QuillAudits

# Audit Report
July, 2024

For

# Table of Content

# Table of Content

# Executive Summary

**Project Name**

Weepad

**Overview**

Weepad smart contracts are a collection of ERC1155 NFT sale contract, staking, and factory and pool contracts. In the GameNFT contract, users purchase varieties of NFTs. The staking contracts

**Timeline**

31st May 2024 - 11th July 2024

**Updated Code Received**

26th July 2024

**Second Review**

26th July 2024 - 27th July 2024

**Method**

Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.

**Audit Scope**

This audit aimed to analyze the Weepad Codebase for quality, security, and correctness.

1. GamepadNFT.sol
2. Pool.sol
3. PoolFactory.sol
4. RoyaReserve.sol
5. RoyaToken.sol
6. NFTStakingLot.sol
7.StakingLot.sol
8. Authorizable.sol
9. Whitelist.sol

**Source Code**

https://github.com/weeweepad-tech/weeweepad-contract

**Commit Hash**

e0aca52f05dfe298f9a0fe329c1723a9a31ff3d3

**Blockchain**

EVM

**Fixed In**

Branch: final_audit
**https://github.com/weeweepad-tech/weeweepad-contract/tree/final_audit**

# Number of Security Issues per Severity

6
Issues Found
High
Medium
Low
Informational

| | High | Medium | Low | Informational |
|---|---|---|---|---|
| Open Issues | 0 | 0 | 0 | 0 |
| Acknowledged Issues | 0 | 0 | 0 | 0 |
| Partially Resolved Issues | 0 | 0 | 0 | 0 |
| Resolved Issues | **6** | **3** | **3** | **4** |

# Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ DoS with Block Gas Limit
- ✓ Transaction-Ordering Dependence
- ✓ Use of tx.origin
- ✓ Exception disorder
- ✓ Gasless send
- ✓ Balance equality
- ✓ Byte array
- ✓ Transfer forwards all gas
- ✓ ERC20 API violation
- ✓ Compiler version not fixed
- ✓ Redundant fallback function
- ✓ Send instead of transfer
- ✓ Style guide violation
- ✓ Unchecked external call
- ✓ Unchecked math
- ✓ Unsafe type inference
- ✓ Implicit visibility level

# Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

### Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

### Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

### Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

### Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

### Tools and Platforms used for Audit

Hardhat, Foundry.

## Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

### High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

### Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

### Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

### Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

## Types of Issues

### Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

### Resolved

These are the issues identified in the initial audit and have been successfully fixed.

### Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

### Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

# High Severity Issues

## 1. Entire funds will be lost on the GamepadNFT contract to the null address

### Path

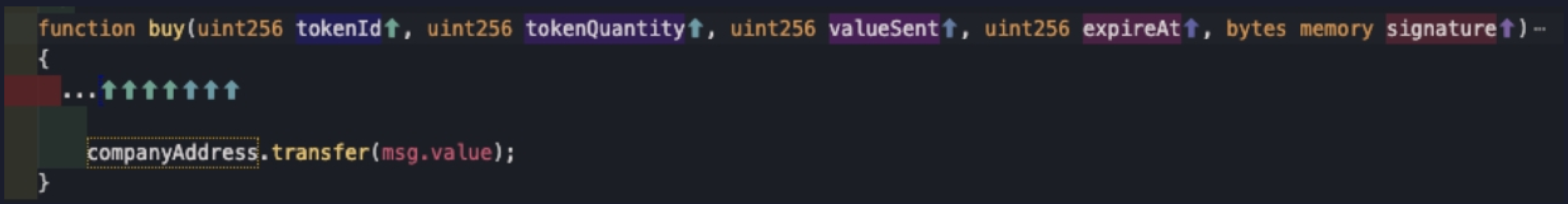
GamepadNFT.sol

### Function

companyAddress

### Description

This is because when nfts are purchased, entire funds are immediately sent to the companyAddress:



```
function buy(uint256 tokenId, uint256 tokenQuantity, uint256 valueSent, uint256 expireAt, bytes memory signature) ⋯
{
  ...

    companyAddress.transfer(msg.value);
}
```

The issue here is that the companyAddress was mistakenly uninitialized in the constructor due to a coding error and therefore set to address(0). Which means all funds will be burnt or lost (sent to address(0)).

