expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

SpaceRush Relics

RISK FINDINGS

Severity	Found
High	3
Medium	1
Low	3
Informational	3

TABLE OF CONTENTS

OVERVIEW

The Expelee team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analysed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

Audit Result	**High Risk Detected**
Audit Date	**24 May 2025**

CONTRACT DETAILS

Summary:
The SRR contract implements an ERC721 NFT with minting controls, transfer validation, royalty support, and owner-managed configuration. The code is generally well-structured and follows best practices, but a few areas require attention to improve security, robustness, and user experience.

Contract Name: SRR (SpaceRush Relics)

AUDIT METHODOLOGY

Audit Details

Our comprehensive audit report provides a full overview of the audited system's architecture, smart contract codebase, and details on any vulnerabilities found within the system.

Audit Goals

The audit goal is to ensure that the project is built to protect investors and users, preventing potentially catastrophic vulnerabilities after launch , that lead to scams and rugpulls.

Code Quality

Our analysis includes both automatic tests and manual code analysis for the following aspects:

- Exploits
- Back-doors
- Vulnerability
- Accuracy
- Readability

Tools

- Manual Review: The code has undergone a line-by-line review by the Ace team.
- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.
- Slither: The code has undergone static analysis using Slither.

VULNERABILITY CHECKS

Design Logic	Passed
Compiler warnings	Passed
Private user data leaks	Passed
Timestamps dependence	Passed
Integer overflow and underflow	Passed
Race conditions & reentrancy. Cross-function race conditions	Passed
Possible delays in data delivery	Passed
Oracle calls	Passed
Front Running	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model	Passed
Impact of the exchange rate on the logic	Passed
Malicious event log	Passed
Scoping and declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Cross-function race conditions	Passed
Safe Zepplin module	Passed

RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and acces control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Low Risk

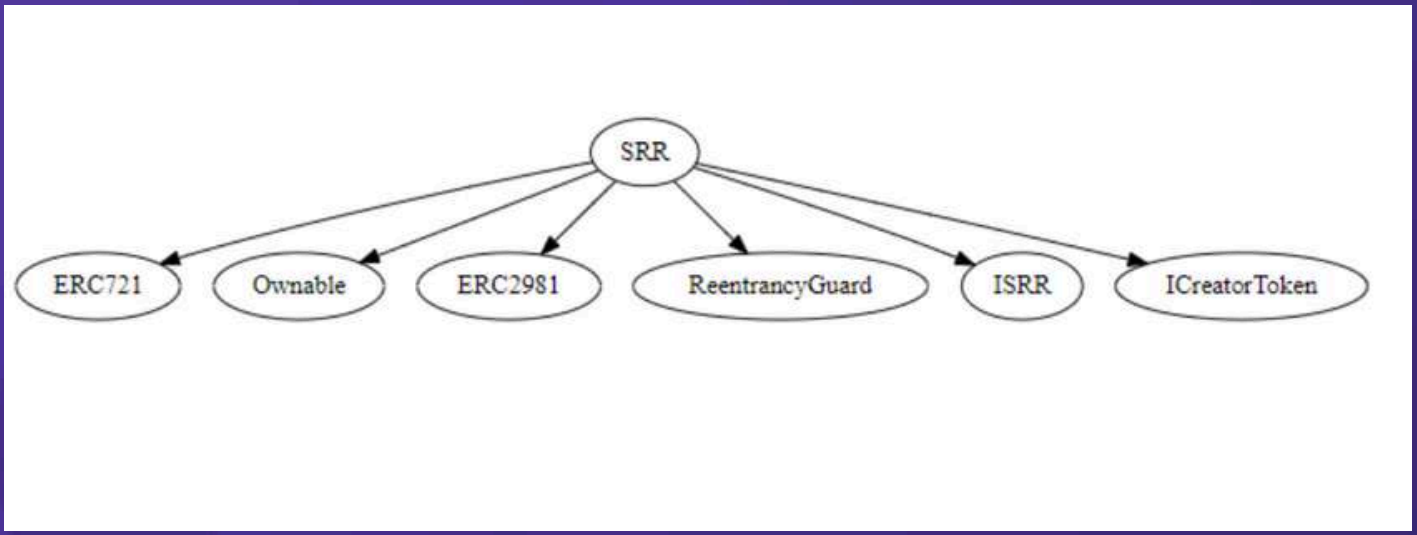
Issues on this level are minor details and warning that can remain unfixed.

