QuillAudits

Audit Report
March, 2023

For

COMEARTH™

Table of Content

Executive Summary

Project Name	COMEARTH by NFTICALLY
Overview	COMEARTH contracts allow for the acquisition of land parcels in their metaverse in form of an nft representation. These parcels come in various sizes and can be grouped as well. The LandSourceWallet holds all the Land parcels that can be purchased by any user via the LandSeller contract. It integrates the standard Openzeppelin library to handle its ownership, mint, and burn features as well as the custom functionality implemented by their backend service to aid transactions.
Timeline	March 13, 2023 - March 24, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze the following COMEARTH contracts in the codebase for quality, security, and correctness. - *LandParcel.sol* - *LandSeller.sol* - *Executable.sol* These contracts are on the "dev" branch with commit hash 60bef1fcb3577080311287 52d49670494667b89b
Fixed in	*https://github.com/NFTically/comearth-smart-contracts/commit/cdafe6a86c4e921722928a95d80ae3677d35af25*

5
Issues Found

🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	0	3

Types of Severities

High

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

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

Low-level severity issues can cause minor impact and or 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 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.

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour 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 analysed 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

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 analysed, 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 behaviour 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

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - LandParcel.sol

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

A1. Gas optimization

The _beforeTokenTransfer function calls super._beforeTokenTransfer before checking whether or not the tokenId to be transferred is in a group. This means that more lines of code would be executed before the function call would fail if the token is already part of a group which means higher transaction costs for failed transactions.

Recommendation

Adjust the beforeTokenTransfer function to perform the require check before calling (super.)_beforeTokenTransfer in ERC721Mintable.

Status

Resolved

B. Contract - LandSeller.sol

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

B1. Slippage Possibility

The LandSeller contract contains hardcoded values for calculations as well as addresses. A significant issue here is in the declaration of values for price conversion. Due to the volatile nature of the markets, the expected rates that were calculated with those values could easily fall out of tune. Users could game the system's economics by scouting for and using a payment token that is depegged from the expected rates.

Recommendation

It is advised to use an oracle for the required time and price valuation in this contract.

NFTICALLY Team Comment

This contract is not expected to be used for long. It would be deprecated shortly after launch to give way to an exchange contract instead. In any case of market fluctuations, our backend service runs automatic checks and can update the conversion factors using the setPriceConversionFactors() function.

Status

Acknowledged

B2. Unused imports

The imported SafeMath library is unused and unnecessary. Since Solidity ^0.8.17 has inbuilt overflow and underflow checks, it is an unnecessary redundancy.

Recommendation

Remove the unused library.

Status

Resolved

C. Contract - Executable.sol

High Severity Issues

No issues were found

Medium Severity Issues

C1. Input validation

The team intends to add only approved contracts as executors but does not include address type checks in the constructor() and setExecutor()

Recommendation

Include the required address checks.

Status

Resolved

Low Severity Issues

No issues were found

Informational Issues

C2. Unused imports

The imported Ownable library is unused and not necessary.

Recommendation

Remove the unused library.

Status

Resolved

General Recommendation

With the manual setting of price conversion that influences the price to be paid by prospective users, it is possible for the price to be manipulated in instances where there is a delay in the update from their backend service. The COMEARTH off-chain backend service is outside the scope of this audit. Its interaction with contracts has not been tested. With the present design architecture adopted by Comearth, it grants rights to platform owners to regulate the prices of these land parcels with the setParcelPricesInUSD and setPriceConversionFactor functions. While the client's comments are captured above on why this design was adopted, we recommend the use of price oracles to aid fetch the latest data about tokens to aid in efficient price calculations.