### Recommendation

+ companyAddress = _companyAddress;
- _companyAddress = companyAddress;

### Status

**Resolved**

## 2. Signature replay attack

**Path**

GamepadNFT.sol

**Function**

buy()

**Description**

The buy function requires a signature operation getting signed by the signingkey om the backend. However, it allows for signature replay attacks as users can use the same signature multiple times.

**Recommendation**

Consider adding a nonce to track used nonce and ensure it is not used again.

**Status**

**Resolved**

**Reference**

https://dacian.me/signature-replay-attacks

### 3. Malicious users can extend the redeem period of stakers due to unguarded permission issue

**Path**

StakedRoya.sol

**Function**

stake()

**Description**

stake function is designed to take address and amount parameters. This implementation introduces the possibility of malicious users passing stakers address with 1 wei, and repeatedly calling this for the purpose of extending the redeem period of these stakers.

**Recommendation**

Consider implementing a mapping that allows for stakers to give approval to addresses that can stake on their behalf to prevent malicious stakers aiming to extend the redeem period of protocol users.

**Status**

**Resolved**

## 4. Removing an address will lead to unexpected behavior

### Path

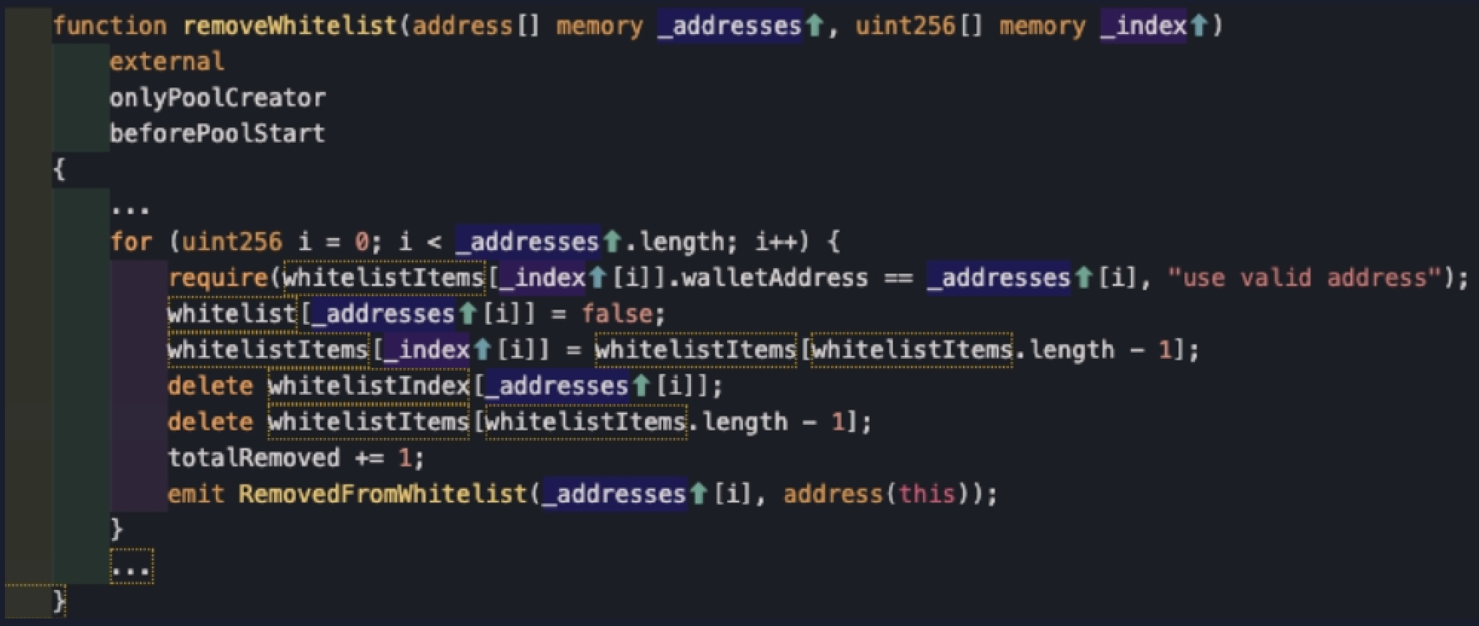
Whitelist.sol

### Function

removeWhitelist()