Informational

Issues on this level are minor details and warning that can remain unfixed.

expelee

INHERITANCE TREE



OWNER PRIVILEGES

The contract owner has significant control over the protocol. The following actions can be performed by the owner:

- Minting Control:
  - Start or pause minting (startMinting, pauseMinting)
  - Set the maximum mints per wallet (setMaxMintsPerWallet)
  - Grant or revoke minter roles (grantMinterRole, grantMinterRoles, revokeMinterRole)

- Transfer Control:
  - Enable transfers (setTransfersEnabled)
  - Set the transfer validator contract (setTransferValidator)
  - Set automatic approval for transfers from the validator (setAutomaticApprovalOfTransfersFromValidator)

- Metadata and Royalties:
  - Set the base URI for token metadata (setBaseURI)
  - Set royalty information, including receiver and fee (setRoyaltyInfo)

- Other:
  - The owner is a default minter and can airdrop tokens if they have the minter role.

MANUAL REVIEW

Severity Criteria

Expelee assesses the severity of disclosed vulnerabilities according to methodology based on OWASP standarts.

Vulnerabilities are dividend into three primary risk categroies:
High
Medium
Low

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious input handling
- Escalation of privileges
- Arithmetic
- Gas use

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

HIGH RISK FINDING

Centralization – No Upper Bound and No event emitted

Severity: HIGH

Description:
The owner can set maxMintsPerWallet to any value, including 0 (which would prevent all future mints) or an extremely high value (removing the limit).

```
function setMaxMintsPerWallet(uint256 _limit) external onlyOwner {
  maxMintsPerWallet = _limit;
}
```

Recommendation: Should have to check for limit >0 and add an event for better transparency and safety. The main risk is owner misuse, which is a governance issue.

HIGH RISK FINDING

Centralization – Unrestricted Royalty Setting

Severity: HIGH

Description:
The setRoyaltyInfo function allows the owner to set royalties up to 100% or more.

```
function setRoyaltyInfo(address receiver, uint96 feeNumerator)
    external
    onlyOwner
  {
    _setDefaultRoyalty(receiver, feeNumerator);
  }
```

Recommendation: Add a maximum cap, e.g.:
require(feeNumerator <= 1000, "Royalty too high"); // Max 10%

HIGH RISK FINDING

Centralization – Airdrop Bypasses Per-Wallet Mint Limit

Severity: HIGH

Description:

The airdrop function does not enforce the maxMintsPerWallet limit for each recipient, allowing wallets to receive more than the intended maximum via airdrops.

```
function airdrop(address[] calldata accounts, uint256[] calldata quantities)
  external
  onlyMinter
  nonReentrant
 {
  uint256 len = accounts.length;
  require(len == quantities.length, "Length mismatch");

  uint256 _currentSupply = currentSupply;
  uint256 _maxTokenCount = maxTokenCount;
  uint256 totalQuantity;

  unchecked {
   for (uint256 i; i < len; ++i) {
    totalQuantity += quantities[i];
   }
  }

  uint256 newSupply = _currentSupply + totalQuantity;
  require(newSupply <= _maxTokenCount, "Max supply reached");

  uint256 nextId = _currentSupply + 1;
  currentSupply = newSupply;

  unchecked {
   for (uint256 i; i < len; ++i) {
    address recipient = accounts[i];
    uint256 qty = quantities[i];

    mintsPerWallet[recipient] += qty;

    for (uint256 j; j < qty; ++j) {
     _safeMint(recipient, nextId++);
    }
   }
  }
```

