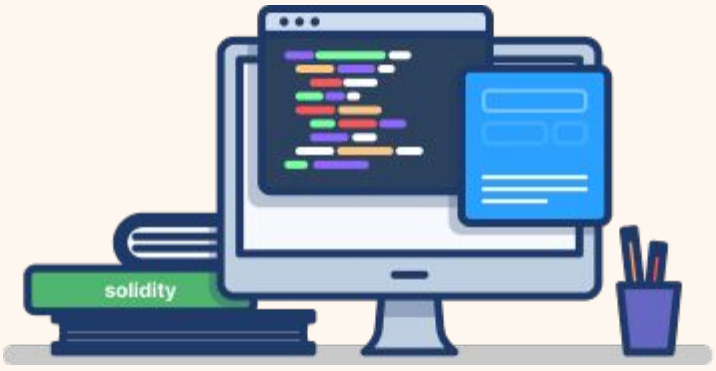
Coinbae Audit



Spacemine Defi November 2020

Contents

Introduction

Audit:

In November 2020 Coinbae's security audit report devision performed a security audit for the Spacemine DeFi team, prior to deployment for the contract that can be found at:

https://github.com/SpaceMineDeFi/SpaceMinePOC/blob/main/contracts/MineLiquidityMining.sol

Spacemine DeFi:

Spacemine DeFi will feature a uniswap like decentralized exchange exchange with various staking pools.

Scope of the audit:

The following Coinbae audit will cover assertions and property checking, ERC Standards, solidity coding best practices, conformity to the solidity style guide.

Audit Report Scope

Assertions and Property Checking:

1. Solidity assert violation.
2. Solidity AssertionFailed event.

ERC Standards:

1. Incorrect ERC20 implementation.

Solidity Coding Best Practices:

1. Outdated compiler version.
2. No or floating compiler version set.
3. Use of right-to-left-override control character.
4. Shadowing of built-in symbol.
5. Incorrect constructor name.
6. State variable shadows another state variable.
7. Local variable shadows a state variable.
8. Function parameter shadows a state variable.
9. Named return value shadows a state variable.
10. Unary operation without effect Solidity code analysis.
11. Unary operation directly after assignment.
12. Unused state variable.
13. Unused local variable.
14. Function visibility is not set.
15. State variable visibility is not set.

Solidity Coding Best Practices (Continued):

16. Use of deprecated functions: call code(), sha3(), ...
17. Use of deprecated global variables (msg.gas, ...).
18. Use of deprecated keywords (throw, var).
19. Incorrect function state mutability.
20. Does the code conform to the Solidity styleguide.

Convert code to conform Solidity styleguide:

1. Convert all code so that it is structured accordingly the Solidity styleguide.

Audit Report Scope

Categories:

High Severity:

High severity issues opens the contract up for exploitation from malicious actors. We do not recommend deploying contracts with high severity issues.

Medium Severity Issues:

Medium severity issues are errors found in contracts that hampers the effectiveness of the contract and may cause outcomes when interacting with the contract. It is still recommended to fix these issues.

Low Severity Issues:

Low severity issues are warning of minor impact on the overall integrity of the contract. These can be fixed with less urgency.

Audit Report

43	43	0
Identified	Confirmed	Critical

0	35	8
High	Medium	Low

Analysis:
https://github.com/SpaceMineDeFi/SpaceMinePOC/blob/main/contracts/MineLiquidityMining.sol

Risk:
Low

Low Severity Issues:

A floating pragma is set SWC-103:

The current pragma Solidity directive is ""^0.6.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Affected lines:

1. pragma solidity ^0.6.12; [#1]

Potential use of "block.number" as source of randomness (SWC-120):

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Affected lines:

1. uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock; [#927]
2. uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock; [#927]
3. if (block.number > pool.lastRewardBlock && lpSupply != 0) { [#953]
4. if (block.number <= pool.lastRewardBlock) { [#978]
5. pool.lastRewardBlock = block.number; [#983]
6. pool.lastRewardBlock = block.number; [#990]
7. user.requestBlock = block.number; [#1063]

Low Severity Issues:

A control flow decision is made based on The block.timestamp environment variable. SWC-116

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Affected lines:

require(block.timestamp <= expiry, "MINE::delegateBySig: signature expired"); [#176]

