SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

For

COMEARTH

Audited By
Satyabrata Dash & Aliasgar Kapadia

Report Prepared By
Aliasgar Kapadia

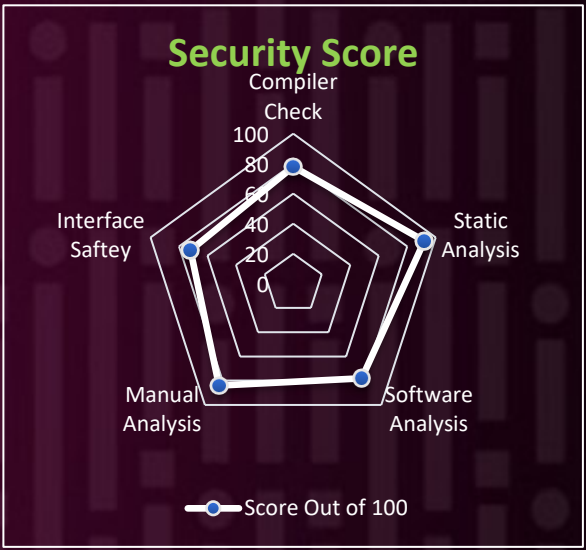
Table of Contents

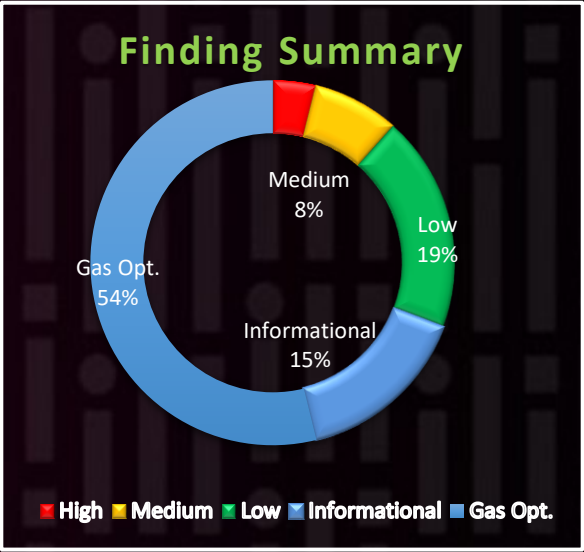
Disclaimer

The Secureverse team examined this smart contract in accordance with industry best practices. We made every effort to secure the code and provide this report. audits done by smart contract auditors and automated algorithms; however, it is crucial to remember that you should not rely entirely on this report. The smart contract may have flaws that allow for hacking. As a result, the audit cannot ensure the explicit security of the audited smart contracts. The Secureverse and its audit report do not encourage readers to consider them as providing any project-related financial or legal advice.

Executive Summary

Project Name	Comearth
Project Type	NFT & DeFi
Audit Scope	Check Security and code quality
Audit Method	Static and Manual
Audit Timeline	17TH Feb., 2023 to 27th Feb., 2023
Source Code	https://github.com/NFTically/comearth-smart-contracts





Issue Tracking Table					
	High	Medium	Low	Informational	Gas Optimization
Open Issues	-	-	-	-	-
Acknowledged Issues	-	1	3	3	9
Resolved Issues	1	1	2	1	5

Types of Severities

- High: The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- Medium: The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
- Low: The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
- Informational: The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth.

Types of Issues

- Open: Security vulnerabilities identified that must be resolved and are currently unresolved.
- Acknowledged: The way in which it is being used in the project makes it unnecessary to address the vulnerabilities. This means that the way it has been acknowledged has no effect on its security.
- Resolved: These are the issues identified in the initial audit and have been successfully fixed.

n

Checked Vulnerabilities

- Re-entrancy
- Access control
- Denial of service
- Timestamp Dependence
- Integer overflow/Underflow
- Transaction Order Dependency
- Requirement Violation
- Functions Visibility Check
- Mathematical calculations
- Dangerous strict equalities
- Unchecked Return values
- Hard coded information
- Safe Ether Transfer
- Gas Consumption
- Incorrect Inheritance Order
- Centralization
- Unsafe external calls
- Business logic and specification
- Input validation
- Incorrect Modifier
- Missing events
- Assembly usage
- ERC777 hooks
- Token handling

Methods

Audit at Secureverse is performed by the experts and they make sure that audited project must comply with the industry security standards.

