



Smart contracts security assessment

Final report

Tariff: Standard

Boltr-Farm

November 2021



0xguard.com



hello@0xguard.com

Contents

1. Introduction	3
2. Contracts checked	3
3. Procedure	3
4. Known vulnerabilities checked	4
5. Conclusion	5
6. Classification of issue severity	5
7. Issues	6
8. Conclusion	9
9. Disclaimer	10
10. Static code analysis result	11

Introduction

The report has been prepared for Boltr.farm. The code is audited after commit [fe4b3b](#). Users must check that the contracts they are interacting are the same as been audited. The recheck was done after commit [cb4a853](#).

Name	Boltr-Farm
Audit date	2021-10-01 - 2021-11-01
Language	Solidity
Platform	Polygon Network

Contracts checked

Name	Address
BoltrSwap.sol	https://polygonscan.com/address/0x6d58c4383b97e9a4e0586759e30574df31cd30d0#code
MasterChef	https://polygonscan.com/address/0x21a1d92d3f837e875e0bbbba749a7909266dbde7#code
Timelock.sol	https://polygonscan.com/address/0x9561C7239b0A2ffF9fcD104564a99cCb91d10330#code

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyse smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	p/passed
Presence of unused variables	not passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	not passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed

Reentrancy	p/passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	not passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Conclusion

test

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Delegation double spend attack (BoltrSwap.sol)

Voting mechanism of the Boltr token is susceptible to [double spend attack](#).

Recommendation: Remove voting mechanism from token if it's not going to be used or fix the delegation mechanism by transferring votes in the transfer() and transferFrom() functions.

Update: Issue was fixed by removing voting mechanism.

2. Mint is open for owner (BoltrSwap.sol)

There are 2 mint functions in the token. Owner can mint token before ownership is transferred to the MasterChef contract.

Update: The issue is mitigated, ownership of the token was transferred to the MasterChef contract.

Medium severity issues

1. Token with commissions on transfer attack (MasterChef)

The contract can be exploited if a token with commissions on transfers is added as a pool. The function deposit() does not check the real amount deposited to MasterChef contract. This leads to discrepancies in the user.amount values and actual deposited amount which leads to broken calculations in the updatePool() functions. See the [Garuda exploit](#) for example.

Recommendation: Check real deposited amount by calling balanceOf(address(this)) before and after the tokens are transferred to the MasterChef contract.

Update: Issue was fixed.

2. Functions massupdatePools() may run out of block gas limit (MasterChef)

Functions massUpdatePools() and updateEmissionRate() may run out of block gas limit if a big number of pools is added as they iterate over an unlimited number of pools.

```
function massUpdatePools() public {
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(pid); }
}
```

Recommendation: Owner of the MasterChef contract should be aware that adding a big number of pools can break some functions of the contract.

3. Emission rate not limited (MasterChef)

The owner of the MasterChef can set an arbitrary big value for emission rate. The emission rate must be capped.

```
function updateEmissionRate(uint256 _BOLPerBlock) public onlyOwner {
    massUpdatePools();
    emit EmissionRateUpdated(msg.sender, BOLPerBlock, _BOLPerBlock);
    BOLPerBlock = _BOLPerBlock;
}
```

Recommendation: Cap the emission rate.

Update: Max emission rate was capped by 1 token per block.

4. Pool with same LP token can be added twice (MasterChef)

If a pool with the same LP token is added more than once, it will break the calculations of the rewards.

Recommendation: Add a mapping(address => bool) to check if the pool with same LP token has already been added.

Update: Issue was fixed in the update

Low severity issues

1. Outdated compiler version (MasterChef)

Outdated compiler version is used (v0.6.12+commit.27d51765).

Recommendation: We recommend using the latest stable version of the Solidity compiler.

Conclusion

Boltr-Farm BoltrSwap.sol, MasterChef, Timelock.sol contracts were audited. 2 high, 4 medium, 1 low severity issues were found. All high and 3 medium severity issues were fixed (see Updates section below the issues).