Low Severity Issues:

Potential use of "block.number" as source of randomness (SWC-120):

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Affected lines:

1. require(blockNumber < block.number, "MINE::getPriorVotes: not yet determined"); [#206]
2. uint32 blockNumber = safe32(block.number, "MINE::_writeCheckpoint: block number exceeds 32 bits"); [#279]
3. require(blockNumber < block.number, "MINE::getPriorVotes: not yet determined"); [#206]

Low Severity Issues:

Unused function parameter "from", “to”, “amount”, SWC-131:

The value of the function parameter "from", “to”, “amount” for the function "_beforeTokenTransfer" of contract "ERC20" does not seem to be used anywhere in "_beforeTokenTransfer".

Affected lines:

1. function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { } [#305]
2. function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { } [#305]
3. function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { } [#30]

Medium Severity Issues:

Function could be marked as external SWC-000:

The function definition of the following is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Affected lines:

1. function totalSupply() public view returns(uint256) { return _lpTotalSupply; }[#61-63]
2. function stake(uint amount) public shouldStarted { updateRewards(msg.sender); checkHalving(); require(!address(msg.sender).isContract(), "Please use your individual account."); if (rewards[msg.sender] > 0) { mine.safeTransfer(msg.sender, rewards[msg.sender]); } lpToken.safeTransferFrom(msg.sender, address(this), amount); _lpTotalSupply = _lpTotalSupply.add(amount); _lpBalances[msg.sender] = _lpBalances[msg.sender].add(amount); rewards[msg.sender] = 0; distributeDevFund(); emit Stake(msg.sender, amount); } [#69-82]
3. function withdraw(uint amount) public shouldStarted { updateRewards(msg.sender); checkHalving(); require(amount <= _lpBalances[msg.sender] && _lpBalances[msg.sender] > 0, "Bad withdraw."); if (rewards[msg.sender] > 0) { mine.safeTransfer(msg.sender, rewards[msg.sender]); } lpToken.safeTransfer(msg.sender, amount); _lpTotalSupply = _lpTotalSupply.sub(amount); _lpBalances[msg.sender] = _lpBalances[msg.sender].sub(amount); rewards[msg.sender] = 0; distributeDevFund(); emit Withdraw(msg.sender, amount); } [#84-97]

Medium Severity Issues:

Function could be marked as external SWC-000:

The function definition of the following is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Affected lines:

1. function claim(uint amount) public shouldStarted { updateRewards(msg.sender); checkHalving(); require(amount <= rewards[msg.sender] && rewards[msg.sender] > 0, "Bad claim."); rewards[msg.sender] = rewards[msg.sender].sub(amount); mine.safeTransfer(msg.sender, amount); distributeDevFund(); emit Claim(msg.sender, amount); } [#99-107]
2. function transferDevAddr(address newAddr) public onlyDev { require(newAddr != address(0), "zero addr"); devAddr = newAddr; } [#176-179]
3. function lpTokenAddress() view public returns(address) { return address(lpToken); } [#193-195]
4. function mineAddress() view public returns(address) { return address(mine); } [#197-199]
5. function testMint() public onlyOwner { mine.mint(address(this), 1); } [#201-203]
6. function setInitRewards(uint256 reward) public onlyOwner { initReward = reward; } [#205-207]

Medium Severity Issues:

Function could be marked as external SWC-000:

The function definition of the following is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Affected lines:

1. function transfer(address recipient, uint256 amount) public override returns (bool) { require(msg.sender == owner() || !_lock, "Transfer is locking"); uint256 taxAmount = amount.mul(_taxRate).div(100); if (_taxWhitelist[msg.sender] == true) { taxAmount = 0; } uint256 transferAmount = amount.sub(taxAmount); require(balanceOf(msg.sender) >= amount, "insufficient balance."); super.transfer(recipient, transferAmount); if (taxAmount != 0) { super.transfer(_taxDestination, taxAmount); } return true; } [#20-35]
2. function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) { require(sender == owner() || !_lock, "TransferFrom is locking"); uint256 taxAmount = amount.mul(_taxRate).div(100); if (_taxWhitelist[sender] == true) { taxAmount = 0; } uint256 transferAmount = amount.sub(taxAmount); require(balanceOf(sender) >= amount, "insufficient balance."); super.transferFrom(sender, recipient, transferAmount); if (taxAmount != 0) { super.transferFrom(sender, _taxDestination, taxAmount); } return true; } [#37-50]
3. function cap() public view returns (uint256) { return _cap; } [#70-72]
4. function mint(address to, uint amount) public onlyMinter { _mint(to, amount); _moveDelegates(address(0), _delegates[to], amount); } [#74-77]
5. function burn(uint amount) public { require(amount > 0); require(balanceOf(msg.sender) >= amount); _burn(msg.sender, amount); } [#302-306]