### Description

Removing an address from the whitelisted addresses will set the mapping of whitelist to false, change the element in the whitelistItems[index[I]] to the last element and then delete from the whitelistIndex and whitelistItems mapping. Not updating the whitelistIndex mapping before deleting will introduce an unexpected behavior.



```
function removeWhitelist(address[] memory _addresses, uint256[] memory _index)
    external
    onlyPoolCreator
    beforePoolStart
{
    ...
    for (uint256 i = 0; i < _addresses.length; i++) {
        require(whitelistItems[_index[i]].walletAddress == _addresses[i], "use valid address");
        whitelist[_addresses[i]] = false;
        whitelistItems[_index[i]] = whitelistItems[whitelistItems.length - 1];
        delete whitelistIndex[_addresses[i]];
        delete whitelistItems[whitelistItems.length - 1];
        totalRemoved += 1;
        emit RemovedFromWhitelist(_addresses[i], address(this));
    }
    ...
}
```

### Recommendation

like the whitelistItems, switch the last items in the whitelistIndex before deleting.

### Status

**Resolved**

## 5. Some addresses will be lost and not whitelisted when multiple addresses are to be added

### Path

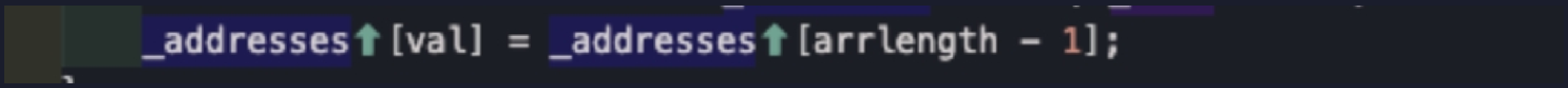
Whitelist.sol

### Function

addPublicRandom()

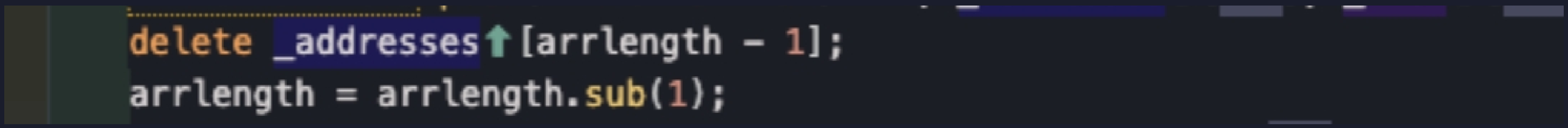
### Description

In addPublicRandom, it updates _addressess[val] to _addresses[arrLength - 1].



```
        _addresses[val] = _addresses[arrlength - 1];
```

However, _addresses[arrLength - 1] was already deleted before that line.



```
      delete _addresses[arrlength - 1];
      arrlength = arrlength.sub(1);
```

Therefore, some addresses will be lost and not get whitelisted. This will also at some point add the null address as a whitelisted address with a tier. Some addresses will be lost in the process of adding and will sometimes run into an arithmetic error due to the second loop (in a situation when provided address parameters are lesser than 32).

### Recommendation

Use an oracle to handle random generation in order to simplify the function. And more so, this function will not always be called so it is reasonable to pay for Chainlink vrf.

### Status

**Resolved**

### 6. RoyaToken holders are unable to reduce approval amount of approved spenders

**Path**

RoyaToken.sol

**Function**

approve()

**Description**

This is a custom token contract built without the integration of the standard ERC20 contract, like the Openzeppelin library. The approve function is designed to always increase the amount of tokens a spender can spend.This introduces an issue that makes it impossible to ever reduce the amount of a spender unless the spender has exhausted their approved amount.

