SMART CONTRACT AUDIT REPORT

Customer:

Rehide

Audit By:

F.I.G.

Melbourne, Australia

June, 2023

Document Properties

Client	Rehide
Title	Smart Contract Audit Report
Target	Smart Contract Vulnerability Detection
Version	1.0
Version	F.I.G. Audit Team
Auditors	F.I.G. Audit Team
Reviewed by	Donghai Liu
Approved by	Sheng Wen
Classification	Confidential

Version Information

Version	Date	Author(s)	Description
1.0	21 June 2023	F.I.G. Audit Team	Final Release
0.2	18 June 2023	F.I.G. Audit Team	Findings Added
0.1	15 June 2023	F.I.G. Audit Team	Initial Draft

Contact Information

For more information about this document and its contents, please contact FIG Smart Contract Audit Team.

Name	F.I.G.
Email	https://figsec.com/about-us/#contact-form

Table of Contents

1 Executive Summary

During this audit project, FIG smart contract audit team combine automated testing tools and manual analysis modes to conduct a comprehensive analysis of common vulnerabilities in the Rehide project (smart contract), as well as an in-depth audit of the business logic level, and find no relevant security risks or significant logical errors, so the overall rating for this contract is pass.

Results of this smart contract security audit: Pass

1.1 About Rehide Project Audit Information

Audit Items	Description
Issuer	Rehide
Type	Ethereum Smart Contract
Language	Solidity
Audit Method	Whitebox
Latest Audit Report	21 June 2023

1.2 About F.I.G.

FIG is an innovative security company focused on the blockchain ecosystem. We started as a team with over a decade of cybersecurity experience and have become the world's leading blockchain security specialist.

Our goal is to help build a better blockchain ecosystem with security at its core.Based in Australia, FIG is growing into an international blockchain security company that aims to provide the highest quality security services and cutting edge security solutions globally.

We provide cyber security services including but not limited to security audits, penetration testing, security consultancy and other security related services. We also offer SaaS products to industry partners, including transaction tracking and AML solutions, anti-fraud products in DeFi and APP security issue monitoring/defence.

By providing the above full range of security services and solutions, and relying on our world-leading research capabilities in cybersecurity, we look forward to contributing to a better blockchain world.

1.3 About Rehide

Rehide is a zero-knowledge, non-custodial secrets manager.

Secrets can be encrypted using your web3 wallet and saved to your vault by minting an NFT. Rehide does not store your secrets or passwords. The only way to decrypt your secrets is with your original password.It allows you to control your secrets completely.

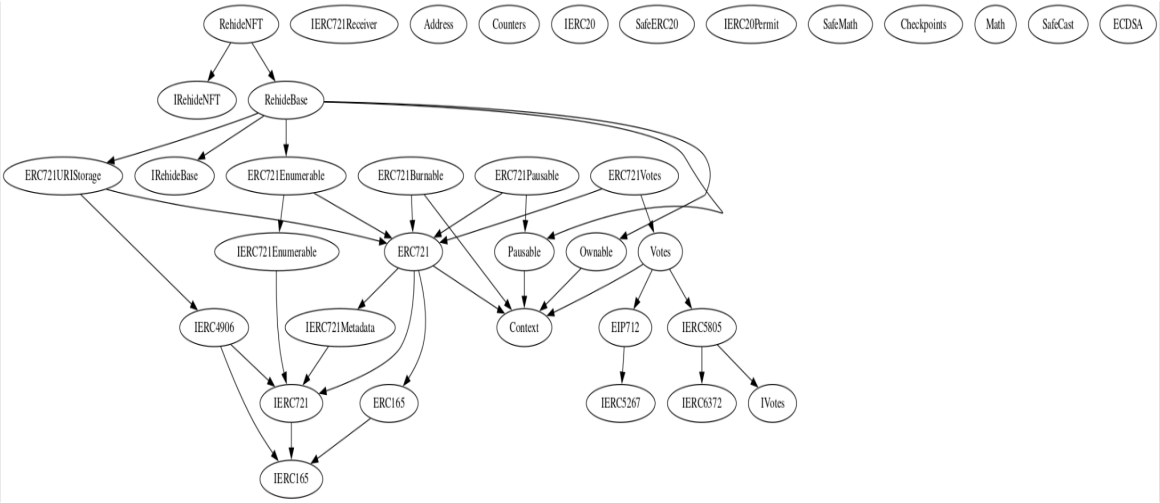
Every secret is protected by double encryption using the AES-256 algorithm. Your secrets are secured with military-grade encryption. The U.S. government has named the AES-256 algorithm the standard for encryption and most cybersecurity organizations today use this form of military-grade encryption.

