QuillAudits

Audit Report November, 2022

For

OPEN Therapeutics

Table of Content

Table of Content

Executive Summary

Project Name	Therapoid

Overview

The scope of the audit was to analyse following contracts with functionalities mentioned below.

ParentOpenTherapoid

The parent ERC20 contract that goes on the Ethereum mainnet.
Apart from the standard ERC20 functions it has below functions which can be invoked only by the owner of the contract (Owner in this regard will be MultiSigWallet - GnosisSafe)
addAdmins
removeAdmins
setIssuerContract
updateMaxTransferLimit
updateTimeBetweenSubmissions
withdrawAll
withdrawStuckTokens
transferOwnership
cancelTransferOwnership
claimOwnership
pause (THIS CAN BE INVOKED EITHER BY OWNER OR ADMIN)
unpause (THIS CAN BE INVOKED EITHER BY OWNER OR ADMIN)
bulkTransfer (THIS CAN BE INVOKED EITHER BY OWNER OR ISSUER_CONTRACT)

ChildOpenTherapoid

It is the child equivalent of ParentOpenTherapoid
It has all the functions which are listed above and with the same constraint
It has 3 other functions which are specific/related to bridging of tokens from Ethereum to Polygon
updateChildChainManager
deposit
Withdraw

Executive Summary

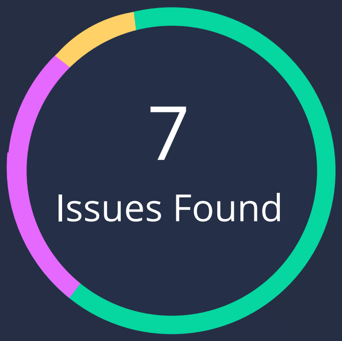
IssueSCI

This contract is specifically used for rewarding SCI tokens to the user
The function through which rewarding happens is issueBulkSCIToken
Functions that can only be called by the owner are (Owner in this regard will be MultiSigWallet - GnosisSafe)
withdrawAll
withdrawStuckTokens
updateThreshold
addIssuers
removeIssuers
transferOwnership

OpenScienceNFT
This contract is ERC721 standard and is used for awarding NFTs to the user
Functions that can only be called by the owner are (Owner in this regard will be MultiSigWallet - GnosisSafe)
updateAdmin
mint (THIS CAN BE INVOKED EITHER BY OWNER OR ADMIN)
withdrawContractEth
updateBaseURI (THIS CAN BE INVOKED EITHER BY OWNER OR ADMIN)

Timeline	14 October,2022 to 28 October,2022
Method	Manual Review, Functional Testing, Automated Testing, etc.
Scope of Audit	Zip File Shared on Mail

Executive Summary



🟥 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	0	1	4	2

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

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A1. Ownership Transfer must be a two-step process.

Contracts Affected: openscineceNFT.sol, IssueSCI.sol

Description

The contract uses openzeppelin's ownable contract to manage ownership. The transferOwnership() function in ownable contract allows the current owner to transfer his privileges to another address. However, inside transferOwnership() , the newOwner is directly stored in the storage, owner, after validating the newOwner is not same as contract address, which may not be enough.

If the current admin enters a wrong address by mistake, he would never be able to take the management permissions back.