Secureverse audit methodology includes following key:

- In depth review of the white paper
- In depth analysis of project and code documentation.
- Checking the industry standards used in Code/Project.
- Checking and understanding Core Functionality of the Code.
- Comparing the code with documentation.
- Static analysis of the code.
- Manual analysis of the code.
- Gas Optimization and Function Testing.
- Verification of the overall audit.
- Report writing.

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

- Static Analysis

Static analysis has been done by using the open source and state of the art automatic smart contract vulnerability scanning tools.

- Manual Analysis

Manual analysis is done by our smart contract auditors' team by performing in depth analysis of the smart contract and identify potential vulnerabilities. Auditor also review and verify all the static analysis results to prevent the false positives identified by automated tools.

- Gas Consumption and Function Testing

Function testing done by auditors by manually writing customized test cases for the smart contract to verify the intended behavior as per code and documentation. Gas Optimization done by reviews potential gas consumption by contract in production.

- Tools and Platforms used for Audit

  - Remix IDE
  - Hardhat
  - Mythril
  - Truffle Team
  - Solhint
  - Solidityscan.com
  - Slither
  - Consensys Surya
  - Open Zeppelin Code Analyzer
  - Manticore

Findings

High Severity Issues:

Vulnerability:	**Absence of access control on `updateGroup()`, So that's why anyone can delete others `_landGroups`**
Reference:	File: LandParcel.sol; Line Number: 130-144, 170-192
Description:	`updateGroup()` exists for simply to update parcel in existing group. It does above by two steps. it deletes the existing group via `_deleteGroup()` and assigns new landParcelIds to the group via `_groupParcels()`. In first step Function calls `_deleteGroup(groupId)`. It first set all parcel/token id to 0 making them group free, and then delete landGroups. Here point is it(_deleteGroup()) or its parent(updateGroup()) never check caller is actual owner of parcelId or caller has approval for Ungrouping. Here bug is any user can ungroup other user by passing there groupId and other parameter and use their groupId to create own group.

Remediation:	There already a function `deleteGroup()` present in code base which makes access control validation, so to implement that function first make it public, and inside `updateGroup` instead of calling `_deleteGroup()` call `deleteGroup()`
	`function deleteGroup(uint256 groupId) external virtual {`
	`require(isGroupExists(groupId), "LandParcel: Land group does not exists");`
	`require(getGroupOwner(groupId) == _msgSender() || isApprovedForAll(getGroupOwner(groupId), _msgSender()), "LandParcel: Caller is not approved to ungroup`
	`_deleteGroup(groupId);`
	`}`

Status:	Resolved

Medium Severity Issues:

Vulnerability:	Minted NFT Could Get Stuck
Reference:	File: ERC721Mintable.sol; Line Number: 119
Description:	In contract file ERC721Mintable.sol, the internal function `_mintInternal()` simply mints Nft to `address to`. But the problem with _mint here is that it does not check receiver (here _to address) implemented ERC721Receivable Interface or not, if not then minted token get stucked.
Remediation:	Here best thing to use _safeMint() instead of _mint(). The safeMint function calls mint and checks if the receiver is a smart contract and implements the ERC721Receivable interface.
Status:	Resolved

```
function _safeMint(address to, uint256 tokenId, bytes memory data)
internal virtual {
    _mint(to, tokenId);
    require(
        _checkOnERC721Received(address(0), to, tokenId, data),
        "ERC721: transfer to non ERC721Receiver implementer"
    );
}
```

Vulnerability:	A Malicious Executor Can Block Other Executors and Take Control of All Executing Power.
Reference:	File: Executable.sol; Line Number: 45-69
Description:	Here Code Flow Works Like, First Executor set inside constructor of Executable.sol contract. Then other Executors will added to contract via ` setExecutor()`.Here point is that only a previous active executor only able to add new executor and can change any other executor status(To active or inactive). So simply if a Malicious Executor get added to contract so he can deactivate other Executors of that contract via calling `setExecutor(target_executor, false)` and deactivate them one by one. At last, he will be the only Executor remain in that contract And Malicious Executor can also Overpopulate `_allExecutors[]` with so many addresses so that adding new valid executors will get more expensive.
Remediation:	Owner Should have control on `_setExecutor()`. So that even something went wrong owner can irradicate Malicious actor, and set new trusted One.
Status:	Acknowledged

Low Severity Issues:

Vulnerability:	Old Solidity Compiler Used	
Reference:	File	Line Number
	ERC721Mintable.sol	3
	Executable.sol	3
	ERC721MintableTradeable.sol	3
	LandParcel.sol	3
	LandSeller.sol	3
	LandSwapper.sol	3
Status:	Resolved	

Vulnerability:	There Must Be Zero Address Check While Setting Important Wallet Addresses	
Reference:	File	Line Number
	ERC721Mintable.sol	44
	LandSeller.sol	68
Status:	Relsoved	

Vulnerability:	`abi.encodePacked()` Should Not Be Used With Dynamic Types When Passing The Result To A Hash Function Like `keccak256()`	
Reference:	File	Line Number
	LandParcel.sol	139
Description:	`abi.encodePacked()` Should Not Be Used With Dynamic Types When Passing The Result To A Hash Function Like `keccak256()`	
Remediation:	Use `abi.encode()` instead which will pad items to 32bytes, which will prevent hash collision.	
Status:	Acknowledged	

Vulnerability:	For Transfering Native Token Instead Of Using `transfer()` Try To Use `call()`	
Reference:	File	Line Number
	LandSwapper.sol	156
	LandSeller.sol	100, 268
Status:	Acknowledged	

Vulnerability:	buyLand() try to return excess send Native Token back to buyer	
Reference:	File	Line Number
	LandSeller.sol	103-106
Description:	If Buyer is a Contract and It doesnot implemented any fallback()/receive() then below code will failed to transfer excess Matic(Native Token).	
Status:	Acknowledged	

```
// transfer access MATIC (native token), if any
        if (paymentToken == PaymentToken.MATIC && msg.value > 
payablePrice) {
            payable(_msgSender()).transfer(msg.value - 
payablePrice);
        }
```

Informational Issues:

Vulnerability:	Unused Function Parameter Present	
Reference:	File	Line Number
	ERC721Mintable.sol	152
Description:	royaltyInfo() has one unused parameter `tokenId`	
Status:	Acknowledged	

Vulnerability:	require()/revert() Should Have A Descriptive Reason String	
Reference:	File	Line Number
	LandParcel.sol	268
Status:	Acknowledged	

Vulnerability:	Event Is Missing Indexed Field	
Reference:	File	Line Number
	LandSeller.sol	63
Status:	Resolved	

Vulnerability:	Use Of ECRECOVER is Succeptible to Signature Malleability	
Reference:	File	Line Number
	LandParcel.sol	259
Description:	The built in EVM precompile ecrecover is susceptible to signature malleability which could lead to replay attack.	
Remediation:	Consider using Openzeppelin's ECDSA library which prevents Malleability instead of built in function.	
Status:	Acknowledged	

Gas Optimization:

Vulnerability:	Using Bools for Storage Incurs Overhead
Reference:	File: ERC721Mintable.sol — Line Number: 29, 33 File: Executable.sol — Line Number: 10, 16 File: LandParcel.sol — Line Number: 17
Description:	Booleans are more expensive than uint256 because each write operation emits an extra SLOAD to first read the slot's contents, replace the bits taken up by the Boolean, and then write back. This is the compiler's defense against contract upgrades and pointer aliasing, and it cannot be disabled.
Remediation:	Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess(100 gas) for the extra SLOAD, and to avoid Gsset (20000 gas) when changing from `false` to `true`, after having been `true` in the past.
Status:	Acknowledged

Vulnerability:	Use of Long Error Messages In require() Statement
Reference:	File: ERC721Mintable.sol — Line Number: 58, 117, 143 File: Executable.sol — Line Number: 47 File: ERC721MintableTradeable.sol — Line Number: 23 File: LandParcel.sol — Line Number: 91, 117, 132, 134, 150, 152, 159, 161, 172, 175, 206, 219, 224, 229, 234, 243 File: LandSeller.sol — Line Number: 83, 91, 144, 160, 184, 199, 220, 258 File: LandSwapper.sol — Line Number: 60, 62, 65, 68, 70, 81, 84, 103, 119, 133
Remediation:	Should use error()
Status:	Acknowledged

Vulnerability:	Instead of `public`, Some Functions Could Be `external`
Reference:	File: ERC721Mintable.sol — Line Number: 70, 74, 78, 82, 86, 95, 100, 105, 142 File: Executable.sol — Line Number: 30, 35, 40, 45 File: LandParcel.sol — Line Number: 74, 79, 84, 89, 97, 196, 204
Remediation:	Those Function which are not called inside contract can make as `external` instead of `public`
Status:	Resolved