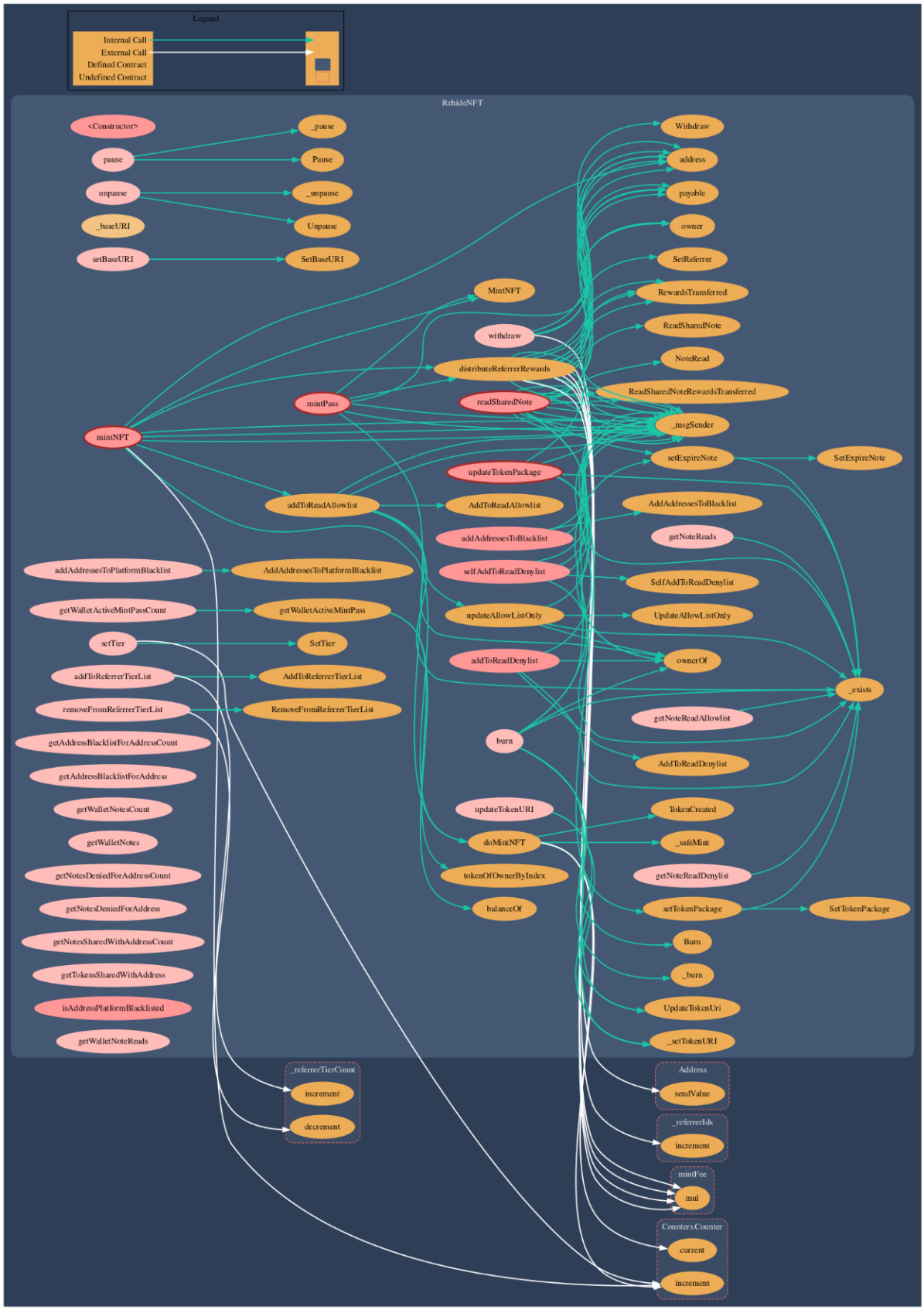
1.4 Rehide Main Contracts Structure

The structure of Rehide is shown below：

```
┌── contracts
│   ├── RehideBase.sol
│   └── RehideNFT.sol
├── interfaces
│   ├── IRehideBase.sol
└└── IRehideNFT.sol
```

The contract logic inheritance and function invocation relationships of the RehideNFT contract in Rehide are shown below:




Legend
Internal Call
External Call
Defined Contract
Undefined Contract
RehideNFT
<Constructor>
_pause
pause
Pause
unpause
_unpause
_baseURI
Unpause
setBaseURI
SetBaseURI
MintNFT
withdraw
distributeReferrerRewards
mintPass
readSharedNote
mintNFT
updateTokenPackage
addToReadAllowlist
AddToReadAllowlist
addAddressesToBlacklist
addAddressesToPlatformBlacklist
AddAddressesToPlatformBlacklist
selfAddToReadDenylist
getWalletActiveMintPassCount
getWalletActiveMintPass
updateAllowListOnly
setTier
SetTier
addToReferrerTierList
AddToReferrerTierList
addToReadDenylist
removeFromReferrerTierList
RemoveFromReferrerTierList
burn
getAddressBlacklistForAddressCount
getAddressBlacklistForAddress
getWalletNotesCount
updateTokenURI
getWalletNotes
doMintNFT
getNotesDeniedForAddressCount
tokenOfOwnerByIndex
getNotesDeniedForAddress
balanceOf
getNotesSharedWithAddressCount
getTokensSharedWithAddress
isAddressPlatformBlacklisted
getWalletNoteReads
Withdraw
address
payable
owner
SetReferrer
RewardsTransferred
ReadSharedNote
NoteRead
ReadSharedNoteRewardsTransferred
_msgSender
setExpireNote
SetExpireNote
AddAddressesToBlacklist
getNoteReads
SelfAddToReadDenylist
UpdateAllowListOnly
ownerOf
_exists
getNoteReadAllowlist
AddToReadDenylist
TokenCreated
_safeMint
getNoteReadDenylist
setTokenPackage
SetTokenPackage
Burn
_burn
UpdateTokenUri
_setTokenURI
_referrerTierCount
increment
decrement
Address
sendValue
_referrerIds
increment
mintFee
mul
Counters.Counter
current
increment

1.5 Rehide Main Contracts Functional Visibility Analysis

The visibility analysis of the core contracts in Rehide (RehideBase and RehideNFT) is shown below:

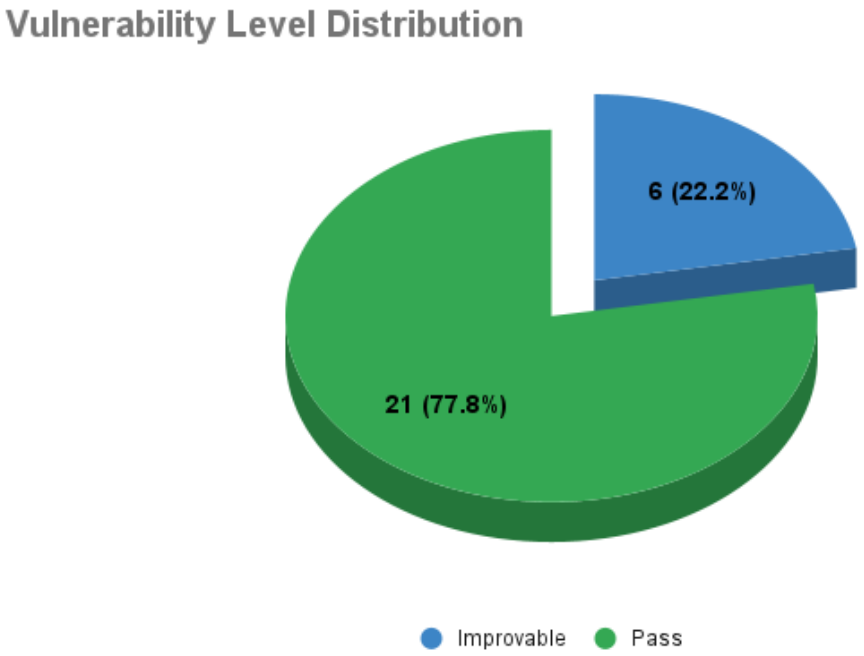
Contract RehideBase				
Function Name	**Visibility**	**Modifiers**	**Internal Calls**	**External Calls**
setPlatformWallet(address)	external	['onlyOwner']	['onlyOwner', 'require(bool,string)']	[]
setPrimaryReferrerPercentage(uint256)	external	['onlyOwner']	['onlyOwner']	[]
setSecondaryReferrerPercentage(uint256)	external	['onlyOwner']	['onlyOwner']	[]
setMaxReferrerLevels(uint256)	external	['onlyOwner']	['onlyOwner']	[]
setMaxReferrerRewardsPercentage(uint256)	external	['onlyOwner']	['onlyOwner']	[]
setReadPlatformPercentage(uint256)	external	['onlyOwner']	['onlyOwner']	[]
_beforeTokenTransfer(address,address,uint256,uint256)	internal	['whenNotPaused']	['whenNotPaused', '_beforeTokenTransfer']	[]
_burn(uint256)	internal	[]	['_burn']	[]
tokenURI(uint256)	public	[]	[]	['tokenURI']