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Static code analysis result

contracts/BoltrSwap.sol: Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see <https://spdx.org> for more information.

contracts/MasterChef.sol: Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see <https://spdx.org> for more information.

contracts/libs/Migrations.sol: Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see <https://spdx.org> for more information.

contracts/libs/MockKRC20.sol: Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see <https://spdx.org> for more information.

contracts/BoltrSwap.sol:27:5: Warning: Documentation tag on non-public state variables will be disallowed in 0.7.0. You will need to use the `@dev` tag explicitly.

```
/// @notice A record of each accounts delegate
```

```
^-----^
```

contracts/Timelock.sol:122:51: Warning: Using ".value(...)" is deprecated. Use "{value: ...}" instead.

```
(bool success, bytes memory returnData) = target.call.value(value)(callData);
```

^-----^

INFO:Detectors:

MasterChef.safeBOLTransfer(address,uint256) (contracts/MasterChef.sol#272-279) ignores return value by BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)

MasterChef.safeBOLTransfer(address,uint256) (contracts/MasterChef.sol#272-279) ignores return value by BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer>

INFO:Detectors:

MasterChef.pendingBOL(uint256,address) (contracts/MasterChef.sol#145-157) performs a multiplication on the result of a division:

-BOLReward = multiplier.mul(BOLPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (contracts/MasterChef.sol#152)

-accBOLPerShare = accBOLPerShare.add(BOLReward.mul(1e12).div(lpSupply)) (contracts/MasterChef.sol#153)

MasterChef.updatePool(uint256) (contracts/MasterChef.sol#174-190) performs a multiplication on the result of a division:

-BOLReward = multiplier.mul(BOLPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (contracts/MasterChef.sol#185)

-pool.accBOLPerShare = pool.accBOLPerShare.add(BOLReward.mul(1e12).div(lpSupply)) (contracts/MasterChef.sol#188)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply>

INFO:Detectors:

BoltrSwap._writeCheckpoint(address,uint32,uint256,uint256) (contracts/BoltrSwap.sol#220-238) uses a dangerous strict equality:

- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (contracts/BoltrSwap.sol#230)

MasterChef.updatePool(uint256) (contracts/MasterChef.sol#174-190) uses a dangerous strict equality:

- lpSupply == 0 || pool.allocPoint == 0 (contracts/MasterChef.sol#180)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities>

INFO:Detectors:

Reentrancy in MasterChef.add(uint256,IKRC20,uint16,uint256,bool) (contracts/MasterChef.sol#108-124):

External calls:

- massUpdatePools() (contracts/MasterChef.sol#112)
- BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
- BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)

State variables written after the call(s):

- poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0,_depositFeeBP,_harvestInterval)) (contracts/MasterChef.sol#116-123)

- totalAllocPoint = totalAllocPoint.add(_allocPoint) (contracts/MasterChef.sol#115)

Reentrancy in MasterChef.deposit(uint256,uint256) (contracts/MasterChef.sol#193-212):

External calls:

- updatePool(_pid) (contracts/MasterChef.sol#196)
 - BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
 - BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)
- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#198)
 - BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)
 - BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)

State variables written after the call(s):

- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#198)
 - user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval) (contracts/MasterChef.sol#248)
 - user.rewardLockedUp = 0 (contracts/MasterChef.sol#258)
 - user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval) (contracts/MasterChef.sol#259)
 - user.rewardLockedUp = user.rewardLockedUp.add(pending) (contracts/MasterChef.sol#265)

Reentrancy in MasterChef.deposit(uint256,uint256) (contracts/MasterChef.sol#193-212):

External calls:

- updatePool(_pid) (contracts/MasterChef.sol#196)
 - BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)

- BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)
- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#198)
- BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)
- BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (contracts/MasterChef.sol#200)
- pool.lpToken.safeTransfer(feeAddress,depositFee) (contracts/MasterChef.sol#204)

State variables written after the call(s):

- user.amount = user.amount.add(_amount).sub(depositFee) (contracts/MasterChef.sol#205)

Reentrancy in MasterChef.deposit(uint256,uint256) (contracts/MasterChef.sol#193-212):

External calls:

- updatePool(_pid) (contracts/MasterChef.sol#196)
- BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
- BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)
- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#198)
- BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)
- BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (contracts/MasterChef.sol#200)

State variables written after the call(s):

- `user.amount = user.amount.add(_amount)` (contracts/MasterChef.sol#207)

Reentrancy in `MasterChef.set(uint256,uint256,uint16,uint256,bool)` (contracts/MasterChef.sol#127-137):

External calls:

- `massUpdatePools()` (contracts/MasterChef.sol#131)
 - `BOL.mint(devAddress,BOLReward.div(10))` (contracts/MasterChef.sol#186)
 - `BOL.mint(address(this),BOLReward)` (contracts/MasterChef.sol#187)

State variables written after the call(s):

- `poolInfo[_pid].allocPoint = _allocPoint` (contracts/MasterChef.sol#134)
- `poolInfo[_pid].depositFeeBP = _depositFeeBP` (contracts/MasterChef.sol#135)
- `poolInfo[_pid].harvestInterval = _harvestInterval` (contracts/MasterChef.sol#136)
- `totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint)` (contracts/MasterChef.sol#133)

Reentrancy in `MasterChef.updateEmissionRate(uint256)` (contracts/MasterChef.sol#295-299):

External calls:

- `massUpdatePools()` (contracts/MasterChef.sol#296)
 - `BOL.mint(devAddress,BOLReward.div(10))` (contracts/MasterChef.sol#186)
 - `BOL.mint(address(this),BOLReward)` (contracts/MasterChef.sol#187)

State variables written after the call(s):

- BOLPerBlock = _BOLPerBlock (contracts/MasterChef.sol#298)

Reentrancy in MasterChef.updatePool(uint256) (contracts/MasterChef.sol#174-190):

External calls:

- BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
- BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)

State variables written after the call(s):

- pool.accBOLPerShare = pool.accBOLPerShare.add(BOLReward.mul(1e12).div(lpSupply)) (contracts/MasterChef.sol#188)
- pool.lastRewardBlock = block.number (contracts/MasterChef.sol#189)

Reentrancy in MasterChef.withdraw(uint256,uint256) (contracts/MasterChef.sol#215-227):

External calls:

- updatePool(_pid) (contracts/MasterChef.sol#219)
 - BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
 - BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)
- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#220)
 - BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)
 - BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)

State variables written after the call(s):

- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#220)
 - user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval) (contracts/MasterChef.sol#248)
 - user.rewardLockedUp = 0 (contracts/MasterChef.sol#258)
 - user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval) (contracts/MasterChef.sol#259)
 - user.rewardLockedUp = user.rewardLockedUp.add(pending) (contracts/MasterChef.sol#265)
- user.amount = user.amount.sub(_amount) (contracts/MasterChef.sol#222)

Reentrancy in MasterChef.withdraw(uint256,uint256) (contracts/MasterChef.sol#215-227):

External calls:

- updatePool(_pid) (contracts/MasterChef.sol#219)
 - BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
 - BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)
- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#220)
 - BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)
 - BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)
- pool.lpToken.safeTransfer(address(msg.sender),_amount) (contracts/MasterChef.sol#223)

State variables written after the call(s):

- user.rewardDebt = user.amount.mul(pool.accBOLPerShare).div(1e12) (contracts/

MasterChef.sol#225)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1>

INFO:Detectors:

KRC20.constructor(string,string).name (contracts/libs/KRC20.sol#58) shadows:

- KRC20.name() (contracts/libs/KRC20.sol#74-76) (function)
- IKRC20.name() (contracts/libs/IKRC20.sol#24) (function)

KRC20.constructor(string,string).symbol (contracts/libs/KRC20.sol#58) shadows:

- KRC20.symbol() (contracts/libs/KRC20.sol#88-90) (function)
- IKRC20.symbol() (contracts/libs/IKRC20.sol#19) (function)

KRC20.allowance(address,address).owner (contracts/libs/KRC20.sol#122) shadows:

- Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#35-37) (function)

KRC20._approve(address,address,uint256).owner (contracts/libs/KRC20.sol#294) shadows:

- Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#35-37) (function)

MockKRC20.constructor(string,string,uint256).name (contracts/libs/MockKRC20.sol#7) shadows:

- KRC20.name() (contracts/libs/KRC20.sol#74-76) (function)
- IKRC20.name() (contracts/libs/IKRC20.sol#24) (function)

MockKRC20.constructor(string,string,uint256).symbol (contracts/libs/MockKRC20.sol#8) shadows:

- KRC20.symbol() (contracts/libs/KRC20.sol#88-90) (function)
- IKRC20.symbol() (contracts/libs/IKRC20.sol#19) (function)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing>

INFO:Detectors:

Timelock.constructor(address,uint256).admin_ (contracts/Timelock.sol#41) lacks a zero-check on :