Functional Testing

Some of the tests performed are mentioned below:

LandSeller.sol

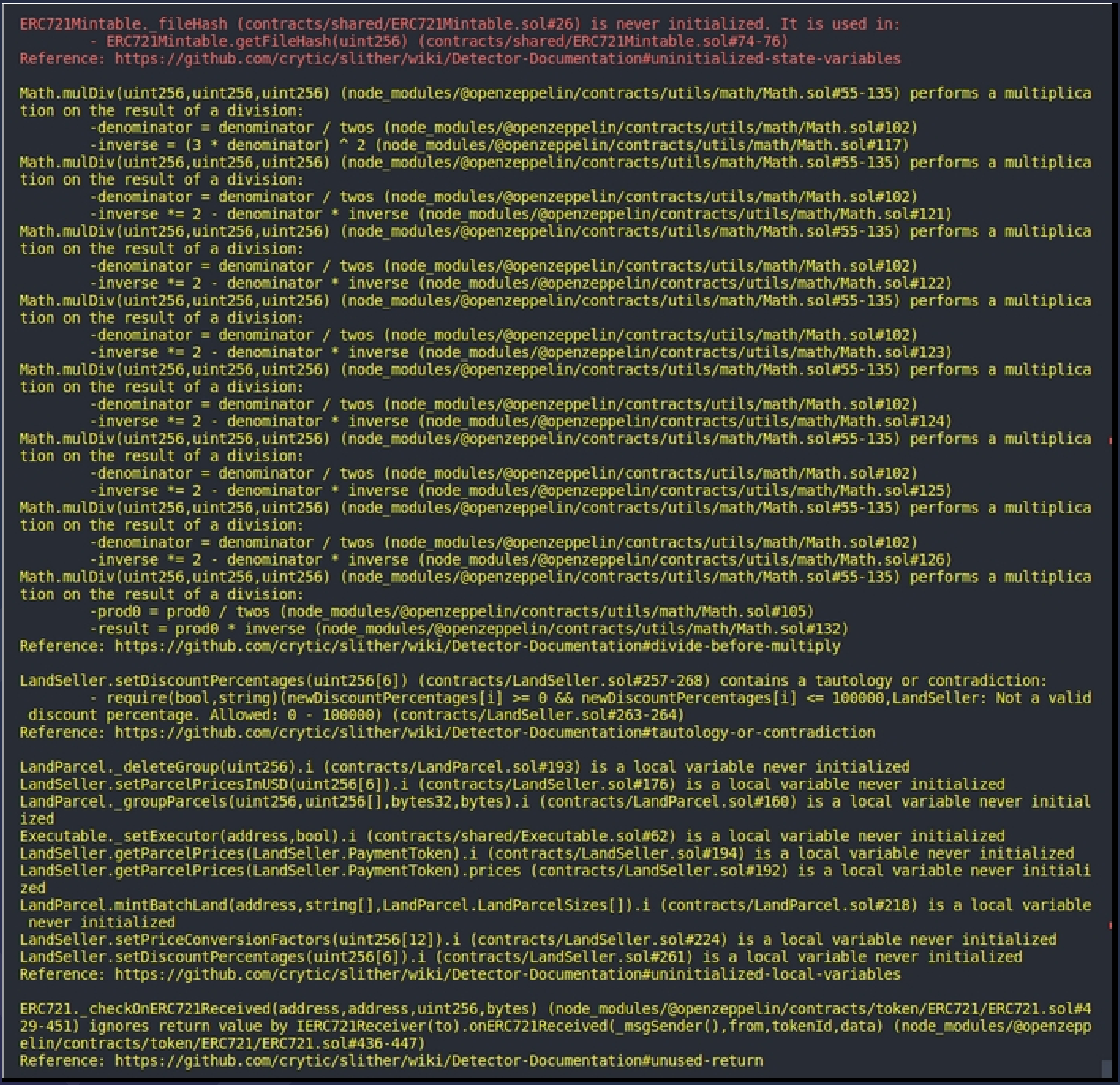
- ✓ Should revert if LandSourceWallet do not give approval on tokens passed to buyLand
- ✓ Should ascertain the pricePayable by an user depending on the land parcels sizes in relation to initialized prices.
- ✓ Should revert when privileged functions are called by unauthorized users.

LandParcel.sol

- ✓ Should check the returned address retrieved by recoverAddress function.
- ✓ Should check the signature passed when creating a new group.
- ✓ Should revert when the token is in a group during a transfer.
- ✓ Should successfully delete a parcel from a group.
- ✓ Should revert when non-owners intend to create a group with tokens Id which are not theirs.
- ✓ Should revert when mintLand and mintBatchLand are called by unauthorized users.
- ✓ Should allow owners to successfully burn tokens.
- ✓ Should disallow burning of tokens Id that is part of a group.