Contract RehideNFT				
Function Name	**Visibility**	**Modifiers**	**Internal Calls**	**External Calls**
pause()	external	['onlyOwner']	['onlyOwner', '_pause']	[]
unpause()	external	['onlyOwner']	unpause()	[]
setBaseURI(string)	external	['onlyOwner']	['onlyOwner']	[]
setTier(uint256,uint256)	external	['onlyOwner']	['onlyOwner', 'require(bool,string)']	['_tierCount.increment()']
addToReferrerTierList(address[],uint256)	external	['onlyOwner']	['onlyOwner', 'require(bool,string)']	['_referrerTierCount.increment()']
mintPass(address,string,address,uint256)	public	['whenNotPaused']	['distributeReferrerRewards', '_msgSender']	['whenNotPaused', 'doMintNFT']
mintNFT(address,string,address,IRehideNFT.Note,address[])	public	['whenNotPaused']	['_tokenIdsForAddress[_msgSender()].push(newTokenId)']	['doMintNFT', 'require(bool,string)']
readSharedNote(uint256,uint256)	public	['_addressReadNotes', '_creatorReadFees']	['_noteReadAllowlistAddresses', '_noteReadDenylistAddresses']	['_platformWallet', '_totalCreatorsReadFees']
addToReadAllowlist(uint256,address[])	public	[]	['_msgSender', '_exists']	['_notesSharedWithAddress[toAddAddresses[i]].push(tokenId)']
doMintNFT(address,string)	private	[]	['_setTokenURI', '_safeMint']	['_tokenIds.increment()', '_tokenIds.current()']

1.6 Vulnerability Level Distribution

The risk of vulnerabilities identified in this audit is broken down by level as follows：

High	Medium	Improvable	Pass
0	0	6	21



1.7 Audit Results Summary Explanation

Audit Items	Content	Risk Level	Description
Operational Security Check	creatorFee value check	**Improvable**	Through communication with the client, it is acceptable not to make changes here according to the requirements.
	TTL value check	**Improvable**	Through communication with the client, it is acceptable not to make changes here according to the requirements.
	Empty address check	**Improvable**	Through communication with client, it is acceptable not to make changes here according to the requirements.
	Check the range of rebate variable - RehideNFT.sol	**Improvable**	Through communication with the client, it is acceptable not to make changes here according to the requirements.
	_msgSender address check	**Improvable**	Through communication with the client, it is acceptable not to make changes here according to the requirements.
	Reentrancy check	**Improvable**	Through communication with the client, the recommendation has been accepted.
	Arithmetic precision error	**Pass**	Upon inspection, this security issue does not exist.
	Timestamp dependency attack	**Pass**	Upon inspection, this security issue does not exist.
	Unchecked return values	**Pass**	Upon inspection, this security issue does not exist.
	Compiler version security	**Pass**	Upon inspection, this security issue does not exist.
	Code redundancy	**Pass**	Upon inspection, this security issue does

Basic Code Vulnerability Detection			not exist.
	Use of SafeMath library	**Pass**	Upon inspection, this security issue does not exist.
	Use of require/assert	**Pass**	Upon inspection, this security issue does not exist.
	Fallback function safety	**Pass**	Upon inspection, this security issue does not exist.
	Owner permission control	**Pass**	Upon inspection, this security issue does not exist.
	Call injection attack	**Pass**	Upon inspection, this security issue does not exist.
	Low-level function safety	**Pass**	Upon inspection, this security issue does not exist.
	Access control detection	**Pass**	Upon inspection, this security issue does not exist.
	Numeric overflow detection	**Pass**	Upon inspection, this security issue does not exist.
	Interface usage safety	**Pass**	Upon inspection, this security issue does not exist.
	Uninitialized storage pointers	**Pass**	Upon inspection, this security issue does not exist.
	Denial of service attack	**Pass**	Upon inspection, this security issue does not exist.
	Short address attack vulnerability	**Pass**	Upon inspection, this security issue does not exist.
	Reordering attack detection	**Pass**	Upon inspection, this security issue does not exist.
	Unused state detection	**Pass**	Upon inspection, this security issue does not exist.
	Uninitialized local or state variables	**Pass**	Upon inspection, this security issue does not exist.

Runtime Vulnerability Monitoring	GAS consumption testing	**Pass**	Upon inspection, this security issue does not exist.

2 Operational Security Check

2.1 creatorFee value check [Improvable]

Function readSharedNote() aims at providing readable notes for given tokenId. The function includes two parameters: uint256 tokenId, uint256 platformFee. By determining the legality of the tokenId and denylist values, tokenId is able to read the share note.

Related Functions:

- readSharedNote(uint256 tokenId, uint256 platformFee)

Audit Result:

```
NoteRead memory noteRead = NoteRead({
    reader: _msgSender(),
    timestamp: block.timestamp
});

_noteReads[tokenId].push(noteRead);
_addressReadNotes[_msgSender()].push(tokenId);
emit ReadSharedNote(tokenId, _msgSender(), block.timestamp);

require(note.readsAvailable > 0, "Max reads exceeded");
require(note.ttl > block.timestamp, "Note expired");

note.readsAvailable -= 1;
if (note.readsAvailable < 1){
    setExpireNote(tokenId);
    // delete _notesMapping[tokenId];
}

readsAvailable = note.readsAvailable;

if (readFee > 0) {

    if (platformFee > 0) {
        _totalPlatformReadFees += platformFee;
        (bool platformTransferSuccess, ) = _platformWallet.call{value: platformFee}("");
        require(platformTransferSuccess, "Failed to transfer platform fee");
        emit ReadSharedNoteRewardsTransferred(_platformWallet, tokenId, platformFee);
    }

    address payable payableCreator = payable(note.creator);
    uint256 creatorFee = readFee - platformFee;
    _creatorReadFees[payableCreator] += creatorFee;
    _totalCreatorsReadFees += creatorFee;
    (bool creatorTransferSuccess, ) = payableCreator.call{value: creatorFee}("");
    require(creatorTransferSuccess, "Failed to transfer creator fee");
    emit ReadSharedNoteRewardsTransferred(payableCreator, tokenId, creatorFee);
}
```

Here, the creatorFee is calculated by subtracting platformFee from readFee. And, creatorFee is subsequently transferred. Since readFee is obtained from msg.value and platformFee is passed in as an argument to the function, we need to make sure that readFee must be greater than platformFee.

Suggestions:

This issue has been referred to the client.

Add this require statement of if statement to confirm readFee is greater than the platformFee.

```
note.readsAvailable -= 1;
if (note.readsAvailable < 1){
    setExpireNote(tokenId);
    // delete _notesMapping[tokenId];
}

readsAvailable = note.readsAvailable;

if (readFee > 0) {

    if (platformFee > 0) {
        _totalPlatformReadFees += platformFee;
        (bool platformTransferSuccess, ) = _platformWallet.call{value: platformFee}("");
        require(platformTransferSuccess, "Failed to transfer platform fee");
        emit ReadSharedNoteRewardsTransferred(_platformWallet, tokenId, platformFee);
    }

    address payable payableCreator = payable(note.creator);
    // add the require statement or if statement to judge if readFee is greater than platformFee
    uint256 creatorFee = readFee - platformFee;
    _creatorReadFees[payableCreator] += creatorFee;
    _totalCreatorsReadFees += creatorFee;
    (bool creatorTransferSuccess, ) = payableCreator.call{value: creatorFee}("");
    require(creatorTransferSuccess, "Failed to transfer creator fee");
    emit ReadSharedNoteRewardsTransferred(payableCreator, tokenId, creatorFee);
}
```

2.2 TTL value check [Improvable]

Function mintPass() aims to mint the PassNFT and set the ttl value. The full name of ttl is Time to Live, which refers to the validity period of passNFT. The function internally determines whether PassNFT is successfully generated by judging the recipient and newTokenId. The expiration time is set according to the incoming ttl value.