- admin = admin_ (contracts/Timelock.sol#45)

Timelock.setPendingAdmin(address).pendingAdmin_ (contracts/Timelock.sol#70) lacks a zero-check on :

- pendingAdmin = pendingAdmin_ (contracts/Timelock.sol#78)

Timelock.executeTransaction(address,uint256,string,bytes,uint256).target (contracts/Timelock.sol#103) lacks a zero-check on :

- (success,returnData) = target.call.value(value)(callData) (contracts/Timelock.sol#122)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation>

INFO:Detectors:

Modifier Migrations.restricted() (contracts/libs/Migrations.sol#7-9) does not always execute _; or revert
Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-modifier>

INFO:Detectors:

Reentrancy in MasterChef.deposit(uint256,uint256) (contracts/MasterChef.sol#193-212):

External calls:

- updatePool(_pid) (contracts/MasterChef.sol#196)
 - BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
 - BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)
- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#198)
 - BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)
 - BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)

Event emitted after the call(s):

- RewardLockedUp(msg.sender,_pid,pending) (contracts/MasterChef.sol#267)
 - payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#198)

Reentrancy in MasterChef.deposit(uint256,uint256) (contracts/MasterChef.sol#193-212):

External calls:

- updatePool(_pid) (contracts/MasterChef.sol#196)
 - BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
 - BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)
- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#198)
 - BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)
 - BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (contracts/MasterChef.sol#200)

- pool.lpToken.safeTransfer(feeAddress,depositFee) (contracts/MasterChef.sol#204)

Event emitted after the call(s):

- Deposit(msg.sender,_pid,_amount) (contracts/MasterChef.sol#211)

Reentrancy in MasterChef.emergencyWithdraw(uint256) (contracts/MasterChef.sol#230-240):

External calls:

- pool.lpToken.safeTransfer(address(msg.sender),amount) (contracts/MasterChef.sol#238)

Event emitted after the call(s):

- EmergencyWithdraw(msg.sender,_pid,amount) (contracts/MasterChef.sol#239)

Reentrancy in Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#103-128):

External calls:

- (success,returnData) = target.call.value(value)(callData) (contracts/Timelock.sol#122)

Event emitted after the call(s):

- ExecuteTransaction(txHash,target,value,signature,data,eta) (contracts/Timelock.sol#125)

Reentrancy in MasterChef.updateEmissionRate(uint256) (contracts/MasterChef.sol#295-299):

External calls:

- massUpdatePools() (contracts/MasterChef.sol#296)
- BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
- BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)

Event emitted after the call(s):

- EmissionRateUpdated(msg.sender,BOLPerBlock,_BOLPerBlock) (contracts/MasterChef.sol#297)

Reentrancy in MasterChef.withdraw(uint256,uint256) (contracts/MasterChef.sol#215-227):

External calls:

- updatePool(_pid) (contracts/MasterChef.sol#219)
 - BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
 - BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)
- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#220)
 - BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)
 - BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)

Event emitted after the call(s):

- RewardLockedUp(msg.sender,_pid,pending) (contracts/MasterChef.sol#267)
- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#220)

Reentrancy in MasterChef.withdraw(uint256,uint256) (contracts/MasterChef.sol#215-227):

External calls:

- updatePool(_pid) (contracts/MasterChef.sol#219)
 - BOL.mint(devAddress,BOLReward.div(10)) (contracts/MasterChef.sol#186)
 - BOL.mint(address(this),BOLReward) (contracts/MasterChef.sol#187)

- payOrLockupPendingBOL(_pid) (contracts/MasterChef.sol#220)
 - BOL.transfer(_to,BOLBal) (contracts/MasterChef.sol#275)
 - BOL.transfer(_to,_amount) (contracts/MasterChef.sol#277)
- pool.lpToken.safeTransfer(address(msg.sender),_amount) (contracts/MasterChef.sol#223)

Event emitted after the call(s):

- Withdraw(msg.sender,_pid,_amount) (contracts/MasterChef.sol#226)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3>

INFO:Detectors:

BoltrSwap.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (contracts/BoltrSwap.sol#86-127) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(now <= expiry,BOL::delegateBySig: signature expired) (contracts/BoltrSwap.sol#125)

MasterChef.canHarvest(uint256,address) (contracts/MasterChef.sol#160-163) uses timestamp for comparisons