**Recommendation**

Integrating a standard library will give token holders the flexibility to increase or decrease an approval amount.

**Status**

**Resolved**

## Medium Severity Issues

### 7. Impossible to invoke buy function due to logic error in _beforeTokenTransfer hook function

**Path**

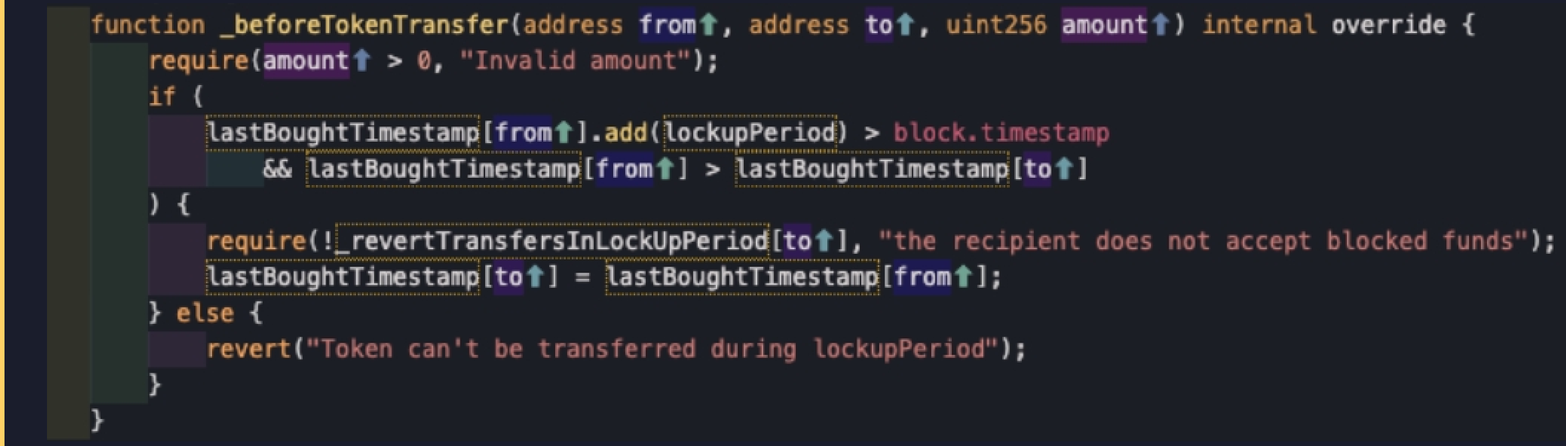
lots/StakingLot.sol
ots/NFTStakingLot.sol

**Function**

_beforeTokenTransfer

**Description**

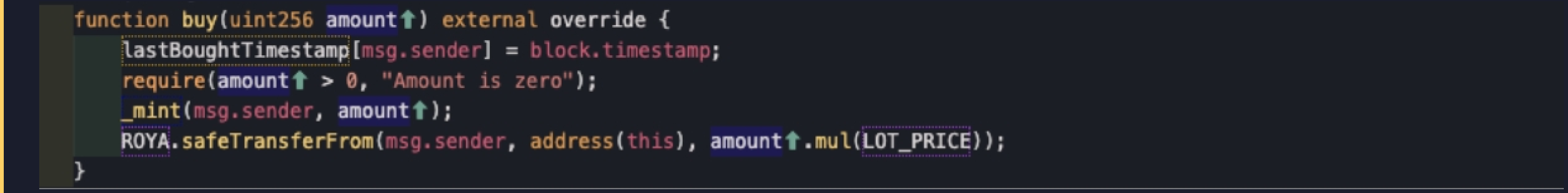
the buy function will revert anytime a user intends to get the StakingLot or NFTStakingLot token. There is a if-condition in the the _beforeTokenTransfer hook function that is ensuring that the sender lockup period has reached and that the lastBoughtTimestamp of the sender is greater than the receiver.



```
function _beforeTokenTransfer(address from, address to, uint256 amount) internal override {
    require(amount > 0, "Invalid amount");
    if (
        lastBoughtTimestamp[from].add(lockupPeriod) > block.timestamp
            && lastBoughtTimestamp[from] > lastBoughtTimestamp[to]
    ) {
        require(!_revertTransfersInLockUpPeriod[to], "the recipient does not accept blocked funds");
        lastBoughtTimestamp[to] = lastBoughtTimestamp[from];
    } else {
        revert("Token can't be transferred during lockupPeriod");
    }
}
```