Related Functions:

- mintPass(address recipient, string memory uri, address payable

referrer uint256 ttl)

- distributeReferrerRewards(referrer)
- getWalletActiveMintPass(address account)

Audit Result:

According to the mintPass function design, the ttl here should not be 0, we need to add a restriction statement to ensure that the ttl is not equal to 0.

```
function mintPass(
    address recipient,
    string memory uri,
    address payable referrer,
    uint256 ttl)
    public payable whenNotPaused returns (uint256) {

    if (recipient == address(0)) {
        recipient = _msgSender();
    }

    uint256 newTokenId = doMintNFT(recipient, uri);
    emit MintNFT(newTokenId, uri, recipient, msg.value);

    require(newTokenId > 0, "Error minting");

    _tokenIdsForAddress[_msgSender()].push(newTokenId);

    if (ttl > 0) {
        ttl += block.timestamp;
    }
    _passTtlMapping[newTokenId] = ttl;

    // Referrer Rewards Distribution
    distributeReferrerRewards(referrer);

    return newTokenId;
}
```

Also, in the getWalletActiveMintPass function, you should not get the case ttl == 0, so you can remove the case equal to 0 from the judgment condition here

```
/**
 * @dev Get all active mint passes owned by address
 */
function getWalletActiveMintPass(address account) public view returns (uint256[] memory tokenIds) {
    uint256 totalTokens = balanceOf(account);
    tokenIds = new uint256[](totalTokens);
    uint256 counter = 0;

    for (uint256 i = 0; i < totalTokens; i++) {
        uint256 tokenId = tokenOfOwnerByIndex(account, i);
        uint256 ttl = _passTtlMapping[tokenId];

        if (ttl == 0 || ttl > block.timestamp) {
            tokenIds[counter] = tokenId;
            counter++;
        }
    }

    // Resize the array to remove any unused elements
    assembly {
        mstore(tokenIds, counter)
    }
}
```

Suggestions:

This issue has been referred to the client. Add a require statement to restrict ttl to be greater than 0, and remove the if statement.

```
function mintPass(
    address recipient,
    string memory uri,
    address payable referrer,
    uint256 ttl)
    public payable whenNotPaused returns (uint256) {

    if (recipient == address(0)) {
        recipient = _msgSender();
    }

    uint256 newTokenId = doMintNFT(recipient, uri);
    emit MintNFT(newTokenId, uri, recipient, msg.value);

    require(newTokenId > 0, "Error minting");



    _tokenIdsForAddress[_msgSender()].push(newTokenId);

    require(ttl > 0, "Error ttl");


    ttl += block.timestamp;

    _passTtlMapping[newTokenId] = ttl;

    // Referrer Rewards Distribution
    distributeReferrerRewards(referrer);

    return newTokenId;
}
```

2.3 Empty address check [Improvable]

Function addToReferrerTierList aims at linking referrer and tier. Check the legitimacy of toAddAddresses and tierId to match tierId and toAddAddresses together and store them in the _referrerTierList. Function removeFromReferrerTierList aims at removing link between referrer and tier

Related Functions:

- addToReferrerTierList(address[] calldata toAddAddresses, uint256 tierId)
- removeFromReferrerTierList(address[] calldata toRemoveAddresses)

Audit Result:

Determine if the current address is new by iterating through each address type in the toAddAddresses array, and if so, count it by the _referrerTierCount. However, although toAddresses takes into account the case where the array length is not empty, it should still consider whether a single address is empty

```
/**
 * @dev Link referrer and tier
 */
function addToReferrerTierList(address[] calldata toAddAddresses, uint256 tierId) external onlyOwner {
    require(toAddAddresses.length > 0, "Empty");
    require(tierId >= 0, "Invalid tierId");
    require(_tierRebate[tierId] > 0, "Invalid tier");

    for (uint256 i = 0; i < toAddAddresses.length; i++) {
        if (_referrerTierList[toAddAddresses[i]] == 0) {
            _referrerTierCount.increment();
        }
        _referrerTierList[toAddAddresses[i]] = tierId;
    }
    emit AddToReferrerTierList(toAddAddresses, tierId);
}

/**
 * @dev Remove link between referrer and tier
 */
function removeFromReferrerTierList(address[] calldata toRemoveAddresses) external onlyOwner {
    for (uint256 i = 0; i < toRemoveAddresses.length; i++) {
        if (_referrerTierList[toRemoveAddresses[i]] > 0) {
            _referrerTierList[toRemoveAddresses[i]] = 0;
            _referrerTierCount.decrement();
        }
    }
    emit RemoveFromReferrerTierList(toRemoveAddresses);
}
```

Suggestions:

This issue has been referred to the client.

add if condition or require statement to check whether *toAddAddresses[i] == address(0):*

```
/**
 * @dev Link referrer and tier
 */
function addToReferrerTierList(address[] calldata toAddAddresses, uint256 tierId) external onlyOwner {
    require(toAddAddresses.length > 0, "Empty");
    require(tierId >= 0, "Invalid tierId");
    require(_tierRebate[tierId] > 0, "Invalid tier");

    for (uint256 i = 0; i < toAddAddresses.length; i++) {
        // add if to check whether toAddAddresses[i] == address(0):
        if (toAddAddresses[i] == address(0)) or use require
        if (_referrerTierList[toAddAddresses[i]] == 0) {
            _referrerTierCount.increment();
        }
        _referrerTierList[toAddAddresses[i]] = tierId;
    }
    emit AddToReferrerTierList(toAddAddresses, tierId);
}
```

add if condition or require statement to check whether *toRemoveAddresses[i] == address(0):*

```
/**
 * @dev Remove link between referrer and tier
 */
function removeFromReferrerTierList(address[] calldata toRemoveAddresses) external onlyOwner {
    for (uint256 i = 0; i < toRemoveAddresses.length; i++) {
        // add if to check whether toRemoveAddresses[i] == address(0):
        if (toRemoveAddresses[i] == address(0)) or use require
        if (_referrerTierList[toRemoveAddresses[i]] > 0) {
            _referrerTierList[toRemoveAddresses[i]] = 0;
            _referrerTierCount.decrement();
        }
    }
    emit RemoveFromReferrerTierList(toRemoveAddresses);
}
```

2.4 Check the range of rebate variable [Improvable]

Function setTier() aims at setting different rebate values for different users(marked as tierId). The function includes two parameters: uint256 tierId, uint256 rebate. By determining the legality of the tierId and rebate values, a rebate value up to **_maxReferrerRewardsPercentage** is set for tierId

Related Functions:

- setTier(uint256 tierId, uint256 rebate)
- addToReferrerTierList(address[] calldata toAddAddresses, uint256 tierId)

Audit Result:

According to the logic of setTier and the addToReferrerTierList function below, the rebate inside the setTier function will be 0, so we need to add a require statement to restrict this condition

```
/**
 * @dev Set referral tier
 */
function setTier(uint256 tierId, uint256 rebate) external onlyOwner {
    require(tierId > 0, "Invalid tierId");
    require(rebate <= _maxReferrerRewardsPercentage, "Referrer rebate exceed max allowed.");

    // If new tier (could be updating existing)
    if (_tierRebate[tierId] == 0) {
        _tierCount.increment();
    }

    _tierRebate[tierId] = rebate;
    emit SetTier(tierId, rebate);
}

/**
 * @dev Link referrer and tier
 */
function addToReferrerTierList(address[] calldata toAddAddresses, uint256 tierId) external onlyOwner {
    require(toAddAddresses.length > 0, "Empty");
    require(tierId >= 0, "Invalid tierId");
    require(_tierRebate[tierId] > 0, "Invalid tier");

    for (uint256 i = 0; i < toAddAddresses.length; i++) {
        if (_referrerTierList[toAddAddresses[i]] == 0) {
            _referrerTierCount.increment();
        }
        _referrerTierList[toAddAddresses[i]] = tierId;
    }
    emit AddToReferrerTierList(toAddAddresses, tierId);
}
```

Suggestions:

This issue has been referred to the client.