Dangerous comparisons:

- block.timestamp >= user.nextHarvestUntil (contracts/MasterChef.sol#162)

Timelock.queueTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#83-92) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(eta >= getBlockTimestamp().add(delay),Timelock::queueTransaction: Estimated execution block must satisfy delay.) (contracts/Timelock.sol#85)

Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#103-128) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(getBlockTimestamp() >= eta,Timelock::executeTransaction: Transaction hasn't surpassed time lock.) (contracts/Timelock.sol#108)

- require(bool,string)(getBlockTimestamp() <= eta.add(GRACE_PERIOD),Timelock::executeTransaction: Transaction is stale.) (contracts/Timelock.sol#109)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp>

INFO:Detectors:

Address.isContract(address) (node_modules/@openzeppelin/contracts/Utils/Address.sol#26-35) uses assembly

- INLINE ASM (node_modules/@openzeppelin/contracts/Utils/Address.sol#33)

Address._verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/Utils/Address.sol#171-188) uses assembly

- INLINE ASM (node_modules/@openzeppelin/contracts/Utils/Address.sol#180-183)

BoltrSwap.getChainId() (contracts/BoltrSwap.sol#245-249) uses assembly

- INLINE ASM (contracts/BoltrSwap.sol#247)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage>

INFO:Detectors:

Different versions of Solidity is used:

- Version used: ['0.6.12', '>=0.4.0', '>=0.4.25<0.7.0', '>=0.6.0<0.8.0', '>=0.6.2<0.8.0', '^0.6.0', '^0.6.12']
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3)
- >=0.6.2<0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/ReentrancyGuard.sol#3)
- 0.6.12 (contracts/BoltrSwap.sol#1)
- 0.6.12 (contracts/MasterChef.sol#1)
- 0.6.12 (contracts/Timelock.sol#14)
- ^0.6.12 (contracts/libs/IKRC20.sol#3)
- >=0.4.0 (contracts/libs/KRC20.sol#3)
- >=0.4.25<0.7.0 (contracts/libs/Migrations.sol#1)
- 0.6.12 (contracts/libs/MockKRC20.sol#1)
- ^0.6.0 (contracts/libs/SafeKRC20.sol#3)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used>

INFO:Detectors:

KRC20._burn(address,uint256) (contracts/libs/KRC20.sol#272-278) is never used and should be removed

KRC20._burnFrom(address,uint256) (contracts/libs/KRC20.sol#311-318) is never used and should be removed

SafeKRC20.safeApprove(IKRC20,address,uint256) (contracts/libs/SafeKRC20.sol#46-60) is never used and should be removed

SafeKRC20.safeDecreaseAllowance(IKRC20,address,uint256) (contracts/libs/SafeKRC20.sol#71-81) is never used and should be removed

SafeKRC20.safeIncreaseAllowance(IKRC20,address,uint256) (contracts/libs/SafeKRC20.sol#62-69) is never used and should be removed

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code>

INFO:Detectors:

Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#3) is too complex

Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/math/SafeMath.sol#3) is too complex

Pragma version>=0.6.2<0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3) is too complex

Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#3) is too complex

Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts/utils/

ReentrancyGuard.sol#3) is too complex

Pragma version>=0.4.0 (contracts/libs/KRC20.sol#3) allows old versions

Pragma version>=0.4.25<0.7.0 (contracts/libs/Migrations.sol#1) allows old versions

Pragma version^0.6.0 (contracts/libs/SafeKRC20.sol#3) allows old versions

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>

INFO:Detectors:

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#53-59):

- (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#57)

Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-121):

- (success, returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)

Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#139-145):

- (success, returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#143)

Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#163-169):

- (success, returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#167)

Low level call in `Timelock.executeTransaction(address,uint256,string,bytes,uint256)` (contracts/Timelock.sol#103-128):

- `(success,returnData) = target.call.value(value)(callData)` (contracts/Timelock.sol#122)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls>

INFO:Detectors:

Parameter `BoltrSwap.mint(address,uint256)._to` (contracts/BoltrSwap.sol#16) is not in mixedCase

Parameter `BoltrSwap.mint(address,uint256)._amount` (contracts/BoltrSwap.sol#16) is not in mixedCase