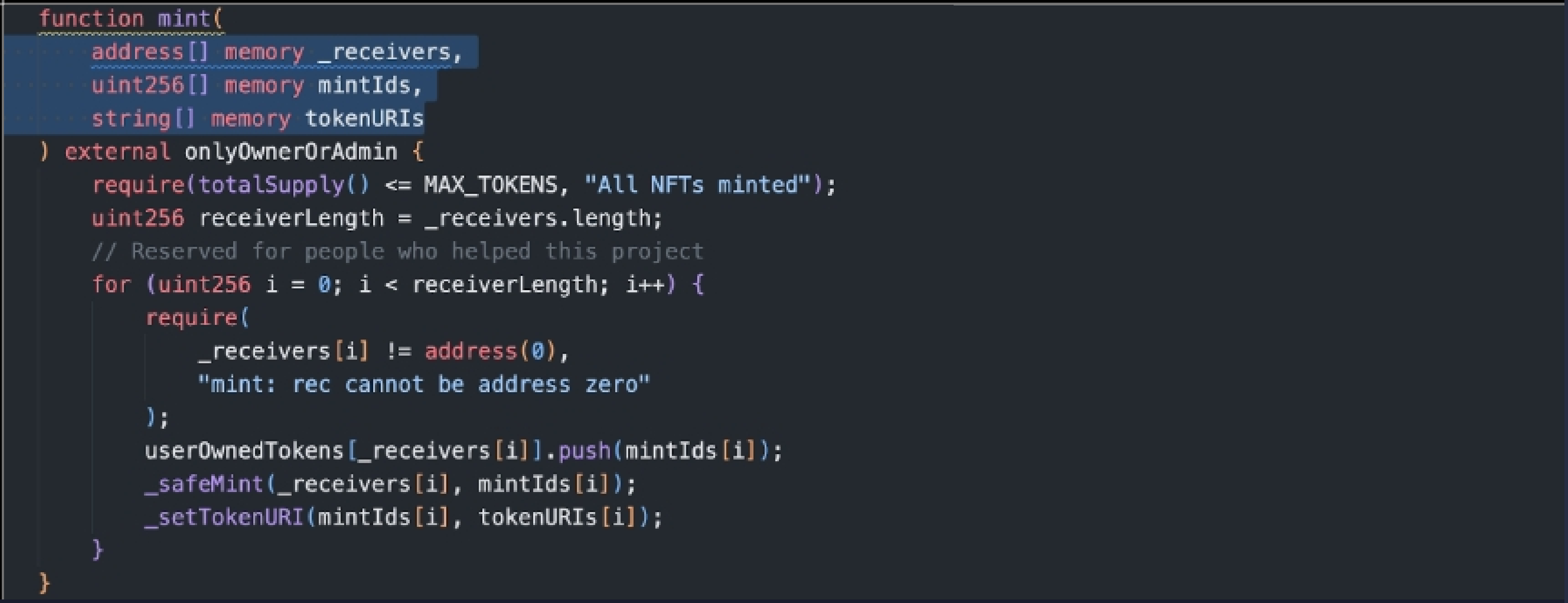
Remediation

It would be much safer if the transition is managed by implementing a two-step approach: _transferOwnership() and _updateOwnership() . Specifically, the _transferOwnership () function keeps the new address in the storage, _newOwner , instead of modifying the _owner() directly. The updateOwnership() function checks whether _newOwner is msg.sender, which means _newOwner signs the transaction and verifies himself as the new owner. After that, _newOwner could be set into _owner.

Status

Resolved

A.2 Missing array length check

Line	mint()
189	

```
function mint(
    address[] memory _receivers,
    uint256[] memory mintIds,
    string[] memory tokenURIs
) external onlyOwnerOrAdmin {
    require(totalSupply() <= MAX_TOKENS, "All NFTs minted");
    uint256 receiverLength = _receivers.length;
    // Reserved for people who helped this project
    for (uint256 i = 0; i < receiverLength; i++) {
        require(
            _receivers[i] != address(0),
            "mint: rec cannot be address zero"
        );
        userOwnedTokens[_receivers[i]].push(mintIds[i]);
        _safeMint(_receivers[i], mintIds[i]);
        _setTokenURI(mintIds[i], tokenURIs[i]);
    }
}
```

Contracts Affected: openscineceNFT.sol, IssueSCI.sol

Description

The mint function takes 3 arrays as input. However, there needs to be a check in place to ensure that the lengths of all the parameters are the same. For instance, the owner might mistakenly pass 2 receiver and only 1 mintID. This will cause transaction failure and other issues depending on the nature of the contract. It is not a major threat, in this case, however, it is recommended to always check the size of the input arrays.

This is not limited to mint() function. It applies to issueBulkSCIToken in – contract.

