Jun 28, 2024

ZAN

Security Assessment

CarvNft

—

Professional Service

Table of Contents

1. Overview

1.1. Executive Summary

CarvNft is a token project compatible with the ERC-721 standard. This report has been prepared for Carv to discover issues and vulnerabilities in the source code of this project as well as any contract dependencies that were not part of an officially recognized library.

Conducted by Static Analysis, Formal Verification and Manual Review, we have identified **1 High and 6 Informational issues**.

In commit a328463, the project team **resolved H-01, I-05, I-06, and I-07**, ackownledged the other issues and decided to keep no change.

1.2. Project Summary

Project Name	CarvNft
Platform	Arbitrum
Language	Solidity
Codebase	Audit 1: • https://github.com/carv-protocol/carv-contracts-alphanet/tree/4a7de1364db7fb9c7c91413735738c8c50a06d66 Final Audit: • https://github.com/carv-protocol/carv-contracts-alphanet/tree/a3284632a28c3114e5486f8ac146c0a139592b3f

1.3. Assessment Summary

Delivery Date	Jun 28, 2024
Audit Methodology	Static Analysis, Formal Verification, Manual Review

1.4. Assessment Scope

ID	File	File Hash
1	/contracts/interfaces/ICarvNft.sol	21b3873f5f807484fbe89eac1881092c
2	/contracts/CarvNft.sol	4d998c6aad488904b99f9b73a59b46f8

2. Checklist

2.1. Code Security

Reentrancy	DelegateCall	Integer Overflow
Input Validation	Unchecked this.call	Frozen Money
Arbitrary External Call	Unchecked Owner Transfer	Do-while Continue
Right-To-Left-Override Character	Unauthenticated Storage Access	Risk For Weak Randomness
TxOrigin	Missing Checks for Return Values	Diamond Inheritance
ThisBalance	VarType Deduction	Array Length Manipulation
Uninitialized Variable	Shadow Variable	Divide Before Multiply
Affected by Compiler Bug		

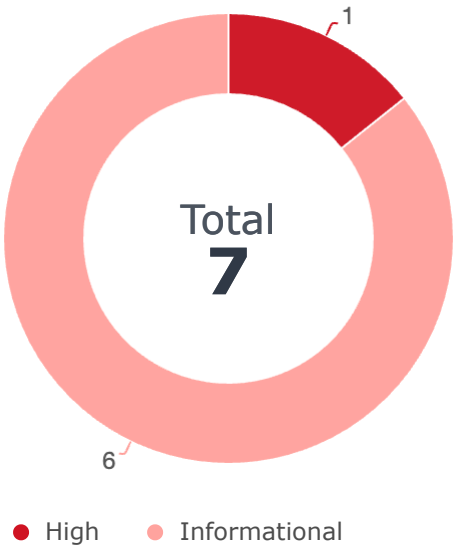
2.2. Optimization Suggestion

Compiler Version	Improper State Variable Modification
Function Visibility	Deprecated Function
Externally Controlled Variables	Code Style
Constant Specific	Event Specific
Return Value Unspecified	Inexistent Error Message
State Variable Defined Without Storage Location	Import Issue
Compare With Timestamp/Block Number/Blockhash	Constructor in Base Contract Not Implemented
Delete Struct Containing the Mapping Type	Usage of '=+'
Paths in the Modifier Not End with "_" or Revert	Non-payable Public Functions Use msg.value
Lack of SafeMath	Compiler Error/Warning
Tautology Issue	Loop Depends on Array Length
Redundant/Duplicated/Dead Code	Code Complexity/Code Inefficiency
Undeclared Resource	Optimizable Return Statement
Unused Resource	

2.3. Business Security

The Code Implementation is Consistent With Comments, Project White Papers and Other Materials
Permission Check
Address Check

