cB210ff1Bd3f79126824b2cbfA4E7443
GaugeUniswapRestake	0x43A89De6F13B3077f8F13dBAC18b9D78868Fe759
Automate	

3. Found issues



🟧 High	2 (13%)
🟪 Medium	5 (33%)
🟦 Low	8 (54%)

SynthetixUniswapLpRestake

ID	Title	Severity	Status
01	Gas optimizations	🟦 Low	Acknowledged

MasterChefJoeLpRestake

ID	Title	Severity	Status
01	Rewards can be syphoned by setting a malicious router	🟧 High	Resolved
02	Lack of emergencyWithdraw mechanism	🟧 High	Resolved
03	Contract works only with LP tokens	🟪 Medium	Resolved

04	Possible lack of liquidity for router swap paths	Medium	Acknowledged
05	The run() function is susceptible to sandwich flashloan attacks	Medium	Resolved
06	Slippage and deadline parameters are not used	Low	Acknowledged
07	Lack of tests	Low	Acknowledged
08	Gas optimizations	Low	Acknowledged
09	Liquidity is added with 100% slippage	Low	Acknowledged
10	Wrong check for pending rewards	Low	Resolved

GaugeUniswapRestake

ID	Title	Severity	Status
01	A malicious pool address can be set to cyphon swapTokens	Medium	Resolved
02	Possible array out of bounds error	Medium	Resolved
03	Gas optimizations	Low	Resolved

Automate

ID	Title	Severity	Status
01	Wrong function documentation	■ Low	Resolved

4. Contracts

4.1 SynthetixUniswapLpRestake

4.1.1 Overview

The Automate contract implementation for BondAppetit protocol.

4.1.2 Issues

01. Gas optimizations

- Low ⓘ Acknowledged

The functions run(), _swap() and deposit() set token allowance to maximum on every call.

Recommendation

We recommend updating allowances only if the current token allowance is not high enough for the transaction to succeed.

4.2 MasterChefJoeLpRestake

4.2.1 Overview

Automate contract implementation that works with TraderJoe's MasterChef farming contract. A user deploys MasterChefJoeLpRestake and deposits their LP tokens. The contract deposits LP tokens to the MasterChef contract. Reward tokens from the MasterChef are swapped to LP tokens and then restaked.

4.2.2 Issues

01. Rewards can be syphoned by setting a malicious router

■ High ✓ Resolved

The function run() approves a maximum possible amount of reward tokens to a router. The address of the router is fetched from a Storage contract (out of the scope of the current audit). If a malicious router is set it can steal the rewards.

The issue also applies to the SynthetixUniswapLpRestake and GaugeUniswapRestake contracts.

Recommendation

Create a whitelist of routers and allow users to pass a router address to make the swaps.

Update

The issue was fixed. The contract creator passes the router's address on the contract creation.

02. Lack of emergencyWithdraw mechanism

■ High ✓ Resolved

The TraderJoe's MasterChef contract has an emergency withdraw mechanism. It is used in case the withdraw() function fails because of an error. In case of an error, a user won't be able to withdraw their deposited tokens.

Recommendation

Add a emergencyWithdraw() function that uses MasterChef's emergencyWithraw() function to widthdraw deposited funds.

Update

Function emergencyWithdraw() was added to the contract.

03. Contract works only with LP tokens

■ Medium ✓ Resolved

The contract assumes that every token that is staked to TraderJoe's MasterChef contract is an LP token. In a general case, this may be not true as MasterChef works with any ERC20 token, including non-LP ones. With a lack of documentation, we can't be sure that this is intended behavior.

Recommendation

Update documentation to state explicitly that the contract works only with LP token or rewrite the code to support any ERC20 token.

Update

Contract documentation was updated stating explicitly that the contract works only with LP tokens.

04. Possible lack of liquidity for router swap paths

■ Medium ⓘ Acknowledged

The function run() makes swaps for direct paths of reward token and tokens in the LP pair. As liquidity is usually added only with the token-WAVAX pair, the direct swap paths may not have enough liquidity to do the swaps or the swap will be done with a token price that isn't optimal.

The issue also applies to the SynthetixUniswapLpRestake and GaugeUniswapRestake contracts.

Recommendation

Make the swaps via WAVAX token if token0 and token1 are not WAVAX.

05. The run() function is susceptible to sandwich flashloan attacks

■ Medium ✓ Resolved

The function run can be called by anyone. If a significant amount of rewards is pending, an attacker can manipulate the rate of the reward token and perform a sandwich attack.

The issue also applies to SynthetixUniswapLpRestake and GaugeUniswapRestake contracts.

Recommendation

We recommend restricting calls run function only by EOA addresses to make sandwich attacks harder to implement.

Update

The DefiHelper team responded that authorization is done in the claim() function of the Balance contract (out of scope of current audit). It must be noted that the claim() function uses tx.origin for authorization which is considered a bad practice (see SWC-115). However, it significantly implicates the attack.

06. Slippage and deadline parameters are not used

■ Low ⓘ Acknowledged

Slippage and deadline parameters that are set in the constructor are not used in the contract.