Remediation

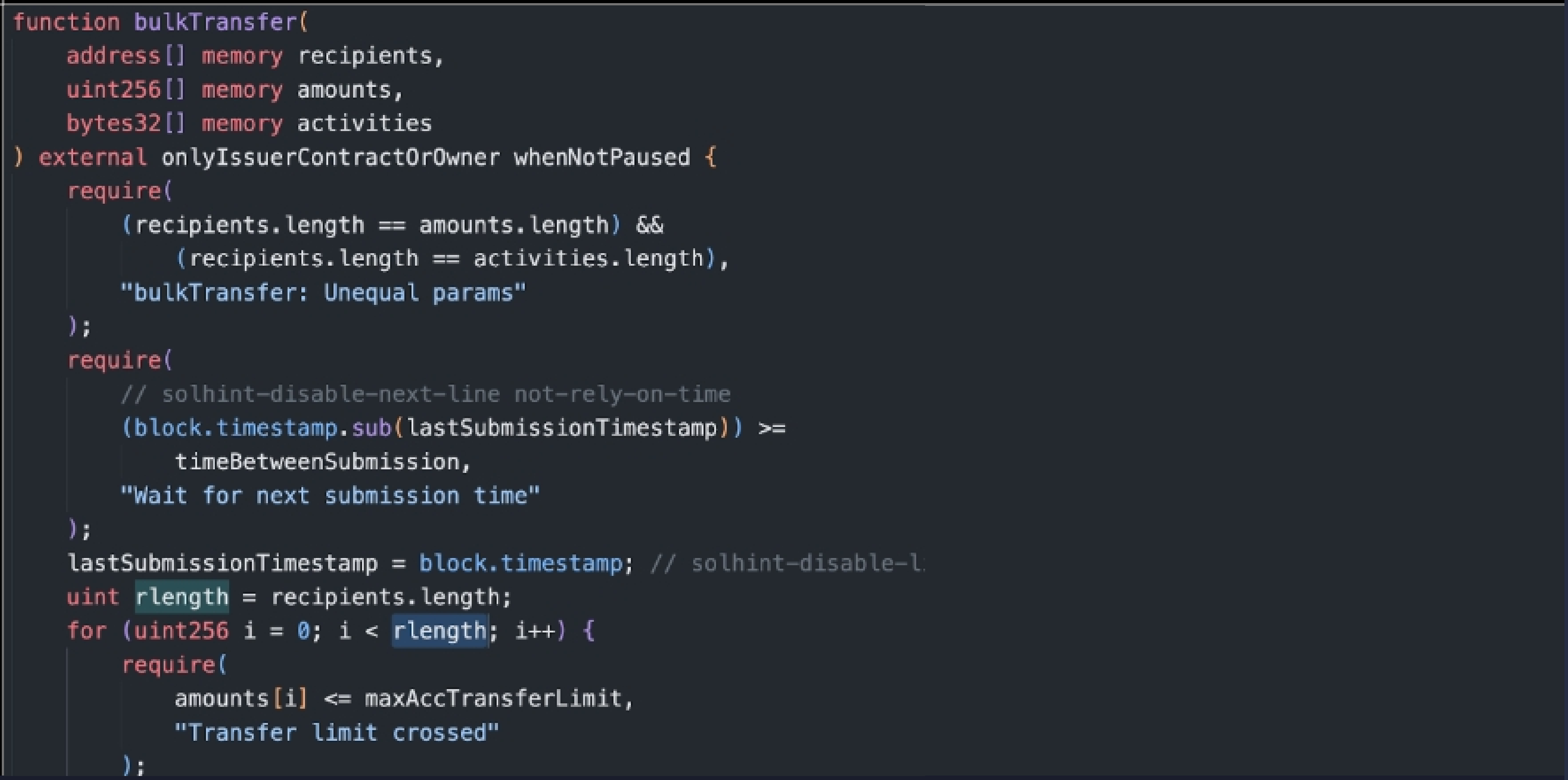
Consider adding a require statement that ensures that the length of all the inputs are same

Status

Resolved

Informational Issues

A.3 For loop optimization

Line	bulkTransfer()
189	

```
function bulkTransfer(
    address[] memory recipients,
    uint256[] memory amounts,
    bytes32[] memory activities
) external onlyIssuerContractOrOwner whenNotPaused {
    require(
        (recipients.length == amounts.length) &&
            (recipients.length == activities.length),
        "bulkTransfer: Unequal params"
    );
    require(
        // solhint-disable-next-line not-rely-on-time
        (block.timestamp.sub(lastSubmissionTimestamp)) >=
            timeBetweenSubmission,
        "Wait for next submission time"
    );
    lastSubmissionTimestamp = block.timestamp; // solhint-disable-l
    uint rlength = recipients.length;
    for (uint256 i = 0; i < rlength; i++) {
        require(
            amounts[i] <= maxAccTransferLimit,
            "Transfer limit crossed"
        );
```