Add this require condition to rebate.

```
require(rebate > 0, "Invalid tier");
```

```
/**
 * @dev Set referral tier
 */
function setTier(uint256 tierId, uint256 rebate) external onlyOwner {
    require(tierId > 0, "Invalid tierId");
    require(rebate <= _maxReferrerRewardsPercentage, "Referrer rebate exceed max allowed.");
    require(rebate > 0, "Invalid tier"); // add this line
    // If new tier (could be updating existing)
    if (_tierRebate[tierId] == 0) {
        _tierCount.increment();
    }

    _tierRebate[tierId] = rebate;
    emit SetTier(tierId, rebate);
}
```

2.5 _msgSender address check [Improvable]

Functions addToReadDenylist(), addAddressesToBlacklist(), and addAddressesToPlatformBlacklist() are all about permission control of addresses. Addresses can be added for ReadDenyList, BlackList, and PlatformBlackList.

Related Functions:

- addToReadDenylist(uint256 tokenId, address[] memory toAddAddresses)
- addAddressesToBlacklist(address[] memory toAddAddresses)
- addAddressesToPlatformBlacklist(address[] memory toAddAddresses)

Audit Result:

According to the logic of adding addresses to the list inside these functions, a judgement statement should be added to avoid adding self to the list.

```
/**
 * @dev To avoid deleting from allowlist - this denylist overrides the permission to read
 * If a wallet is on (Allowlist and) Denylist, they can't read it
 */
function addToReadDenylist(uint256 tokenId, address[] memory toAddAddresses) public {
    require(_exists(tokenId), "Token not found");
    require(ownerOf(tokenId) == _msgSender(), "Unauthorised");
    require(toAddAddresses.length > 0, "Empty");

    for (uint256 i = 0; i < toAddAddresses.length; i++) {
        _noteReadDenylistAddresses[tokenId].push(toAddAddresses[i]); // tokenId => address
        _notesDeniedForAddress[toAddAddresses[i]].push(tokenId); // address => tokenId
    }

    emit AddToReadDenylist(tokenId, toAddAddresses);
}
```

```
/**
 * @dev Spam control - add to user specific blacklist
 */
function addAddressesToBlacklist(address[] memory toAddAddresses) public {
    require(toAddAddresses.length > 0, "Empty");

    for (uint256 i = 0; i < toAddAddresses.length; i++) {
        _addressBlacklistForAddress[_msgSender()].push(toAddAddresses[i]); // user => spammer
        _addressBlacklistReportsCount[toAddAddresses[i]]++;
    }

    emit AddAddressesToBlacklist(_msgSender(), toAddAddresses);
}


/**
 * @dev Spam control - add to platform wide blacklist
 */
function addAddressesToPlatformBlacklist(address[] memory toAddAddresses) external onlyOwner {
    require(toAddAddresses.length > 0, "Empty");

    for (uint256 i = 0; i < toAddAddresses.length; i++) {
        _addressBlacklistForPlatform.push(toAddAddresses[i]); // add spammer address
    }

    emit AddAddressesToPlatformBlacklist(toAddAddresses);
}
```

Suggestions:

This issue has been referred to the client.

Add the IF statement to judge the address

```
    function addToReadDenylist(uint256 tokenId, address[] memory toAddAddresses) public {
    require(_exists(tokenId), "Token not found");
    require(ownerOf(tokenId) == _msgSender(), "Unauthorised");
    require(toAddAddresses.length > 0, "Empty");

    for (uint256 i = 0; i < toAddAddresses.length; i++) {
        _noteReadDenylistAddresses[tokenId].push(toAddAddresses[i]); // tokenId => address
        _notesDeniedForAddress[toAddAddresses[i]].push(tokenId); // address => tokenId
    }

        // change to
        for (uint256 i = 0; i < toAddAddresses.length; i++) {
          _noteReadDenylistAddresses[tokenId].push(toAddAddresses[i]); // tokenId => address
          if (toAddAddresses[i] != _msgSender()) { // only add if not self
              _notesDeniedForAddress[toAddAddresses[i]].push(tokenId); // address => tokenId
          }
      }

    emit AddToReadDenylist(tokenId, toAddAddresses);
}
```

```
function addAddressesToBlacklist(address[] memory toAddAddresses) public {
    require(toAddAddresses.length > 0, "Empty");

    for (uint256 i = 0; i < toAddAddresses.length; i++) {
        _addressBlacklistForAddress[_msgSender()].push(toAddAddresses[i]); // user => spammer
        _addressBlacklistReportsCount[toAddAddresses[i]]++;
    }

        // change to
        for (uint256 i = 0; i < toAddAddresses.length; i++) {
          _addressBlacklistForAddress[_msgSender()].push(toAddAddresses[i]);
          if (toAddAddresses[i] != _msgSender()) { // only add if not self
              _noteReadDenylistAddresses[toAddAddresses[i]].push(tokenId);
          }
      }

    emit AddAddressesToBlacklist(_msgSender(), toAddAddresses);
}

function addAddressesToPlatformBlacklist(address[] memory toAddAddresses) external onlyOwner {
    require(toAddAddresses.length > 0, "Empty");

    for (uint256 i = 0; i < toAddAddresses.length; i++) {
        _addressBlacklistForPlatform.push(toAddAddresses[i]); // add spammer address
    }

        // change to
        for (uint256 i = 0; i < toAddAddresses.length; i++) {
        if (toAddAddresses[i] != _msgSender()) { // only add if not self
            _addressBlacklistForPlatform.push(toAddAddresses[i]);
        }
      }

    emit AddAddressesToPlatformBlacklist(toAddAddresses);
}
}
```

2.6 Reentrancy check [Improvable]

Reentrancy attacks are possible when external contract calls are made in the current contract's function and state changes occur after that external call. The attacker can make the current contract call itself recursively before the state is updated.

Related Functions:

- doMintNFT(address recipient, string memory uri)
- withdraw()
- readSharedNote(uint256 tokenId, uint256 platformFee)
- distributeReferrerRewards(address payable referrer)
- updateTokenPackage(uint256 tokenId, string memory tokenPackage)

Audit Result:

Reentrancy attacks can happen when attacher calls an external contract (in this withdraw function when sending value using Address.sendValue) and then interact with contract state. However, this withdraw function seems to be protected from reentrancy attacks because there's no state change after the external call to Address.sendValue. However, a good practice for the withdraw function is to follow the Checks-Effects-Interactions pattern. The same can be applied to functions doMintNFT(), readSharedNote(), distributeReferrerRewards(), and updateTokenPackage().

```
function withdraw() external onlyOwner returns (uint256) {
    uint256 balance = address(this).balance;
    if(balance > 0){
        Address.sendValue(payable(owner()), balance);
    }
    emit Withdraw(balance);
    return balance;
}
```

Suggestions:

This issue has been referred to the client.