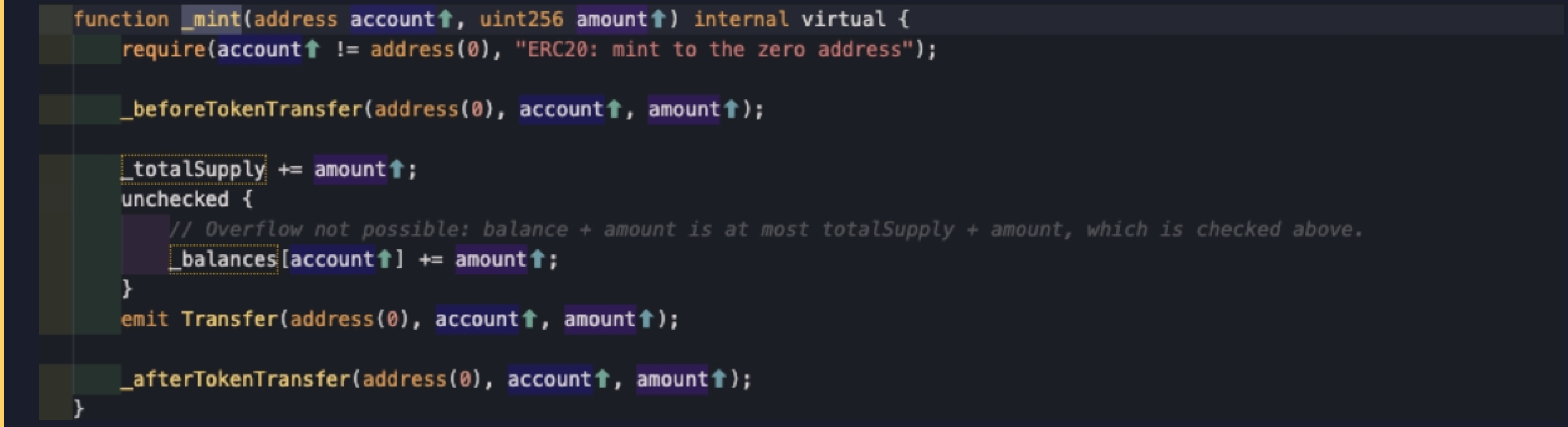
When users invoke the buy function, it runs until it enters the _mint function and then reverts due to the extended logic implementation on the _beforeTokenTransfer since this is called within the _mint function.

It is important to stress that the logic in the _beforeTokenTransfer is flawed. Since it expects that token transfer to be enabled after the lockup period is over, the if-condition is expected to be lastBoughtTimestamp[from].add(lockupPeriod) <= block.timestamp (check lockupFree modifier).



```
function buy(uint256 amount) external override {
    lastBoughtTimestamp[msg.sender] = block.timestamp;
    require(amount > 0, "Amount is zero");
    _mint(msg.sender, amount);
    ROYA.safeTransferFrom(msg.sender, address(this), amount.mul(LOT_PRICE));
}
```

At the level of the _mint, the sender will be the address(0), and the buyer address as the receiver but because the null address lastBoughtTimestamp is 0, the function will revert.



```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    unchecked {
        // Overflow not possible: balance + amount is at most totalSupply + amount, which is checked above.
        _balances[account] += amount;
    }
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}
```

### Recommendation

Redesign the contract to track for when users have purchased the tokens already before the implementation in the _beforeTokenTransfer hook takes effect of preventing transfer of the tokens during lockup period.