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.



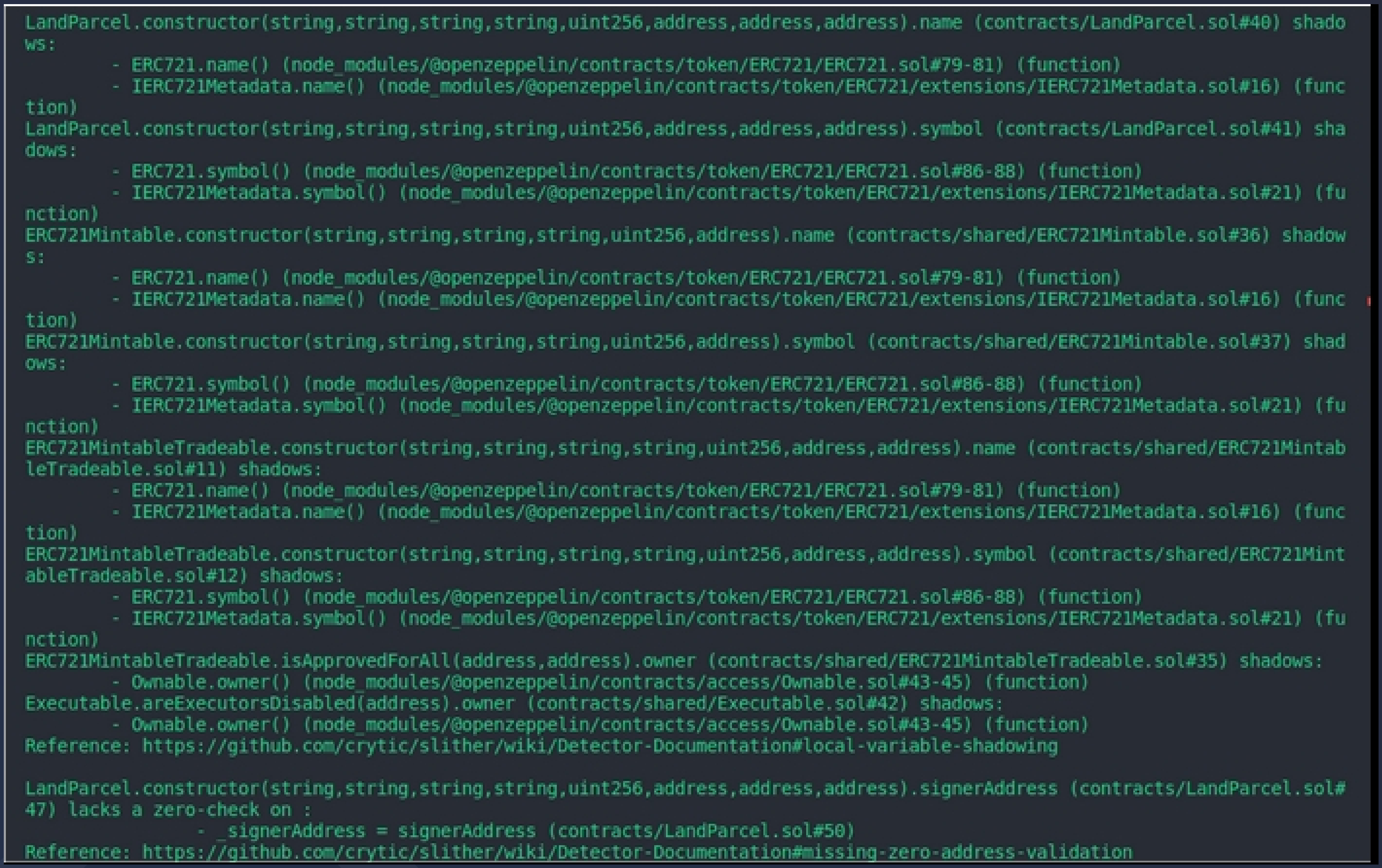
```
ERC721Mintable._fileHash (contracts/shared/ERC721Mintable.sol#26) is never initialized. It is used in:
        - ERC721Mintable.getFileHash(uint256) (contracts/shared/ERC721Mintable.sol#74-76)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplica
tion on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplica
tion on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplica
tion on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplica
tion on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplica
tion on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplica
tion on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplica
tion on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplica
tion on the result of a division:
        -prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
        -result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

LandSeller.setDiscountPercentages(uint256[6]) (contracts/LandSeller.sol#257-268) contains a tautology or contradiction:
        - require(bool,string)(newDiscountPercentages[i] >= 0 && newDiscountPercentages[i] <= 100000,LandSeller: Not a valid
 discount percentage. Allowed: 0 - 100000) (contracts/LandSeller.sol#263-264)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction

LandParcel._deleteGroup(uint256).i (contracts/LandParcel.sol#193) is a local variable never initialized
LandSeller.setParcelPricesInUSD(uint256[6]).i (contracts/LandSeller.sol#176) is a local variable never initialized
LandParcel._groupParcels(uint256,uint256[],bytes32,bytes).i (contracts/LandParcel.sol#160) is a local variable never initial
ized
Executable._setExecutor(address,bool).i (contracts/shared/Executable.sol#62) is a local variable never initialized
LandSeller.getParcelPrices(LandSeller.PaymentToken).i (contracts/LandSeller.sol#194) is a local variable never initialized
LandSeller.getParcelPrices(LandSeller.PaymentToken).prices (contracts/LandSeller.sol#192) is a local variable never initiali
zed
LandParcel.mintBatchLand(address,string[],LandParcel.LandParcelSizes[]).i (contracts/LandParcel.sol#218) is a local variable
 never initialized
LandSeller.setPriceConversionFactors(uint256[12]).i (contracts/LandSeller.sol#224) is a local variable never initialized
LandSeller.setDiscountPercentages(uint256[6]).i (contracts/LandSeller.sol#261) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4
29-451) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzepp
elin/contracts/token/ERC721/ERC721.sol#436-447)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```