3. Findings



Code Security

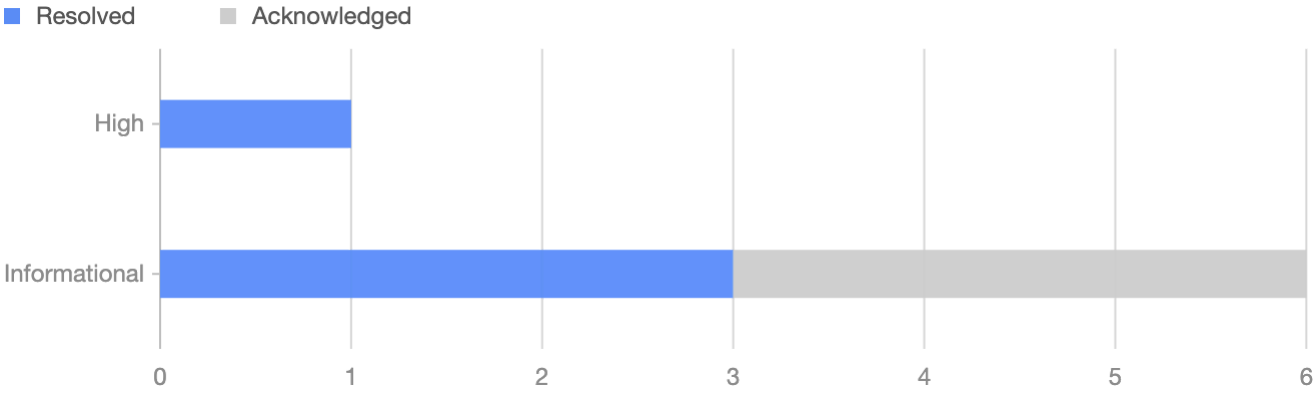
no vulnerabilities.

Optimization Suggestion

Informational: 6

Business Security

High: 1



ID	Title	Category	Severity	Status
H-01	The CarvNft Can Be Transferred by Anyone at Any Time	Business Security	High	Resolved
I-02	Addresses Can Only Be Added to the canTransferOnce Whitelist and Cannot Be Removed	Optimization Suggestion	Informational	Acknowledged
I-03	Cache State Variables That Are Read Multiple Times within a Function	Optimization Suggestion	Informational	Acknowledged
I-04	No Check of Address Params with Zero Address	Optimization Suggestion	Informational	Acknowledged
I-05	Parameters Should Be Declared as Calldata	Optimization Suggestion	Informational	Resolved
I-06	Floating Pragma	Optimization Suggestion	Informational	Resolved
I-07	Recommend to Follow Code Layout Conventions	Optimization Suggestion	Informational	Resolved

H-01: The CarvNft Can Be Transferred by Anyone at Any Time

High: Business Security

File Location: /contracts/CarvNft.sol:32

Description

In the `transferFrom` function, restrictions are placed on the transfer of CarvNft in terms of time, sender, and receiver. However, since the `CarvNft` contract may inherit the `ERC721` contract from OpenZeppelin of version lower than v5.0.0, users can bypass the various restrictions in the `transferFrom` function by directly calling the `safeTransferFrom` function of the `ERC721` contract to transfer CarvNft.

Code from `ERC721` contract of OpenZeppelin of version v4.8.3:

```
function safeTransferFrom(
    address from,
    address to,
    uint256 tokenId
) public virtual override {
    safeTransferFrom(from, to, tokenId, "");
}

/**
 * @dev See {IERC721-safeTransferFrom}.
 */
function safeTransferFrom(
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) public virtual override {
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: caller is
not token owner or approved");
    _safeTransfer(from, to, tokenId, data);
}
```

From the project's configuration file package.json, it can be seen that the project specifies an OpenZeppelin version greater than or equal to 4.8.3. Therefore, when compiling the contracts, the project could potentially use a version of the OpenZeppelin library that is less than 5.0.0.

package.json

```
"dependencies": {
  "@chainlink/contracts": "1.1.0",
  "@openzeppelin/contracts": ">=4.8.3",
```

```
    "@openzeppelin/contracts-upgradeable": "4.9.3",
    "@openzeppelin/hardhat-upgrades": "^1.17.0",
    "dotenv": "^16.4.5"
  },
```

/contracts/CarvNft.sol

```
32  function transferFrom(address from, address to, uint256 tokenId) public
    override {
33      if (to == redeemAddress) {
34          require(block.timestamp > redeemProhibitedUntil, "Redeem not
            allowed");
35          super.transferFrom(from, to, tokenId);
36          return;
37      }
38
39      if (block.timestamp < transferProhibitedUntil) {
40          require(canTransferOnce[from], "Transfer not allowed");
41          require(!transferred[tokenId], "Already Transferred");
42          transferred[tokenId] = true;
43      }
44
45      super.transferFrom(from, to, tokenId);
46  }
```

Recommendation

It is recommended to specify in the configuration file package.json that the version of OpenZeppelin used should be greater than or equal to v5.0.0.

```
  "@openzeppelin/contracts": ">=5.0.0",
```

Alleviation

Fixed in commit a328463.