Medium Severity Issues:

Function could be marked as external SWC-000:

The function definition of the following is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Affected lines:

1. function addMinter(address account) public onlyOwner { minters[account] = true; } [#308-310]
2. function removeMinter(address account) public onlyOwner { minters[account] = false; } [#312-314]
3. function setTaxer(address account) public onlyTaxer { _taxer = account; } [#326-328]
4. function setTaxRate(uint256 rate) public onlyTaxer { _taxRate = rate; } [#330-332]
5. function setTaxDestination(address account) public onlyTaxer { _taxDestination = account; } [#334-336]
6. function addToWhitelist(address account) public onlyTaxer { _taxWhitelist[account] = true; } [#338-340]
7. function removeFromWhitelist(address account) public onlyTaxer { _taxWhitelist[account] = false; } [#342-344]
8. function taxer() public view returns(address) { return _taxer; } [#346-348]
9. function taxDestination() public view returns(address) { return _taxDestination; } [#350-352]
10. function taxRate() public view returns(uint256) { return _taxRate; } [#354-356]
11. function isInWhitelist(address account) public view returns(bool) { return _taxWhitelist[account]; } [#358-360]
12. function unlock() public onlyOwner { _lock = false; } [#362-364]
13. function getLockStatus() view public returns(bool) { return _lock; } [#366-368]
14. function renounceOwnership() public virtual onlyOwner { emit OwnershipTransferred(_owner, address(0)); _owner = address(0); } [#54-56]

Medium Severity Issues:

Function could be marked as external SWC-000:

The function definition of the following is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Affected lines:

1. function transferOwnership(address newOwner) public virtual onlyOwner { require(newOwner != address(0), "Ownable: new owner is the zero address"); emit OwnershipTransferred(_owner, newOwner); _owner = newOwner; } [#63-67]
2. function symbol() public view returns (string memory) { return _symbol; } [#72-74]
3. function decimals() public view returns (uint8) { return _decimals; } [#89-91]
4. function allowance(address owner, address spender) public view virtual override returns (uint256) { return _allowances[owner][spender]; } [#123-125]
5. function approve(address spender, uint256 amount) public virtual override returns (bool) { _approve(_msgSender(), spender, amount); return true; } [#134-137]
6. function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) { _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue)); return true; } [#170-173]
7. function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) { _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero")); return true; } [#189-192]

Audit Team

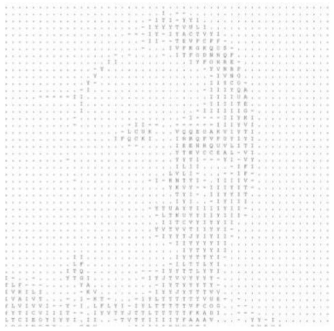
Team Lead: Eelko Neven

Eelko has been in the it/security space since 1991. His passion started when he was confronted with a formatted hard drive and no tools to undo it. At that point he started reading a lot of material on how computers work and how to make them work for others. After struggling for a few weeks he finally wrote his first HD data recovery program. Ever since then when he was faced with a challenge he just persisted until he had a solution.

This mindset helped him tremendously in the security space. He found several vulnerabilities in large corporation servers and notified these corporations in a responsible manner. Among those are Google, Twitter, General Electrics etc.

For the last 12 years he has been working as a professional security /code auditor and performed over 1500 security audits / code reviews, he also wrote a similar amount of reports.

He has extensive knowledge of the Solidity programming language and this is why he loves to do Defi and other smartcontract reviews.



Email:
info@coinbae.com

Disclaimer

Coinbae audit is not a security warranty, investment advice, or an endorsement of the Spacemine DeFi platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Spacemine DeFi team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.