Variable `BoltrSwap._delegates` (contracts/BoltrSwap.sol#28) is not in mixedCase

Parameter `MasterChef.add(uint256,IKRC20,uint16,uint256,bool)._allocPoint` (contracts/MasterChef.sol#108) is not in mixedCase

Parameter `MasterChef.add(uint256,IKRC20,uint16,uint256,bool)._lpToken` (contracts/MasterChef.sol#108) is not in mixedCase

Parameter `MasterChef.add(uint256,IKRC20,uint16,uint256,bool)._depositFeeBP` (contracts/MasterChef.sol#108) is not in mixedCase

Parameter `MasterChef.add(uint256,IKRC20,uint16,uint256,bool)._harvestInterval` (contracts/MasterChef.sol#108) is not in mixedCase

Parameter `MasterChef.add(uint256,IKRC20,uint16,uint256,bool)._withUpdate` (contracts/MasterChef.sol#108) is not in mixedCase

Parameter `MasterChef.set(uint256,uint256,uint16,uint256,bool)._pid` (contracts/MasterChef.sol#127) is not in mixedCase

Parameter `MasterChef.set(uint256,uint256,uint16,uint256,bool)._allocPoint` (contracts/

MasterChef.sol#127) is not in mixedCase

Parameter MasterChef.set(uint256,uint256,uint16,uint256,bool)._depositFeeBP (contracts/MasterChef.sol#127) is not in mixedCase

Parameter MasterChef.set(uint256,uint256,uint16,uint256,bool)._harvestInterval (contracts/MasterChef.sol#127) is not in mixedCase

Parameter MasterChef.set(uint256,uint256,uint16,uint256,bool)._withUpdate (contracts/MasterChef.sol#127) is not in mixedCase

Parameter MasterChef.getMultiplier(uint256,uint256)._from (contracts/MasterChef.sol#140) is not in mixedCase

Parameter MasterChef.getMultiplier(uint256,uint256)._to (contracts/MasterChef.sol#140) is not in mixedCase

Parameter MasterChef.pendingBOL(uint256,address)._pid (contracts/MasterChef.sol#145) is not in mixedCase

Parameter MasterChef.pendingBOL(uint256,address)._user (contracts/MasterChef.sol#145) is not in mixedCase

Parameter MasterChef.canHarvest(uint256,address)._pid (contracts/MasterChef.sol#160) is not in mixedCase

Parameter MasterChef.canHarvest(uint256,address)._user (contracts/MasterChef.sol#160) is not in mixedCase

Parameter MasterChef.updatePool(uint256)._pid (contracts/MasterChef.sol#174) is not in mixedCase

Parameter MasterChef.deposit(uint256,uint256)._pid (contracts/MasterChef.sol#193) is not in mixedCase

Parameter MasterChef.deposit(uint256,uint256)._amount (contracts/MasterChef.sol#193) is not in mixedCase

Parameter MasterChef.withdraw(uint256,uint256)._pid (contracts/MasterChef.sol#215) is not in mixedCase

Parameter MasterChef.withdraw(uint256,uint256)._amount (contracts/MasterChef.sol#215) is not in mixedCase

Parameter MasterChef.emergencyWithdraw(uint256)._pid (contracts/MasterChef.sol#230) is not in mixedCase

Parameter MasterChef.payOrLockupPendingBOL(uint256)._pid (contracts/MasterChef.sol#243) is not in mixedCase

Parameter MasterChef.safeBOLTransfer(address,uint256)._to (contracts/MasterChef.sol#272) is not in mixedCase

Parameter MasterChef.safeBOLTransfer(address,uint256)._amount (contracts/MasterChef.sol#272) is not in mixedCase

Parameter MasterChef.setDevAddress(address)._devAddress (contracts/MasterChef.sol#282) is not in mixedCase

Parameter MasterChef.setFeeAddress(address)._feeAddress (contracts/MasterChef.sol#288) is not in mixedCase

Parameter MasterChef.updateEmissionRate(uint256)._BOLPerBlock (contracts/MasterChef.sol#295) is not in mixedCase

Variable MasterChef.BOL (contracts/MasterChef.sol#59) is not in mixedCase

Variable MasterChef.BOLPerBlock (contracts/MasterChef.sol#65) is not in mixedCase

Variable `Timelock.admin_initialized` (`contracts/Timelock.sol#36`) is not in mixedCase