I-02: Addresses Can Only Be Added to the canTransferOnce Whitelist and Cannot Be Removed

Informational: Optimization Suggestion

File Location: /contracts/CarvNft.sol:83

Description

The owner of the `CarvNft` contract can add an address to the `canTransferOnce` whitelist by calling the `setTransferOnceWhitelist` function, allowing an address in the whitelist to have the opportunity to transfer their NFTs once before the time `transferProhibitedUntil`. However, the `CarvNft` contract does not have a feature to remove an address from the `canTransferOnce` whitelist. If an address engages in malicious behavior, it is not possible to prevent it from transferring the NFT before the time `transferProhibitedUntil`.

/contracts/CarvNft.sol

```
80  function setTransferOnceWhitelist(address[] calldata whitelist) external
    onlyOwner {
81      require(whitelist.length > 0, "Empty whitelist");
82      for (uint i = 0; i < whitelist.length; i++) {
83          canTransferOnce[whitelist[i]] = true;
84      }
85  }
```

Recommendation

It is suggested to add the functionality to remove addresses from the `canTransferOnce` whitelist.

Alleviation

The project team acknowledged this issue and decided to keep no change.

I-03: Cache State Variables That Are Read Multiple Times within a Function

Informational: Optimization Suggestion

File Location: /contracts/CarvNft.sol:49

Description

The state variable `tokenIndex` is read multiple times in a function, using a local variable to cache the state variable can avoid frequently reading data from storage, thereby saving gas.

/contracts/CarvNft.sol

```
47
48      function mint(address receiver, uint256 count, MetaData calldata
        meta) public onlyOwner {
49          require(tokenIndex+count <= MAX_SUPPLY, "Mint finished");
50          for (uint i = 1; i <= count; i++) {
51              _safeMint(receiver, tokenIndex+i);
```

Recommendation

When a state variable is read multiple times in a function, it is recommended to use a local variable to cache the state variable.

Alleviation

The project team acknowledged this issue and decided to keep no change.

I-04: No Check of Address Params with Zero Address

Informational: Optimization Suggestion

File Location: /contracts/CarvNft.sol:76

Description

The input parameter of the address type in the function does not use the zero address for verification.

/contracts/CarvNft.sol

```
74      }
75
76      function setRedeemAddress(address newRedeemAddress) external
        onlyOwner {
77          redeemAddress = newRedeemAddress;
78      }
```

Recommendation

It is recommended to perform zero address verification on the input parameters of the address type.

Alleviation

The project team acknowledged this issue and decided to keep no change.

I-05: Parameters Should Be Declared as Calldata

Informational: Optimization Suggestion

File Location:
/contracts/CarvNft.sol:64
/contracts/interfaces/ICarvNft.sol:49

Description

When the compiler parses the external or public function, it can directly read the function parameters from calldata. Setting it to other storage locations may waste gas. About 300-400 gas can be saved with optimization turned off while 120-150 gas can be saved vice versa.

/contracts/CarvNft.sol

```
62      }
63
64      function setBaseURI(string memory newBaseURI) external onlyOwner {
65          baseURI = newBaseURI;
66      }
```

/contracts/interfaces/ICarvNft.sol

```
47       * @param newBaseURI: newBaseURI
48       */
49      function setBaseURI(string memory newBaseURI) external;
50
51      /**
```

Recommendation

In external or public functions, the storage location of function parameters should be set to calldata to save gas.

Alleviation

Fixed in commit a328463.

I-06: Floating Pragma

Informational: Optimization Suggestion

File Location:
/contracts/CarvNft.sol:2
/contracts/interfaces/ICarvNft.sol:2

Description

Contracts should be deployed with fixed compiler version which has been tested thoroughly or make sure to lock the contract compiler version in the project configuration. Locked compiler version ensures that contracts will not be compiled by untested compiler version.