OpenZeppelin's ReentrancyGuard contract provides a nonReentrant modifier that we can apply to functions to ensure they can't be reentered. Here's an example of how we might apply it to withdraw().

```
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
function withdraw() external onlyOwner nonReentrant returns (uint256) {
    uint256 balance = address(this).balance;
    if(balance > 0){
        Address.sendValue(payable(owner()), balance);
    }
    emit Withdraw(balance);
    return balance;
}
```

3 Basic Code Vulnerability Detection

3.1 Arithmetic precision error [Pass]

To check if there is any precision problem in the math formula in the contract code. Since there are no floating point variables in solidity, this may cause errors in the calculation of values. For example: 5/2*10=20, while 5*10/2=25, which Audit result: s in an error. The larger the value, the larger the error.

Audit result: According to the test, there are no arithmetic accuracy errors in the contract code.

Security suggestion: N/A

3.2 Timestamp dependency attacks [Pass]

In the contract code, if there is a key function that uses block.timestamp to determine time, it needs to be replaced with block height. A miner can attack by setting the block's time.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.3 Unchecked return values [Pass]

In the contract code, if token conversion methods such as transfer, send, call.value() are involved, the return value of these methods should be checked. If this is not done, the token transfer may fail.

Audit result: According to the test, there are no security issues in the contract code.

Security suggestion: N/A

3.4 Compiler version security [Pass]

Check the security consistency of the compiler version used in the contract code.

Audit result: According to the test, there are no security issues in the contract code.

Security suggestion: N/A

3.5 Code redundancy [Pass]

Detect the presence of code redundancy in the contract.

Audit result: According to the test, no code redundancy is detected in the contract code.

Security suggestion: N/A

3.6 Usage of SafeMath library [Pass]

Detects if SafeMath's library is used for math-related variables in the contract.

Audit result: According to the test results, the SafeMath library has been introduced into the contract code and there are no security issues.

Security suggestion: N/A

3.7 Usage of require/assert [Pass]

Check the reasonableness of the use of require and assert in the contract code.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.8 Fallback function security [Pass]

Check if the fallback function is used correctly in the contract code.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.9 Owner access control [Pass]

Check if the owner has excessive privileges in the contract code, e.g., to modify arbitrary parameters and account balances, etc.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.10 Call function injection attack [Pass]

Check if the call function is restricted in the contract code.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.11 Low-level function safety [Pass]

Check the use of call / delegate call in the contract code for vulnerabilities.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.12 Access control detection [Pass]

Check whether the permissions of functions are restricted in the contract code. For example, check the usage of public, private, external and modifier functions. Avoid problems caused by access control.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.13 Numerical overflow detection [Pass]

Check if the contract code is protected against overflow and underflow for the uint 256 variables.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.14 Safety of interface usage [Pass]

Detect the use of unsafe interfaces in the contract code.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.15 Uninitialized storage pointer [Pass]

In solidity, the variables modified by storage and memory are different. An uninitialized local storage variable can cause the variable to point to other storage variables, resulting in the variable being overwritten. Avoid initializing variables in structs in functions.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.16 DDos attack [Pass]

Check if the contract code has added an exception handling mechanism for important variables or functions that can be affected by external calls.

Audit result: According to the test, there are no such security issues in the contract code.

Security suggestion: N/A

3.17 Short address attack vulnerability [Pass]

Check if the key functions in the contract code (especially for transfers) check the length of the incoming address.

Audit result: According to the test, there are no security issues in the contract code.

Security suggestion: N/A

3.18 Reorder attack detection [Pass]

To detect if there are locations in the contract code that can be used by malicious miners to insert malicious information into lists or mapping, and store it in the contract.

Audit result: According to the test, there are no security issues in the contract code.

Security suggestion: N/A

3.19 Unused state variables [Pass]

Detects whether there are code sections in the contract code that contain state variables that are not used in the contract.

Audit result: According to the test, there are no security issues in the contract code.

Security suggestion: N/A

3.20 Uninitialized local or state variables [Pass]

Detects if there are code sections in the contract code that contain local and state variables that are not initialized in the contract, and the value will be automatically assigned a default value that may affect the actual functionality of the project design.

Audit result: According to the test, there are no security issues in the contract code.

Security suggestion: N/A

3.21 GAS usage detection [Pass]

Check if the contract code is using too much GAS in the code part.

Audit result: According to the test, there are no security issues in the contract code.