Contracts Affected: ChildOpenTherapoid, IssueSCI, ParentOpenTherapoid

Description

In bulkTransfer(), there is a for loop which iterates the value of recipients.length times. Each time the for loop executes recipient.length is calculated which consumes some gas. This can be optimized by calculating the value of recipient.length outside the for loop. The optimized loop would look like

This is not limited to bulkTransfer() function. It applies to all the for loops in the contracts.

Remediation

Consider modifying the for loops as suggested above.

Status

Resolved

B. Contract - ChildOpenTherapoid.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

B.1 State variable that can be declared immutable

Description

State variables that get initialized in the constructor and don't
change their value throughout the code, should be declared immutable to save gas. Here, the following variables could be declared immutable:

- L17 - deployer

Remediation

Consider declaring this variable as immutable

Status

Resolved

C. Contract - IssueSCI.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

D. Contract - OpenScienceNFT.sol

High Severity Issues

No issues found

Medium Severity Issues

D.1 NFT with mintID 1111111 will be replaced upon receiving another NFT.

Description

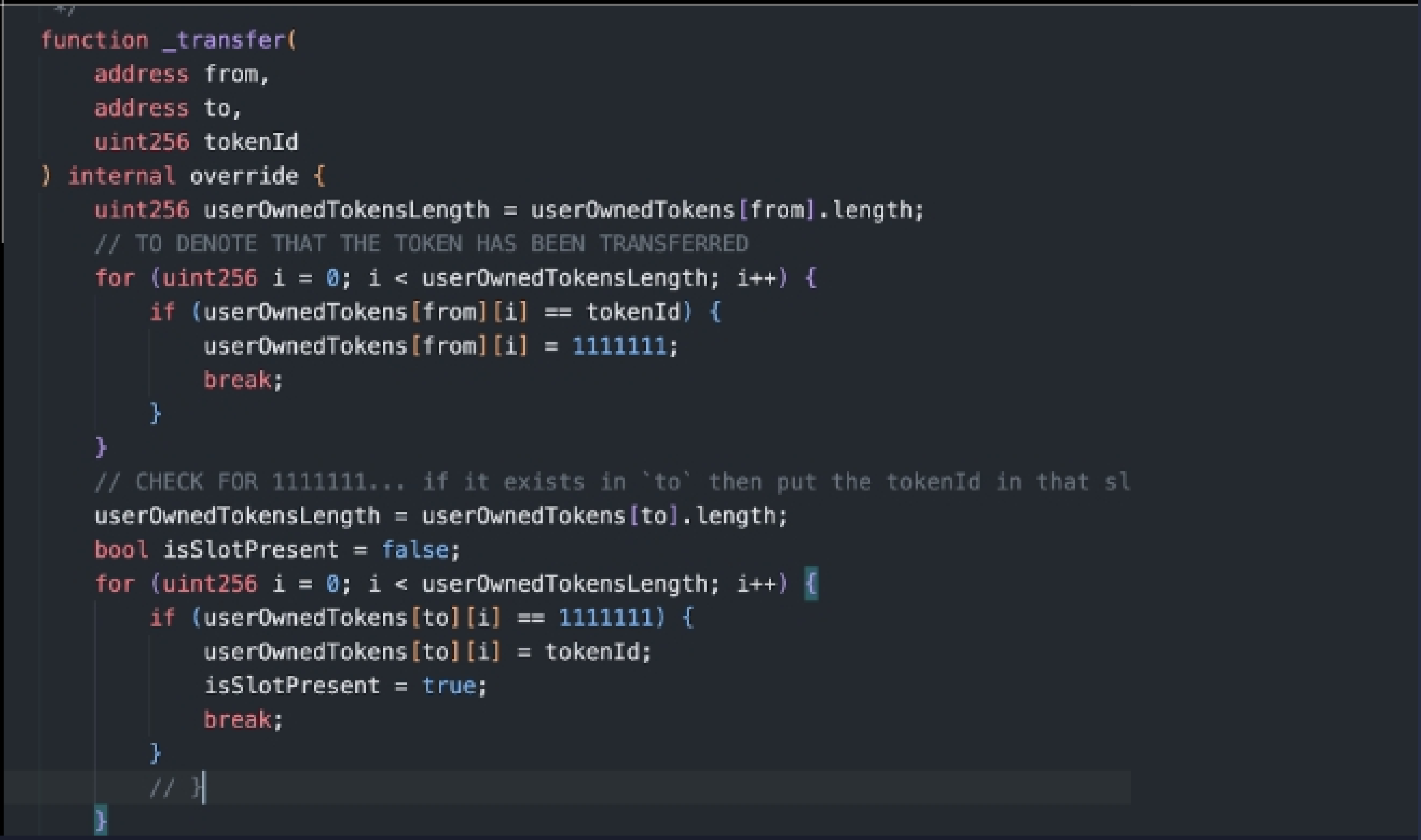
The contract OpenScienceNFT maintains an mapping named userOwnedTokens this mapping contains the information about what tokenIds does an address hold. This mapping is implemented to avoid using loops. The _transfer function is used for transferring an NFT from one address to another. However, in order to reduce the size of the array in long run, the developers have implemented a mechanism that empty slots as 1111111. Under normal circumstances, this is not an issue. However, if the receiver holds an NFT of mintId 1111111, it will be treated as an empty slot and it will be replaced by the mintID of received NFT.