### Status

**Resolved**

### 8. Centralization Risks - It is possible for the pool creator to remove the entire sale token from the contract

**Path**

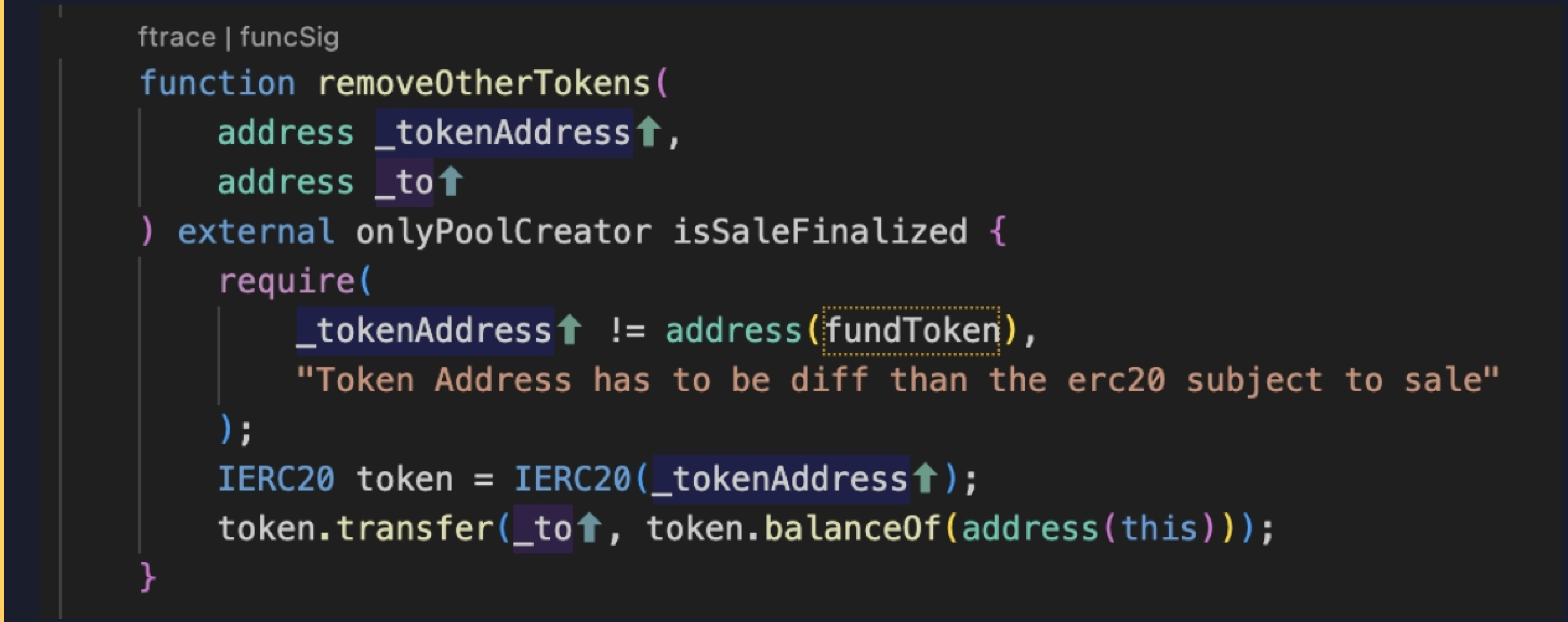
Pool.sol

**Function**

removeOtherTokens()

**Description**

In the case where isNativeToken == false after people funds are already in the contract and sale has ended, even without minimum raise not check before removing sale token. This is possible due to removeOtherTokens() function preventing token from being fundToken but doesn't prevent token from being saleToken in the case where isNativeToken == false, making rugpull a possibility:



```
ftrace | funcSig
function removeOtherTokens(
    address _tokenAddress,
    address _to
) external onlyPoolCreator isSaleFinalized {
    require(
        _tokenAddress != address(fundToken),
        "Token Address has to be diff than the erc20 subject to sale"
    );
    IERC20 token = IERC20(_tokenAddress);
    token.transfer(_to, token.balanceOf(address(this)));
}
```

**Recommendation**

Pool creator should not be able to remove the sale token with minimum raise already passed and people funds staked.