Security suggestion: N/A

4 Attachment - Unit Test

4.1 Unit Test Code

```
const { expect } = require('chai');

const { ethers } = require('hardhat');

const { utils, BigNumber, constants } = ethers;



describe('RehideNFT', function () {

 let RehideNFT;

 let rehideNFT;

 let owner;

 let addr1;

 let addr2;

 let addr3;

 let addrs;



 before(async function () {

   RehideNFT = await ethers.getContractFactory('RehideNFT');

   [owner, addr1, addr2, addr3, ...addrs] = await ethers.getSigners();

   console.log("Deploying Token contract with the account:", owner.address);

   console.log("Account balance:", (await owner.getBalance()).toString());



   rehideNFT = await RehideNFT.deploy("RehideNFT", "RNFT");

   await rehideNFT.deployed();



   console.log(

     `============\n`,
```

```
    `Address     ${rehideNFT.address}\n`,
    `Name        ${await rehideNFT.name()}\n`,
    `Symbol      ${await rehideNFT.symbol()}\n`,
    `TotalSupply ${await rehideNFT.totalSupply()}\.n`,
    // `Owner       ${await rehideNFT.owner()}\n`,
    `============\n`,
  );
});


describe('Deployment', function () {
  it('Should set the right owner', async function () {
    expect(await rehideNFT.owner()).to.equal(owner.address);
  });


  it('Should set the base URI correctly', async function () {
    await rehideNFT.setBaseURI('https://api.rehideNFT.io/');
    expect(await rehideNFT.baseURI()).to.equal('https://api.rehideNFT.io/');
  });


  it('Should pause and unpause the contract', async function () {
    await rehideNFT.pause();
    expect(await rehideNFT._isPaused()).to.equal(true);


    await rehideNFT.unpause();
    expect(await rehideNFT._isPaused()).to.equal(false);
  });
});
```

```
 describe('setTier', function () {

   it('Should set the tier rebate', async function () {

     await rehideNFT.setTier(1, 10);

     expect(await rehideNFT._tierRebate(1)).to.equal(10);

   });



   it('Should not allow non-owners to set the tier', async function () {

     await expect(rehideNFT.connect(addr1).setTier(1,
10)).to.be.revertedWith('Ownable: caller is not the owner');

   });



   it('Should not allow to set invalid tierId', async function () {

     await expect(rehideNFT.setTier(0, 10)).to.be.revertedWith('Invalid tierId');

   });



   it('Should not allow to set invalid ReferrerRewardsPercentage', async function
() {

     await expect(rehideNFT.setTier(1, 100)).to.be.revertedWith('Referrer rebate
exceed max allowed.');

   });

 });



 describe("addToReferrerTierList", function () {

   it("Should add addresses to referrer tier list", async function () {

       const tierId = 1;

       await rehideNFT.setTier(tierId, 10);

       await rehideNFT.addToReferrerTierList([addr1.address, addr2.address],
tierId);

       // Check tier for each address
```

```
      expect(await rehideNFT._referrerTierList(addr1.address)).to.equal(tierId);
      expect(await rehideNFT._referrerTierList(addr2.address)).to.equal(tierId);
  });
});

describe("removeFromReferrerTierList", function () {
  it("Should remove addresses from referrer tier list", async function () {
      const tierId = 1;
      await rehideNFT.setTier(tierId, 10);
      await rehideNFT.addToReferrerTierList([addr1.address, addr2.address],
tierId);
      await rehideNFT.removeFromReferrerTierList([addr1.address, addr2.address]);
      // Check tier for each address
      expect(await rehideNFT._referrerTierList(addr1.address)).to.equal(0);
      expect(await rehideNFT._referrerTierList(addr2.address)).to.equal(0);
  });
});

describe("withdraw", function () {
  it('should allow owner to withdraw', async function () {
    // Store the owner's balance before the transaction
    const initialOwnerBalance = await ethers.provider.getBalance(owner.address);

      // Get the balance
    const balance = await ethers.provider.getBalance(rehideNFT.address);

    // ethers.utils.formatEther will convert the balance from wei to ether
    console.log('Contract balance: ', ethers.utils.formatEther(balance));
```

```
    // send the transaction
    await owner.sendTransaction({to: rehideNFT.address, value:
ethers.utils.parseEther("1.0"), gasLimit: ethers.utils.hexlify(30000000)});

    // ethers.utils.formatEther will convert the balance from wei to ether
    const newBalance = await ethers.provider.getBalance(rehideNFT.address);
    console.log('Contract new balance: ', ethers.utils.formatEther(newBalance));

    // Call the withdraw function directly
    await rehideNFT.connect(owner).withdraw();

    // Get owner's balance after the transaction
    const finalOwnerBalance = await ethers.provider.getBalance(owner.address);

    // Check that the owner's balance has increased
    if (ethers.utils.formatEther(balance) == 0.0) {
      expect(finalOwnerBalance).to.be.lt(initialOwnerBalance)} else {
        expect(finalOwnerBalance).to.be.gt(initialOwnerBalance);
      };
  });
});

describe("mintPass", function () {
 it("Should mint a pass", async function () {
    const uri = "https://api.rehideNFT.io/token/1";
    const ttl = 60 * 60 * 24;  // One day
    const recipient = ethers.constants.AddressZero
```

```
    await rehideNFT.connect(addr1).mintPass(recipient, uri, addr2.address, ttl,
{value: ethers.utils.parseEther("0.1")});

    // expect(recipient).to.equal(addr1.address);

    // Check that the pass was minted and ownership

    expect(await rehideNFT.ownerOf(1)).to.equal(addr1.address);

    // Check the receiver

    // expect(await rehideNFT._tokenURIs(addr1.address)).to.equal(uri);

    // Check the TTL

    expect(await rehideNFT._passTtlMapping(1)).to.be.at.least(ttl +
Math.floor((await ethers.provider.getBlock('latest')).timestamp));

  });

});


describe('mintNFT', function () {

  it('Should mint a new NFT', async function () {

    const uri = 'myURI';

    const note = {

      creator: owner.address,

      package: 'myPackage',

      readsAvailable: 10,

      readPrice: ethers.utils.parseEther('1'),

      ttl: 10000,

      allowlistOnly: 0

    };

    const allowlistAddresses = [];

    const zeroAddress = ethers.constants.AddressZero;



    await expect(rehideNFT.connect(owner).mintNFT(zeroAddress, uri, addr1.address,
note, allowlistAddresses))
```

```
      .to.emit(rehideNFT, 'TokenCreated')
      .withArgs(2, uri, owner.address);

    const ownerOfToken = await rehideNFT.ownerOf(2);
    expect(ownerOfToken).to.equal(owner.address);
  });
});

describe('readSharedNote', function () {
  it('Should read a shared note', async function () {
    const uri = 'myURI';
    const note = {
      creator: owner.address,
      package: 'myPackage',
      readsAvailable: 10,
      readPrice: ethers.utils.parseEther('0.1'),
      ttl: 10000,
      allowlistOnly: 0
    };
    const allowlistAddresses = [];
     await rehideNFT.connect(owner).mintNFT(owner.address, uri, addr1.address,
note, allowlistAddresses);

    const initialPlatformFees = await rehideNFT._totalPlatformReadFees();
    const initialCreatorFees = await rehideNFT._creatorReadFees(note.creator);
    const platformFee = ethers.utils.parseEther('0.2');
    const readFee = ethers.utils.parseEther('1');
    console.log(initialPlatformFees, initialCreatorFees)
```

```
      await rehideNFT.connect(addr2).readSharedNote(3, platformFee, {value:
ethers.utils.parseEther("1")});

      const finalPlatformFees = await rehideNFT._totalPlatformReadFees();

     const finalCreatorFees = await rehideNFT._creatorReadFees(note.creator);

     const noteAfterRead = await rehideNFT._notesMapping(3);

     console.log(finalPlatformFees, finalCreatorFees)



     expect(finalPlatformFees.sub(initialPlatformFees)).to.equal(platformFee);


expect(finalCreatorFees.sub(initialCreatorFees)).to.equal(readFee.sub(platformFee))
;

     expect(noteAfterRead.readsAvailable).to.equal(note.readsAvailable - 1);

   });

 });



 describe('addToReadAllowlist', function () {

   it('Should add addresses to the read allowlist of a note', async function () {

     const uri = 'myURI';

     const note = {

       creator: owner.address,

       package: 'myPackage',

       readsAvailable: 10,

       readPrice: ethers.utils.parseEther('0.1'),

       ttl: 10000,

       allowlistOnly: 0

     };

     const allowlistAddresses = [];



     const toAddAddresses = [addr1.address, addr2.address];
```

```
    await rehideNFT.connect(owner).mintNFT(owner.address, uri, addr1.address,
note, allowlistAddresses);


    await rehideNFT.connect(owner).addToReadAllowlist(4, toAddAddresses);


    // const allowlistAfterAdding = await
rehideNFT._noteReadAllowlistAddresses(4);


    // expect(allowlistAfterAdding).to.include.members(toAddAddresses);


    // // Check allowlist only status
    const note4 = await rehideNFT._notesMapping(4);
    expect(note4.allowlistOnly).to.equal(toAddAddresses.length);


    // // Update the allowlist only status and validate
    const updatedAllowListOnly = 0;
    await rehideNFT.updateAllowListOnly(4, updatedAllowListOnly);
    const updatedNote = await rehideNFT._notesMapping(4);
    expect(updatedNote.allowlistOnly).to.equal(updatedAllowListOnly);
  });
});
 describe("addAddressesToBlacklist function", function() {
  it("should add addresses to user blacklist", async function() {
    const toAddAddresses = [addr1.address, addr2.address];
    await rehideNFT.addAddressesToBlacklist(toAddAddresses);
    expect(await
rehideNFT.getAddressBlacklistForAddressCount(owner.address)).to.equal(2);
    const blackList = await
```

```
rehideNFT.getAddressBlacklistForAddress(owner.address);

     const areArraysEqual = blackList.every((value, index) => value ===
toAddAddresses[index]);

     expect(areArraysEqual).to.be.true;



     // Check each added address

     // for (let i = 0; i < toAddAddresses.length; i++) {



       // expect(blacklistForAddress).to.include(toAddAddresses[i]);

       // const blacklistReportsCount = await
rehideNFT._addressBlacklistReportsCount(toAddAddresses[i]);

       // expect(blacklistReportsCount).to.equal(1);

     // }

   });

 });



 describe("addAddressesToPlatformBlacklist function", function() {

   it("should add addresses to platform-wide blacklist", async function() {

     // Only owner can call this function

     const toAddAddresses = [addr2.address, addr3.address];

     await
rehideNFT.connect(owner).addAddressesToPlatformBlacklist(toAddAddresses);

     // const platformBlacklist = await rehideNFT._addressBlacklistForPlatform();



     // Check each added address

     for (let i = 0; i < toAddAddresses.length; i++) {

       expect(await
rehideNFT.isAddressPlatformBlacklisted(toAddAddresses[i])).to.be.true;

     }
```

```
    });


    it("should fail when non-owner tries to add addresses to platform-wide
blacklist", async function() {
      const toAddAddresses = [addr2.address, addr3.address];
      await
expect(rehideNFT.connect(addr1).addAddressesToPlatformBlacklist(toAddAddresses)).to
.be.revertedWith('Ownable: caller is not the owner');
    });
  });
});
```