/contracts/CarvNft.sol

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity ^0.8.20;
3
4  import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
```

/contracts/interfaces/ICarvNft.sol

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity ^0.8.20;
3
4  interface ICarvNft {
```

Recommendation

Use a fixed compiler version, and consider whether the bugs in the selected compiler version (https://github.com/ethereum/solidity/releases) will affect the contract.

Alleviation

Fixed in commit a328463.

I-07: Recommend to Follow Code Layout Conventions

Informational: Optimization Suggestion

File Location: /contracts/CarvNft.sol:8

Description

In the solidity document (https://docs.soliditylang.org/en/v0.8.17/style-guide.html), there are the following conventions for code layout:

Layout contract elements in the following order: 1. Pragma statements, 2. Import statements, 3. Interfaces, 4. Libraries, 5. Contracts.

Inside each contract, library or interface, use the following order: 1. Type declarations, 2. State variables, 3. Events, 4. Modifiers, 5. Functions.

Functions should be grouped according to their visibility and ordered: 1. constructor, 2. receive function (if exists), 3. fallback function (if exists), 4. external, 5. public, 6. internal, 7. private.

/contracts/CarvNft.sol

```
6   import "./interfaces/ICarvNft.sol";
7
8   contract CarvNft is ICarvNft, ERC721, Ownable {
9
10      uint256 constant MAX_SUPPLY = 100000;
```

Recommendation

It is recommended to follow the above code layout conventions to improve code readability.

Alleviation

Fixed in commit a328463.

4. Disclaimer

No description, statement, recommendation or conclusion in this report shall be construed as endorsement, affirmation or confirmation of the project. The security assessment is limited to the scope of work as stipulated in the Statement of Work.

This report is prepared in response to source code, and based on the attacks and vulnerabilities in the source code that already existed or occurred before the date of this report, excluding any new attacks or vulnerabilities that exist or occur after the date of this report. The security assessment are solely based on the documents and materials provided by the customer, and the customer represents and warrants documents and materials are true, accurate and complete.

CONSULTANT DOES NOT MAKE AND HEREBY DISCLAIMS ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, REGARDING THE SERVICES, DELIVERABLES, OR ANY OTHER MATTER PERTAINING TO THIS REPORT.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR AND HEREBY DISCLAIMS MERCHANTABILITY, FITNESS FOR PURPOSE, TITLE, NON-INFRINGEMENT OR NON-APPROPRIATION OF INTELLECTUAL PROPERTY RIGHTS OF A THIRD PARTY, SATISFACTORY QUALITY, ACCURACY, QUALITY, COMPLETENESS, TIMELINESS, RESPONSIVENESS, OR PRODUCTIVITY OF THE SERVICES OR DELIVERABLES.

CONSULTANT EXCLUDES ANY WARRANTY THAT THE SERVICES AND DELIVERABLES WILL BE UNINTERRUPTED, ERROR FREE, FREE OF SECURITY DEFECTS OR HARMFUL COMPONENTS, REVEAL ALL SECURITY VULNERABILITIES, OR THAT ANY DATA WILL NOT BE LOST OR CORRUPTED.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR (A) ANY REPRESENTATIONS MADE BY ANY PERSON REGARDING THE SUFFICIENCY OR SUITABILITY OF SERVICES AND DELIVERABLES IN ANY ACTUAL APPLICATION, OR (B) WHETHER ANY SUCH USE WOULD VIOLATE OR INFRINGE THE APPLICABLE LAWS, OR (C) REVIEWING THE CUSTOMER MATERIALS FOR ACCURACY.