Recommendation: Add a check in the airdrop loop:
if (mintsPerWallet[recipient] + qty > maxMintsPerWallet) revert ThouShallNotMint();

MEDIUM RISK FINDING

Centralization – No Zero Address Check in setTransferValidator (External Function)

Severity: Medium

Description:

The external function setTransferValidator(address _validator) does not explicitly check if _validator is the zero address before calling the internal function. While the internal _setValidator function does have this check, it is best practice to validate inputs as early as possible for clarity and to avoid confusion.

```
function setTransferValidator(address _validator) external onlyOwner {
    _setValidator(_validator);
  }
```

Recommendation: Add a zero address check in the external function:

LOW RISK FINDING

Centralization – Redundant Supply Check in Mint

Severity: Low

Description:
The spacerushMint function checks supply limits twice.

```
 function spacerushMint(address to, uint256 quantity)
   public
   onlyMinter
   mintingActive
   nonReentrant
 {
   uint256 _currentSupply = currentSupply; // Cache storage variable
   uint256 _maxTokenCount = maxTokenCount;

   if (_currentSupply + quantity > _maxTokenCount)
     revert MaxSupplyReached();
   // Check limits using PRE-update values
   uint256 newSupply = currentSupply + quantity;
   if (newSupply > maxTokenCount) revert MaxSupplyReached();
   if (mintsPerWallet[to] + quantity > maxMintsPerWallet)
     revert ThouShallNotMint();
```

Recommendation:
Remove the redundant check for clarity and gas efficiency.

LOW RISK FINDING

Centralization – Missing Event Definition

Severity: Low

Description:
The contract emits TransferValidatorUpdated but does not define this event.

```
function _setValidator(address _validator) internal { @//@audit Add an event for
TransferValidatorUpdated
   require(_validator != address(0), "Zero address");
   emit TransferValidatorUpdated(validator, _validator);
   validator = _validator;

   if (_validator != address(0)) {
    // Register token type (721 for ERC721)
    (bool success, ) = _validator.call(
     abi.encodeWithSignature(
      "setTokenTypeOfCollection(address,uint16)",
      address(this),
      721
     )
    );
    if (!success) revert InvalidValidator();
   }
  }
```

Recommendation:
Add:
event TransferValidatorUpdated(address previousValidator, address
newValidator);

LOW RISK FINDING

Centralization – Missing Error Definition

Severity: Low

Description:

The contract reverts with TokenDoesNotExist() in tokenURI, but this error is not defined.

```
function tokenURI(uint256 id) public view override returns (string memory) {
//@audit Should have a check for token existence
   if (_exists(id) == false) {
     revert TokenDoesNotExist();
   }
   return string.concat(baseURI, LibString.toString(id));
 }
```

Recommendation:
Add: event TransferValidatorUpdated(address previousValidator, address newValidator);

INFORMATIONAL FINDING

Centralization – No Burn or Withdraw Function

Severity: Information

Description:

The contract does not allow burning tokens or withdrawing ETH (if received). Add if needed.

INFORMATIONAL FINDING

Centralization – Centralization Risk

Severity: Information

Description:

Owner can change validator and royalty settings at any time. This is a governance risk, not a code bug.

INFORMATIONAL FINDING

Centralization – Floating Pragma

Severity: Information

Description:

The pragma is set to ^0.8.28. Pinning to a specific version is recommended for production deployments.

ABOUT EXPELEE

Expelee is a product-based aspirational Web3 start-up. Coping up with numerous solutions for blockchain security and constructing a Web3 ecosystem from deal making platform to developer hosting open platform, while also developing our own commercial and sustainable blockchain.

www.expelee.com

expeleeofficial
expelee
Expelee
expelee
expelee_official
expelee-co

expelee

Building the Futuristic Blockchain Ecosystem

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantess against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always do your own research and project yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Alway do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.