Variable `Migrations.last_completed_migration` (`contracts/libs/Migrations.sol#5`) is not in mixedCase

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions>

INFO:Detectors:

Redundant expression `"this (node_modules/@openzeppelin/contracts/utils/Context.sol#21)"`
`inContext (node_modules/@openzeppelin/contracts/utils/Context.sol#15-24)`

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements>

INFO:Detectors:

`renounceOwnership()` should be declared external:

- `Ownable.renounceOwnership()` (`node_modules/@openzeppelin/contracts/access/Ownable.sol#54-57`)

`transferOwnership(address)` should be declared external:

- `Ownable.transferOwnership(address)` (`node_modules/@openzeppelin/contracts/access/Ownable.sol#63-67`)

`mint(address,uint256)` should be declared external:

- `BoltrSwap.mint(address,uint256)` (`contracts/BoltrSwap.sol#16-19`)

`add(uint256,IKRC20,uint16,uint256,bool)` should be declared external:

- `MasterChef.add(uint256,IKRC20,uint16,uint256,bool)` (`contracts/MasterChef.sol#108-124`)

`set(uint256,uint256,uint16,uint256,bool)` should be declared external:

- MasterChef.set(uint256,uint256,uint16,uint256,bool) (contracts/MasterChef.sol#127-137)

deposit(uint256,uint256) should be declared external:

- MasterChef.deposit(uint256,uint256) (contracts/MasterChef.sol#193-212)

withdraw(uint256,uint256) should be declared external:

- MasterChef.withdraw(uint256,uint256) (contracts/MasterChef.sol#215-227)

emergencyWithdraw(uint256) should be declared external:

- MasterChef.emergencyWithdraw(uint256) (contracts/MasterChef.sol#230-240)

setDevAddress(address) should be declared external:

- MasterChef.setDevAddress(address) (contracts/MasterChef.sol#282-286)

setFeeAddress(address) should be declared external:

- MasterChef.setFeeAddress(address) (contracts/MasterChef.sol#288-292)

updateEmissionRate(uint256) should be declared external:

- MasterChef.updateEmissionRate(uint256) (contracts/MasterChef.sol#295-299)

setDelay(uint256) should be declared external:

- Timelock.setDelay(uint256) (contracts/Timelock.sol#53-60)

acceptAdmin() should be declared external:

- Timelock.acceptAdmin() (contracts/Timelock.sol#62-68)

setPendingAdmin(address) should be declared external:

- Timelock.setPendingAdmin(address) (contracts/Timelock.sol#70-81)

queueTransaction(address,uint256,string,bytes,uint256) should be declared external:

- Timelock.queueTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#83-92)

cancelTransaction(address,uint256,string,bytes,uint256) should be declared external:

- Timelock.cancelTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#94-101)

executeTransaction(address,uint256,string,bytes,uint256) should be declared external:

- Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/Timelock.sol#103-128)

decimals() should be declared external:

- KRC20.decimals() (contracts/libs/KRC20.sol#81-83)

symbol() should be declared external:

- KRC20.symbol() (contracts/libs/KRC20.sol#88-90)

totalSupply() should be declared external:

- KRC20.totalSupply() (contracts/libs/KRC20.sol#95-97)

transfer(address,uint256) should be declared external:

- KRC20.transfer(address,uint256) (contracts/libs/KRC20.sol#114-117)

allowance(address,address) should be declared external:

- KRC20.allowance(address,address) (contracts/libs/KRC20.sol#122-124)

approve(address,uint256) should be declared external:

- KRC20.approve(address,uint256) (contracts/libs/KRC20.sol#133-136)

transferFrom(address,address,uint256) should be declared external:

- KRC20.transferFrom(address,address,uint256) (contracts/libs/KRC20.sol#150-162)

increaseAllowance(address,uint256) should be declared external:

- KRC20.increaseAllowance(address,uint256) (contracts/libs/KRC20.sol#176-179)

decreaseAllowance(address,uint256) should be declared external:

- KRC20.decreaseAllowance(address,uint256) (contracts/libs/KRC20.sol#195-202)

mint(uint256) should be declared external:

- KRC20.mint(uint256) (contracts/libs/KRC20.sol#212-215)

setCompleted(uint256) should be declared external:

- Migrations.setCompleted(uint256) (contracts/libs/Migrations.sol#15-17)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external>



 Guard