**Status**

**Resolved**

### 9. It is possible for pool creator to prevent the protocol factory owner from setting fees on the pool

**Path**

Pool.sol

**Function**

addPrivate

**Description**

The changeFeeInfoOnPool() call setFeeInfo() on the pool which requires pool not started yet, however their is not minimum cooldown time before pool starts meaning pool creator can set pool to start 2 seconds immediately pool gets created preventing protocol factory from setting fee.

**Recommendation**

Have a minimum cooldown period between when the pool is created and when it starts, to give pool factory owner enough time to set necessary configuration of the created pool.

**Status**

**Resolved**

## Low Severity Issues

### 10. Missing null address check

**Path**

Pool.sol

**Function**

setNewOwner, setFeeInfo

**Description**

These functions receive an address parameter in order to either modify the pool creator or the fee addresses but both fail to check that these addresses are non-zero addresses.

**Recommendation**

Add a null check in both functions.

**Status**

**Resolved**

### 11. Unsafe ERC20 Operations should not be used

**Path**

Pool.sol, GamepadNFT

**Description**

ERC20 functions may not behave as expected. For example: return values are not always meaningful. It is recommended to use OpenZeppelin's SafeERC20 library.

**Recommendation**

It is recommended to use OpenZeppelin's SafeERC20 library.

**Status**

**Resolved**

## 12. No way to remove nativeTokens from the contract, if isNativeToken == false

**Path**

Pool.sol

**Function**

removeOtherTokens()

**Description**

removeOtherTokens() allows creator to remove other token from the contract but doesn’t handle native token in the case where isNativeToken == false

**Recommendation**

Add a way to handle native tokens as well.

**Status**

**Resolved**

## Informational Issues

### 13. Remove unused state variables and modifier

**Path**

NFTStakingLot.sol
StakingLot.sol

**Function**

discountedLots

**Modifier**

lockupFreeNFT

**Description**

There are some variables and modifiers declared within the contracts but were never used.

**Recommendation**

Remove unused variables and modifiers.

**Status**

**Resolved**

### 14. Comparison to boolean constant

**Path**

Whitelist.sol

**Function**

addPrivate

**Description**

To detect when addresses have been whitelisted, it is compared that whitelist[addresses[I]] is not equal to the true constant value.

**Recommendation**

There is no need for comparison since the whitelist[addresses[I]] mapping returns a true or false. This could be directly used.

**Status**

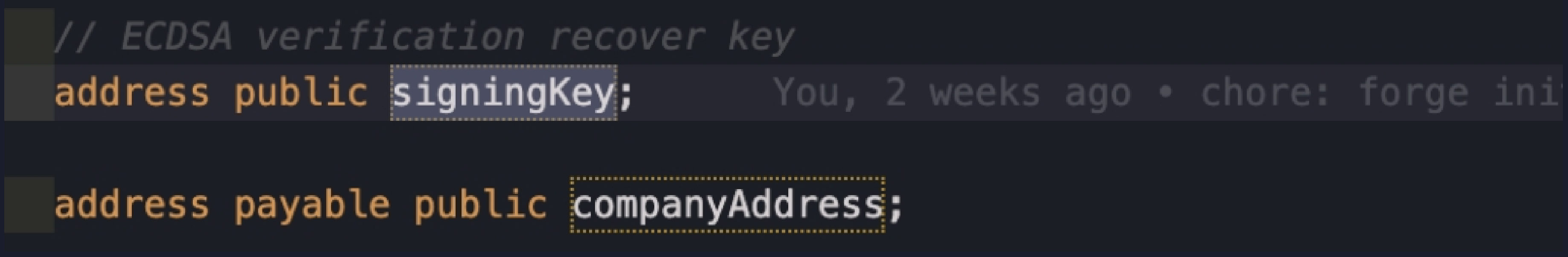
**Resolved**

## 15. Variables only set at the constructor should be made immutable

### Path

GamepadNFT

### Function



```
// ECDSA verification recover key
address public signingKey;        You, 2 weeks ago • chore: forge ini

address payable public companyAddress;
```

### Description

These variables are only set at the constructor level within the contract. This will create a storage slot for each of the variables.