5. Appendix

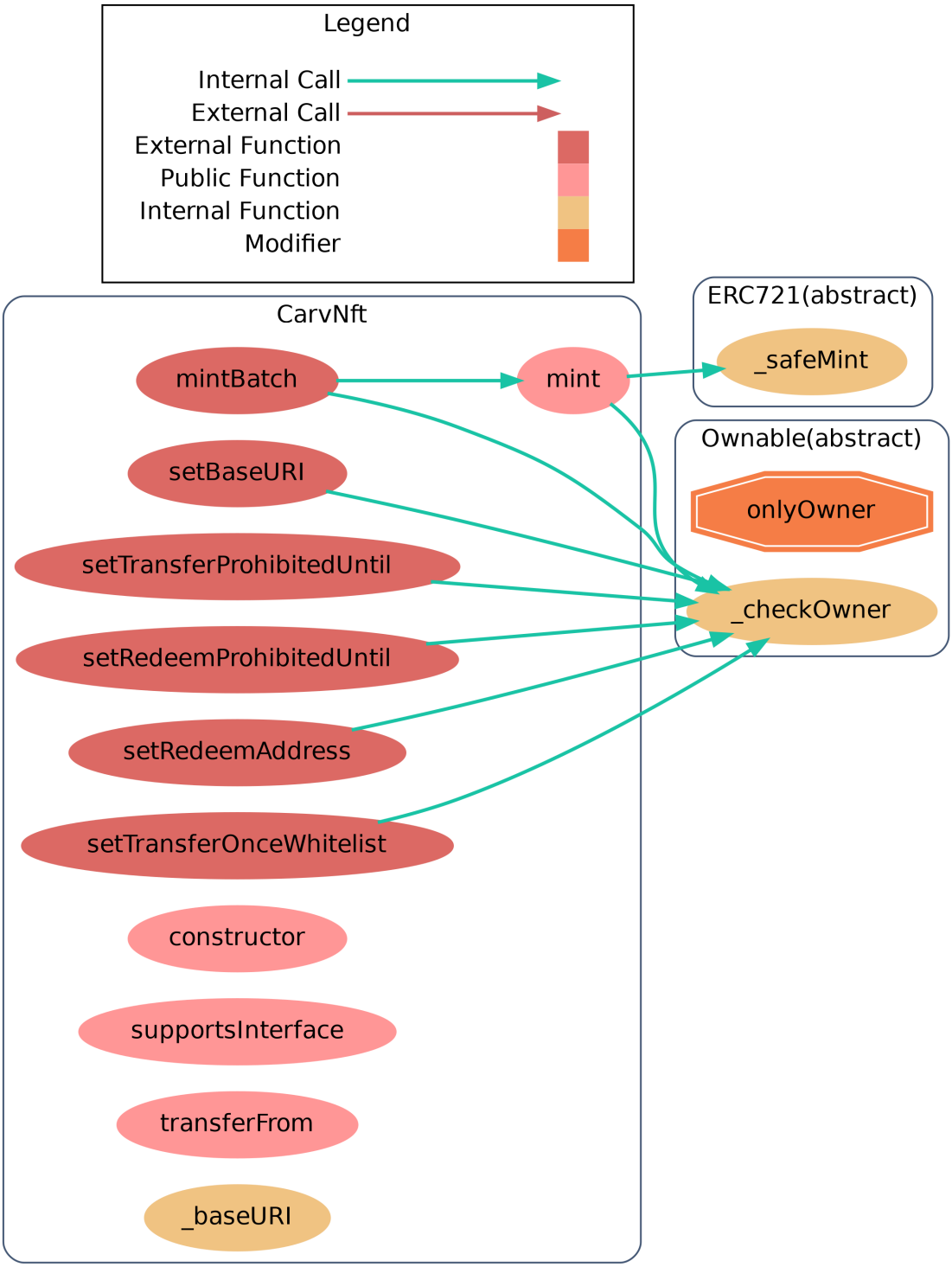
5.1 Visibility

Contract	FuncName	Visibility	Mutability	Modifiers
CarvNft	_CTOR_	public	N	
CarvNft	supportsInterface	public	N	
CarvNft	_baseURI	internal	N	
CarvNft	transferFrom	public	Y	
CarvNft	mint	public	Y	onlyOwner
CarvNft	mintBatch	external	Y	onlyOwner
CarvNft	setBaseURI	external	Y	onlyOwner
CarvNft	setTransferProhibitedUntil	external	Y	onlyOwner
CarvNft	setRedeemProhibitedUntil	external	Y	onlyOwner
CarvNft	setRedeemAddress	external	Y	onlyOwner
CarvNft	setTransferOnceWhitelist	external	Y	onlyOwner