Exploit Scenario

Consider a scenario where there are two users A and B. A has NFTs of mintID 1 and 2. B has tokens of id 3 and 4. When A transfers token of id 1 to 2,
sciNFTsOF will return [1111111,2] for A and [3,4,1] for B. But lets assume that B had an NFT of token ID 1111111. In this case, it will be considered as an empty slot and it will be replaced by the mintid of received NFT.

Remediation

Adding a small require statement at mint that ensures that an NFT of mintID 1111111 can not be minted.

Line	_transfer()
145	

```
    */
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal override {
    uint256 userOwnedTokensLength = userOwnedTokens[from].length;
    // TO DENOTE THAT THE TOKEN HAS BEEN TRANSFERRED
    for (uint256 i = 0; i < userOwnedTokensLength; i++) {
        if (userOwnedTokens[from][i] == tokenId) {
            userOwnedTokens[from][i] = 1111111;
            break;
        }
    }
    // CHECK FOR 1111111... if it exists in `to` then put the tokenId in that sl
    userOwnedTokensLength = userOwnedTokens[to].length;
    bool isSlotPresent = false;
    for (uint256 i = 0; i < userOwnedTokensLength; i++) {
        if (userOwnedTokens[to][i] == 1111111) {
            userOwnedTokens[to][i] = tokenId;
            isSlotPresent = true;
            break;
        }
        // }
    }
```

Status

Acknowledged

Low Severity Issues

No issues found

Informational Issues

No issues found

E. Contract - ParentOpenTherapoid.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

E.1 Centralization Issue

Description

The constructor of ParentOpenTherapoid takes fixedsupply of token as input. Whenever the contract is deployed, whatever value is passed as a fixedSupply, is minted to owner address. Until the tokens are distributed, the tokens are held by the owner. The implementation is prone to centralization risk that may arise if the owner loses its private key.

Recommendation

Ensure the private keys of the owner account are diligently handled.