```
LandParcel.constructor(string,string,string,string,uint256,address,address,address).name (contracts/LandParcel.sol#40) shado
ws:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#79-81) (function)
        - IERC721Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#16) (func
tion)
LandParcel.constructor(string,string,string,string,uint256,address,address,address).symbol (contracts/LandParcel.sol#41) sha
dows:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#86-88) (function)
        - IERC721Metadata.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#21) (fu
nction)
ERC721Mintable.constructor(string,string,string,string,uint256,address).name (contracts/shared/ERC721Mintable.sol#36) shadow
s:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#79-81) (function)
        - IERC721Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#16) (func
tion)
ERC721Mintable.constructor(string,string,string,string,uint256,address).symbol (contracts/shared/ERC721Mintable.sol#37) shad
ows:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#86-88) (function)
        - IERC721Metadata.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#21) (fu
nction)
ERC721MintableTradeable.constructor(string,string,string,string,uint256,address,address).name (contracts/shared/ERC721Mintab
leTradeable.sol#11) shadows:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#79-81) (function)
        - IERC721Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#16) (func
tion)
ERC721MintableTradeable.constructor(string,string,string,string,uint256,address,address).symbol (contracts/shared/ERC721Mint
ableTradeable.sol#12) shadows:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#86-88) (function)
        - IERC721Metadata.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#21) (fu
nction)
ERC721MintableTradeable.isApprovedForAll(address,address).owner (contracts/shared/ERC721MintableTradeable.sol#35) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
Executable.areExecutorsDisabled(address).owner (contracts/shared/Executable.sol#42) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

LandParcel.constructor(string,string,string,string,uint256,address,address,address).signerAddress (contracts/LandParcel.sol#
47) lacks a zero-check on :
                - _signerAddress = signerAddress (contracts/LandParcel.sol#50)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

Closing Summary

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

Some issues with low, and informational severity were found during the course of the audit and NFTICALLY Team Resolved all Issues and Acknowledged one issue.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the COMEARTH 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 COMEARTH Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
March, 2023

For

COMEARTH™

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com