5. Appendix

5.2 Call Graph

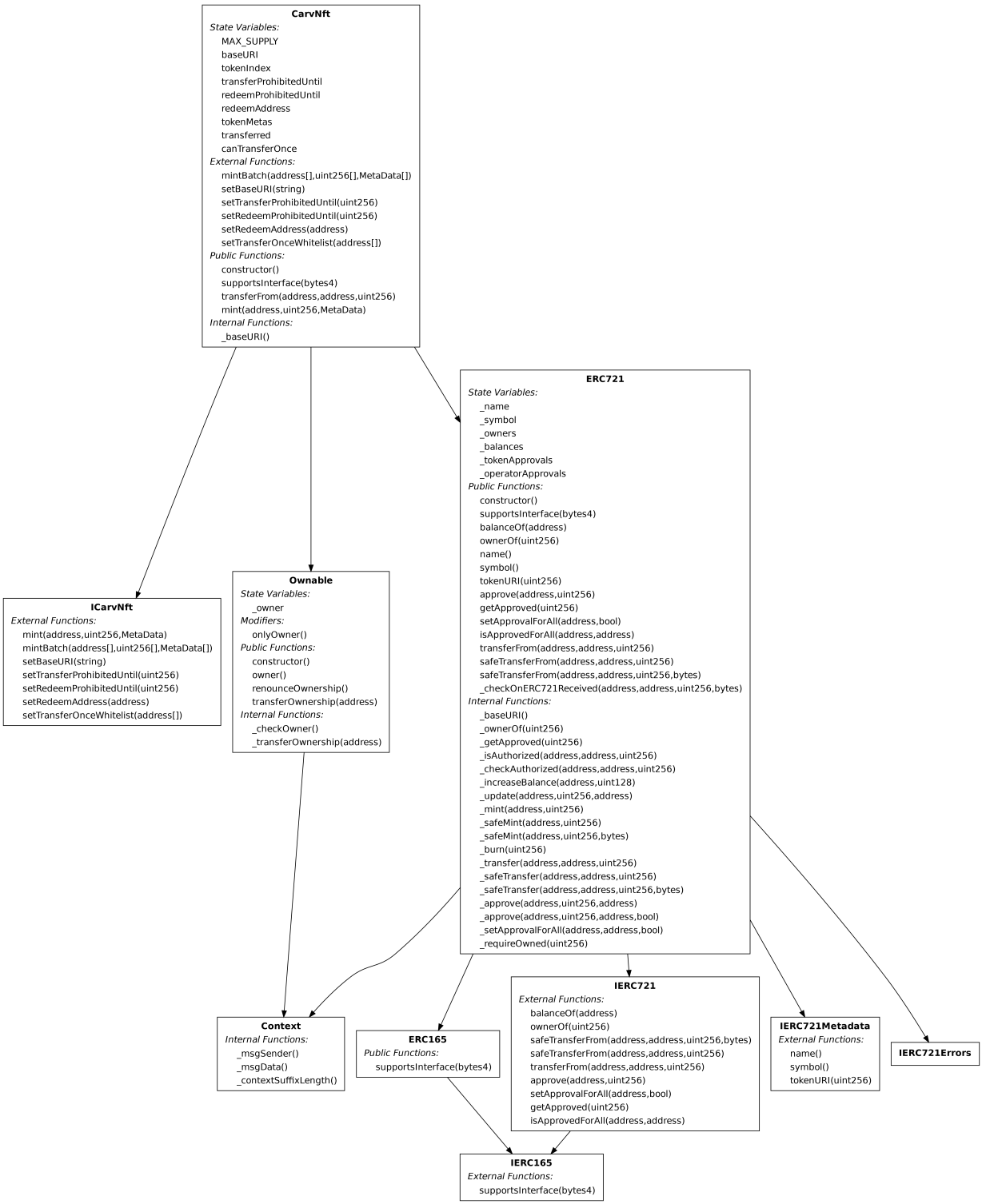
CarvNft



5. Appendix

5.3 Inheritance Graph

CarvNft



5.4 Formal Verification Metadata

1. When transferring a token to an address other than `redeemAddress`, the transaction should fail if the block timestamp is before `transferProhibitedUntil` and the sender is not allowed a one-time transfer.

```
/// #if_succeeds {:msg "When transferring a token to an address other than
`redeemAddress`, the transaction should fail if the block timestamp is
before `transferProhibitedUntil` and the sender is not allowed a one-time
transfer."} always((to != redeemAddress && block.timestamp <
transferProhibitedUntil) ==> (previously(canTransferOnce[from]) &&
previously(!transferred[tokenId])));
function transferFrom(address from, address to, uint256 tokenId) public
override {
```

Passed.