Status

Resolved

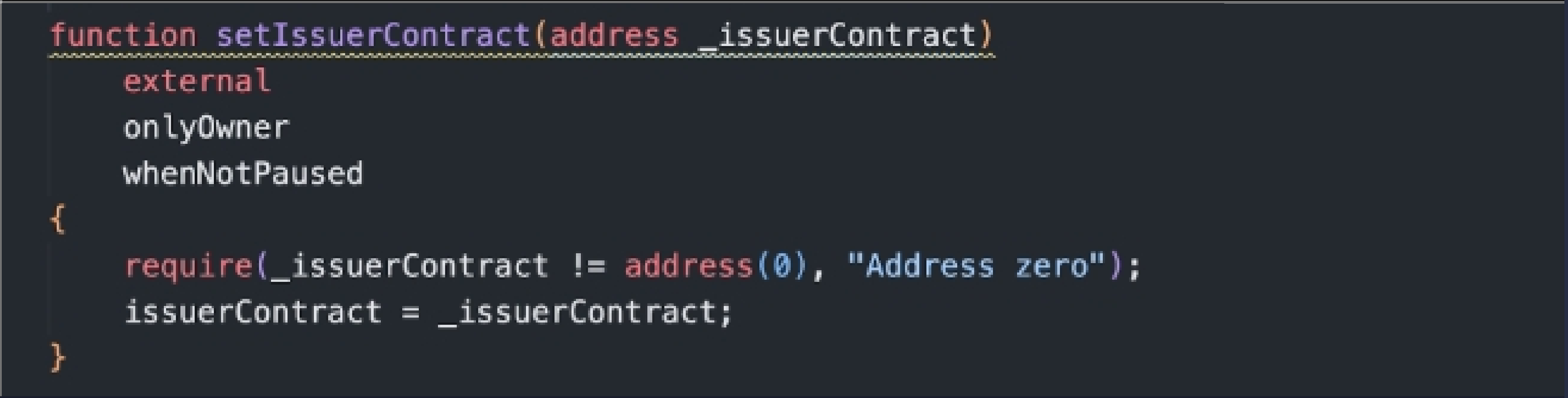
E.2 Missing events for significant actions

Description

Whenever certain significant privileged actions are performed within the contract, it is recommended to emit an event about it. Changing IssuerContract, is an important action hence it is recommended to emit an event.

Remediation

Consider emitting an event whenever certain significant changes are made in the contracts.

Line	setIssuerContract()
151	

```
function setIssuerContract(address _issuerContract)
    external
    onlyOwner
    whenNotPaused
{
    require(_issuerContract != address(0), "Address zero");
    issuerContract = _issuerContract;
}
```

Status

Resolved

Informational Issues

No issues found

Functional Testing

For ChildOpenTherapoid.sol

- ✓ Should deploy the contract
- ✓ Should test name and symbol
- ✓ Should test add admins
- ✓ Should revert when an user without owner role tries to add admin
- ✓ Should test isAdmin
- ✓ Should test removeAdmin
- ✓ Should test updateChildChainManager
- ✓ Should test deposit
- ✓ Should test withdraw
- ✓ Should test addAdmins
- ✓ Should test removeAdmins
- ✓ Should test burn
- ✓ Should test setIssuerContract
- ✓ Should test updateMaxTransferLimit
- ✓ Should test updateTimeBetweenSubmissions
- ✓ Should test bulkTransfer
- ✓ Should test withdrawall and withdraw stuck tokens
- ✓ Should test pause/unpause
- ✓ Should test transfer ownership and related functions.

For IssueSCI.sol

- ✓ Should test getters
- ✓ Should test setters
- ✓ Should test addissuers
- ✓ Should test removeIssuers
- ✓ Should test transfer ownership and related functions
- ✓ Should test issueBulkSCIToken

For openscineceNFT.sol

- ✓ Should test getters
- ✓ Should test setters
- ✓ Should test updateAdmin
- ✓ Should test mint
- ✓ Should test mint must revert when called by non-admin or owner.
- ✓ Should test withdrawContractEth
- ✓ Should test transfe

For UGWSFTUpgradeable.sol

- ✓ Should test getters
- ✓ Should test setters
- ✓ Should test add and remove admins
- ✓ Should test approve nd transferFrom
- ✓ Should test increase and decrease allowance
- ✓ Should test pause and unpause
- ✓ Should test burn
- ✓ Should test transfer ownership and related functions

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 Therapoid. We performed our audit according to the procedure described above.

Some issues of 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. In the End, Therapoid team resolved all Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of Therapoid. 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 Therapoid 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.

600+ Audits Completed	**$15B** Secured	**600K** Lines of Code Audited

Follow Our Journey

Audit Report November, 2022

For

OPEN Therapeutics

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com