Vulnerability:	Functions Those Will Revert When Called By Normal User Should Marked As `payable`
Reference:	File: ERC721Mintable.sol — Line Number: 78, 86 File: LandSeller.sol — Line Number: 119, 142, 251, 266, 273 File: LandParcel.sol — Line Number: 60, 69 File: LandSwapper.sol — Line Number: 161, 154, 131, 117, 100
Status:	Acknowledged

Vulnerability:	Instead Of `memory`, `calldata` Could Used
Reference:	File: ERC721Mintable.sol — Line Number: 105 File: LandParcel.sol — Line Number: 115 File: LandSeller.sol — Line Number: 216, 233
Remediation:	If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`
Status:	Resolved

Vulnerability:	Assigning Solidity Variables with Their Default Value Costs Extra Gas	
Reference:	File	Line Number
	ERC721Mintable.sol	33, 107
	Executable.sol	60
	LandParcel.sol	154, 185, 208
	LandSeller.sol	170, 188, 218, 255
	LandSwapper.sol	75
Status:	Acknowledged	

Vulnerability:	Variables Should Be Cached in Memory and Then Further Use It	
Reference:	File	Line Number
	ERC721Mintable.sol	107
	Executable.sol	60
	LandParcel.sol	154,185,208
	LandSeller.sol	170, 295-380
Status:	Resolved	

Vulnerability:	Operation `i++` Costs More Gas Than `++i` Same For Subtractions As Well	
Reference:	File	Line Number
	Executable.sol	60
	LandParcel.sol	154,185,208
	LandSeller.sol	170, 188, 218, 255
	LandSwapper.sol	75
Status:	Resolved	

Vulnerability:	Arithmetic Operations Which Will Not Overflow or Underflow Should Marked `unchecked`	
Reference:	File	Line Number
	LandParcel.sol	61
	LandSeller.sol	144, 160
	LandSwapper.sol	133, 119
Status:	Acknowledged	

Vulnerability:	Zero Address Checks Should Preform Through `assembly` Will More Cost Effective `
Reference:	File: LandParcel.sol, Line Number: 61 File: LandSeller.sol, Line Number: 144, 160 File: LandSwapper.sol, Line Number: 133, 119
Remediation:	Save 6 gas per instance by using assembly to check for address(0) `assembly { if iszero(_addr) { mstore(0x00, "zero address") revert(0x00, 0x20) }}`
Status:	Acknowledged

Vulnerability:	Use `selfbalance()` Instead of address(this).balance
Reference:	File: LandSeller.sol, Line Number: 268 File: LandSwapper.sol, Line Number: 156
Remediation:	You can use selfbalance() instead of address(this).balance when getting your contract's balance of ETH to save gas. Additionally, you can use balance(address) instead of address.balance() when getting an external contract's balance of ETH. Saves 15 gas when checking internal balance, 6 for external
Status:	Acknowledged

Vulnerability:	Making CONSTANT Variables `private` Will Save Gas During Deployment
Reference:	File: LandSeller.sol, Line Number: 16, 17 File: LandSwapper.sol, Line Number: 16, 18
Status:	Acknowledged

Vulnerability:	Splitting `require()` Statements That Use &&, Can Save Gas	
Reference:	File	Line Number
	LandSeller.sol	82, 257
Status:	Acknowledged	

Vulnerability:	Instead of ` > 0`, ` != 0` Is More Gas Efficient	
Reference:	File	Line Number
	LandSeller.sol	99, 95, 220
Status:	Resolved	

Automated Testing

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 audit, we examined the COMEARTH's smart contract with our framework, and we discovered several medium, low, and informational flaws in the smart contract. We have included solutions and recommendations in the audit report to improve the quality and security posture of the code. All of the findings and solutions have been acknowledged by the project team. In summary, we find that the codebase with the latest version greatly improved on the initial version. We believe that the overall level of security provided by the codebase in its current state is reasonable, so we have marked it as **Secure** and the Customer's smart contract has the following score: 8

About Secureverse

Secureverse is the Singapore and India based emerging Web3 Security solution provider. We at Secureverse provides the Smart Contract audit, Blockchain infrastructure Penetration testing and the Cryptocurrency forensic services with very affordable prices.

To Know More

Twitter: https://twitter.com/secureverse

LinkedIn: https://www.linkedin.com/company/secureverse/

Telegram: https://t.me/secureverse

Email Address: http://secureverse@protonmail.com/