2. When transferring a token to the redeemAddress, the block timestamp should after `redeemProhibitedUntil`.

```
/// #if_succeeds {:msg "When transferring a token to the redeemAddress, the
block timestamp should after `redeemProhibitedUntil`."} always((to ==
redeemAddress) ==> (block.timestamp > redeemProhibitedUntil));
function transferFrom(address from, address to, uint256 tokenId) public
override {
```

Passed.

3. If the transfer is performed to an address other than `redeemAddress` during the prohibited period, and the sender is allowed a one-time transfer, the `transferred` mapping should mark the token as transferred.

```
/// #if_succeeds {:msg "If the transfer is performed to an address other
than `redeemAddress` during the prohibited period, and the sender is
allowed a one-time transfer, the `transferred` mapping should mark the
token as transferred."} always((to != redeemAddress && block.timestamp <
transferProhibitedUntil && previously(canTransferOnce[from])) ==>
transferred[tokenId]);
function transferFrom(address from, address to, uint256 tokenId) public
override {
```

Passed.

4. If the transfer timestamp is past the `transferProhibitedUntil` period, the transfer should succeed without any checks on `canTransferOnce` or `transferred`.

```
/// #if_succeeds {:msg "If the transfer timestamp is past the
`transferProhibitedUntil` period, the transfer should succeed without any
```

```
checks on `canTransferOnce` or `transferred`."} always((block.timestamp >=
transferProhibitedUntil) ==> true);
function transferFrom(address from, address to, uint256 tokenId) public
override {
```

Passed.

5. Upon successful execution of the `mint` function, the minted tokens' indices should reside between the original `tokenIndex` and the new `tokenIndex`.

```
/// #if_succeeds {:msg "Upon successful execution of the `mint` function,
the minted tokens' indices should reside between the original `tokenIndex`
and the new `tokenIndex`."} always(forall (uint256 i in
previously(tokenIndex + 1) .. tokenIndex) tokenMetas[i].code.length != 0);
function mint(address receiver, uint256 count, MetaData calldata meta)
public onlyOwner {
```

Passed.

6. The `tokenIndex` should be incremented by `count` after successfully executing the `mint` function.

```
/// #if_succeeds {:msg "The `tokenIndex` should be incremented by `count`
after successfully executing the `mint` function."} always(tokenIndex ==
previously(tokenIndex) + count);
function mint(address receiver, uint256 count, MetaData calldata meta)
public onlyOwner {
```

Passed.

7. The total supply of tokens after execution should not exceed MAX_SUPPLY.

```
/// #if_succeeds {:msg "The total supply of tokens after execution should
not exceed MAX_SUPPLY."} always(previously(tokenIndex) + sum(counts) ==
tokenIndex);
function mintBatch(address[] calldata receivers, uint256[] calldata counts,
MetaData[] calldata metas) external onlyOwner {
```

Passed.

8. The lengths of the input arrays 'receceiver', 'counts', and 'metas' must remain equal.

```
/// #if_succeeds {:msg "The lengths of the input arrays 'receceiver',
'counts', and 'metas' must remain equal."} always(receivers.length ==
counts.length && counts.length == metas.length);
```

```
function mintBatch(address[] calldata receivers, uint256[] calldata counts,
MetaData[] calldata metas) external onlyOwner {
```

Passed.

9. Token balances may solely be updated through invoking the functions `transferFrom`, `mint` or `mintBatch`.

```
/// #if_updated {:msg "Token balances may solely be updated through
invoking the functions `transferFrom`, `mint` or `mintBatch`."}
always($function == transferFrom(address,address,uint256) || $function ==
mint(address,uint256,MetaData) || $function ==
mintBatch(address[],uint256[],MetaData[]));
mapping(address owner => uint256) private _balances;
```

Violated. Please refer to H-01.