The issue also applies to SynthetixUniswapLpRestake and GaugeUniswapRestake contracts.

Team response

The DefiHelper team responded that slippage and deadline were used by the backend to calculate the params for the run() function. They were added to the contract to add an additional layer of security and prevent the substitution of these parameters and potential loss of funds.

07. Lack of tests

■ Low ⓘ Acknowledged

The contract has no unit tests. Having full test coverage is crucially important for smart contract development

Team response

The DefiHelper team responded that tests for the MasterChefJoeLpRestake will be added ASAP.

08. Gas optimizations

■ Low ⓘ Acknowledged

Every time the function deposit() is called it approves the maximum amount of tokens to the MasterChef contract which makes a costly write operation. The same applies to approvals of reward and stake tokens in the run() and swap() functions.L118 and L119 should use local variable _stakingToken instead of stakingToken to save gas on reads from storage.

Recommendation

We recommend calling approving tokens only if the amount that should be transferred is bigger than the current allowance.

09. Liquidity is added with 100% slippage

- Low ⓘ Acknowledged

The run() function adds liquidity with zero amountOutMin parameters.

```
  function run(
    uint256 gasFee,
    uint256 _deadline,
    uint256[2] memory _outMin
  ) external bill(gasFee, "AvaxSmartcoinMasterChefJoeLPRestake") {
    ...
    uint256[2] memory amountOutMin = [uint256(0), uint256(0)];

    _addLiquidity([router, tokens[0], tokens[1]], amountIn, amountOutMin,
_deadline);
    ...
  }
```

The issue also applies to the SynthetixUniswapLpRestake contract.

10. Wrong check for pending rewards

- Low ✓ Resolved

The function run() checks if there are pending rewards in the MasterChef contract before proceeding via

```
    require(userInfo.rewardDebt > 0, "MasterChefJoeLpRestake::run: no earned");
```

The user.rewardDebt in the MasterChef contract is used for internal reward calculations and does not show if any pending rewards are available.

Recommendation

Use the pendingTokens() function to check if any rewards are available in the MasterChef contract.

4.3 GaugeUniswapRestake

4.3.1 Overview

The Automate contract implementation for Curve protocol.

4.3.2 Issues

01. A malicious pool address can be set to cyphon swapTokens

▪ Medium ✓ Resolved

The function run() approves an unlimited amount of _swapToken to the _pool address. The _pool value is set from the registry contract (out of scope of current audit) on contract initialization. If the registry returns a malicious _pool address, it can steal tokens swapTokens from the contract on run() function calls.

Recommendation

Users should check the address of the _pool variable after the contract deployment.

02. Possible array out of bounds error

▪ Medium ✓ Resolved

The function init() supposes that _swapTokenN parameter may have value from 0 to 7:

```
  function init(
    address _staking,
    address _swapToken,
    uint16 _slippage,
    uint16 _deadline
  ) external initializer {
    ...
    for (; _swapTokenN < 9; _swapTokenN++) {
      require(_swapTokenN < 8, "GaugeUniswapRestake::init: invalid swap token
address");
      if (coins[_swapTokenN] == _swapToken) break;
    }
  }
```

In functions calcTokenAmount() and _addLiquidity() there are calls to arrays of length 2 and 3 with index equal to _swapTokenN. If the _swapTokenN is bigger or equal to array size, the functions will fail.

```
  function _addLiquidity(
    address pool,
    uint256 amount,
    uint256 minOut
  ) internal {
    ...
    if (registry.get_n_coins(pool) == 3) {
      uint256[3] memory amountIn;
      amountIn[_swapTokenN] = amount;
      ...
    } else {
      uint256[2] memory amountIn;
      amountIn[_swapTokenN] = amount;
      ...
    }
  }
```

```
function calcTokenAmount(uint256 amount) external view returns (uint256) {
  ...

  if (registry.get_n_coins(pool) == 3) {
    uint256[3] memory amountIn;
    amountIn[_swapTokenN] = amount;
    return IPlainPool(pool).calc_token_amount(amountIn, true);
  } else {
    uint256[2] memory amountIn;
    amountIn[_swapTokenN] = amount;
    return IMetaPool(pool).calc_token_amount(amountIn, true);
  }
}
```

03. Gas optimizations

▪ Low ✓ Resolved

L147 should use local _staking variable instead of staking to save gas on reads from the storage.

4.4 Automate

4.4.1 Issues

01. Wrong function documentation

- Low ✓ Resolved

Modifiers whenPaused() and whenNotPaused() have wrong NatSpec documenation.

The documentation for whenPaused() modifier should state "Throws if contract unpaused".

The documentation for whenNotPaused() modifier should state "Throws if contract paused".

Update

The documentation was fixed in the update.

5. Conclusion

2 high and 5 medium severity issues were found.

This audit includes recommendations on the code improving and preventing potential attacks.

Update: all high and 4 medium severity issues were fixed in the update.

Appendix A. Issues’ severity classification

Critical. Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.

High. Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.

Medium. Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.

Low. Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.

Informational. Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

✉ contact@hashex.org

@hashexbot

blog.hashex.org

linkedin

github

twitter

HashEx
Blockchain Security