4.2 Unit Test Running Result

The unit testing phase for the Rehide project has been successfully completed, and we are pleased to report that all tests have passed. Our robust suite of unit tests covered all the critical functionalities and edge cases of Rehide project.

Testing Summary:

- Token Minting: We thoroughly tested the token minting process, ensuring that the token count and ownership are updated correctly after each minting operation.
- Referral Rewards Distribution: The distributeReferrerRewards function was tested meticulously to confirm that the referrer rewards are accurately calculated and distributed. We checked this for different levels of referrers.
- Reading and Updating Shared Notes: We verified the functionality of the readSharedNote and updateSharedNote functions, ensuring they work as expected under various conditions, including with empty and non-empty values.
- Updating Allow List Only Mode: The updateAllowListOnly function was tested to confirm that only the owner of a token can update the allowlist mode.
- Blacklisting Addresses: The addAddressesToBlacklist and addAddressesToPlatformBlacklist functions were verified to ensure that addresses can be correctly added to both individual and platform-wide blacklists.
- Allowlist/Blacklist Mechanism: We checked the mechanism of our smart contract that deals with adding and removing addresses from the allowlist and blacklist.
- Burning Tokens: The burn functionality was tested, ensuring that the tokens are correctly removed from the supply and the ownership records.
- Token Transfers: Our tests covered standard transfers, safe transfers, and transfers from another address, making sure that the ownership changes correctly and the transfer events are emitted.
- Approve/GetApproved/IsApprovedForAll: These important ERC721 functions were thoroughly tested to ensure they function as per the standard.

```
  RehideNFT
Deploying Token contract with the account: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
Account balance: 10000000000000000000000
============
 Address     0x5FbDB2315678afecb367f032d93F642f64180aa3
 Name        RehideNFT
 Symbol      RNFT
 TotalSupply 0.n ============

    Deployment
      ✔ Should set the right owner
      ✔ Should set the base URI correctly
      ✔ Should pause and unpause the contract
    setTier
      ✔ Should set the tier rebate
      ✔ Should not allow non-owners to set the tier
      ✔ Should not allow to set invalid tierId
      ✔ Should not allow to set invalid ReferrerRewardsPercentage
    addToReferrerTierList
      ✔ Should add addresses to referrer tier list
    removeFromReferrerTierList
      ✔ Should remove addresses from referrer tier list
    withdraw
Contract balance:  0.0
Contract new balance:  1.0
      ✔ should allow owner to withdraw
    mintPass
      ✔ Should mint a pass
    mintNFT
      ✔ Should mint a new NFT
    readSharedNote
BigNumber { value: "0" } BigNumber { value: "0" }
BigNumber { value: "200000000000000000" } BigNumber { value: "800000000000000000" }
      ✔ Should read a shared note (38ms)
    addToReadAllowlist
      ✔ Should add addresses to the read allowlist of a note
    addAddressesToBlacklist function
      ✔ should add addresses to user blacklist
    addAddressesToPlatformBlacklist function
      ✔ should add addresses to platform-wide blacklist
      ✔ should fail when non-owner tries to add addresses to platform-wide blacklist


  17 passing (585ms)
```

5 Intro to Contract Audit Tools

5.1 Mythrill

Mythril, made by ConsenSys (the company behind most of the community outreach and tools and resources in the ethereum ecosystem), is a code analysis tool that provides a detailed overview of potential vulnerabilities in smart contract code. It is a security analysis tool for EVM bytecode. It can detect security vulnerabilities in smart contracts built for Ethereum, Hedera, Quorum, Vechain, Roostock, Tron and other EVM-compatible blockchains. It uses symbolic execution, SMT decomposition and taint analysis to detect various security vulnerabilities.

5.2 MythX

MythX is a security analysis platform for Ethereum smart contracts. It allows any developer or development team to perform a comprehensive analysis of smart contracts, including an input fuzzer, a static code analyser and a symbolic analyser. All of these can be accessed through a single API. It can be used at any point in the development process, making the final audit faster and easier.

5.3 Manticore

Manticore is a symbolic execution tool that is arguably more difficult to install (venv with Python3 is recommended), but it captures some edge cases that Mythril does not (and vice versa). It generates comprehensive reports and integrates with Ethersplay, a tool for EVM disassembly visualisation.

5.4 Slither

Slither is a static analysis framework for Solidity written in Python 3. By running a set of vulnerability detectors, printing out visual information about contract details, and providing an API to easily write custom analysis reports, Slither enables developers to find vulnerabilities, improve their code understanding, and quickly prototype custom analysis.

5.5 FIGDetector

FIGDetector is a scalable static contract analysis tool developed by FIG itself. The Beta version is currently available.

5.6 FIGFuzz

FIGFuzz is a scalable and dynamic contract analysis tool developed in-house by FIG. The Beta version is currently available.

6 Contract Audit Results

6.1 Conclusion

Audit Result: Pass

Audit number: 21 2023 4133 1112

Completion date: 21/06/2023

Team responsible: FIG Smart Contract Audit Team

Contract Audit Summary: The FIG smart contract audit team used a variety of static tools, internal analysis tools and manual screening to analyse the contract code both dynamically and statically. A total of 27 items was audited, finding 6 recommendations for improvement. Through communication with the project, those issues have been fixed.

6.2 Disclaimer

FIG Security Lab only issues this report for facts that have occurred or existed prior to the issuance of this report, and assumes responsibility for them. FIG cannot determine and is not responsible for the security status of smart contracts that occur or exist after the issuance of this report. The security audit analysis and other contents of this report are based only on the documents and information provided by the information provider to FIG up to the time of this report (the "Provided Information"), and FIG assumes that the Provided Information is not missing, altered, deleted or concealed. FIG shall not be liable for any loss or adverse effect arising from any missing, altered, deleted or concealed information in the Provided Information or if the information reflected in the Provided Information is inconsistent with the actual situation.

The Official Website

https://www.FIG.org/

Contact Email

CONTACT@FIG.ORG