### Recommendation

Immutable variables do not take up a storage slot.

### Status

**Resolved**

## 16. Double check will cause users to pay for extra gas

### Path

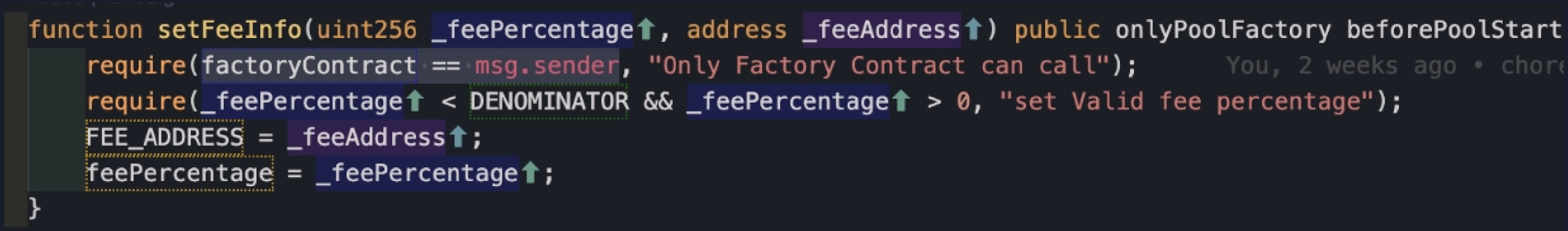
Pool.sol
StakingLot.sol
NFTStakingLot.sol

### Function

setFeeInfo(), buy()

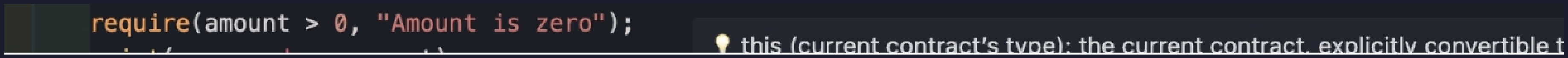
### Description

There is a double check happening in these functions which will cause the users to pay for extra gas for that check operation. In the case of the setFeeInfo function, there is the onlyPoolFactory modifier and there is also the check in function that requires that the caller is the factory contract.



```
function setFeeInfo(uint256 _feePercentage, address _feeAddress) public onlyPoolFactory beforePoolStart
    require(factoryContract == msg.sender, "Only Factory Contract can call");      You, 2 weeks ago • chore
    require(_feePercentage < DENOMINATOR && _feePercentage > 0, "set Valid fee percentage");
    FEE_ADDRESS = _feeAddress;
    feePercentage = _feePercentage;
}
```

Likewise, for the buy function, there is a check ensuring that amount is greater than 0. This check appears also in the _beforeTokenTransfer function.



```
    require(amount > 0, "Amount is zero");
```

this (current contract's type): the current contract, explicitly convertible t

### Recommendation

Implement functions to perform one single check operation to save gas.

### Status

**Resolved**

# Functional Tests Cases

**Some of the tests performed are mentioned below:**

- ✓ Should add addresses with the addPrivate function by pool creator
- ✓ Should add addresses with addPublicRandom while observing the randomized tier
- ✓ Should remove addresses from whitelist with accurate state update
- ✓ Should allow only pool factory set limitations for each tier
- ✓ Should allow the pool creator to pause and unpause the pool contract.

# Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

# Closing Summary

In this report, we have considered the security of the Weepad codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

# Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Weepad smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Weepad smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Weepad to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

# About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

**1000+**
Audits Completed

**$30B**
Secured

**1M+**
Lines of Code Audited

## Follow Our Journey

# Audit Report
# July, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com