



AUDIT REPORT

PRODUCED BY CERTIK

FOR



23RD DEC, 2019

CERTIK AUDIT REPORT FOR SKYPEOPLE



Request Date: 2019-10-30
Revision Date: 2019-12-23
Platform Name: Ethereum



Contents

Disclaimer	1
About CertiK	2
Executive Summary	3
Vulnerability Classification	3
Testing Summary	4
Audit Score	4
Type of Issues	4
Vulnerability Details	5
Manual Review Notes	6
Static Analysis Results	15
Formal Verification Results	16
How to read	16
Source Code with CertiK Labels	149

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and SkyPeople (the “Company”), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the “Agreement”). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK’s prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: <https://certik.org/>

Executive Summary

This report has been prepared for SkyPeople to discover issues and vulnerabilities in the source code of their MineralNFTMarket, MineralNFT, Mineral, Counters, Context, SafeMath, ERC20, ERC20Burnable, ERC165, ERC721, ERC721Enumerable, ERC721Metadata, ERC1132, IERC20, IERC20Receiver, IERC165, IERC721, IERC721Enumerable, IERC721Metadata, IERC721Receiver, ERC721Full, IERC721Full, SafeERC20, Address, BytesLib and Ownable smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Vulnerability Classification

CertiK categorizes issues into three buckets based on overall risk levels:

Critical

Code implementation does not match specification, which could result in the loss of funds for contract owner or users.

Medium

Code implementation does not match the specification under certain conditions, which could affect the security standard by loss of access control.

Low

Code implementation does not follow best practices, or uses suboptimal design patterns, which could lead to security vulnerabilities further down the line.

Testing Summary

PASS

CERTIK believes this
smart contract passes security
qualifications to be listed on
digital asset exchanges.

Dec 23, 2019



Type of Issues

CertiK's smart label engine applied 100% formal verification coverage on the source code. Our team of engineers has scanned the source code using proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues	SWC ID
Integer Overflow/Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	1	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Using block attributes to generate random numbers is unreliable, as they can be influenced by miners to some degree.	0	SWC-120
"tx.origin" for Authorization	tx.origin should not be used for authorization. msg.sender instead.	Use 0	SWC-115

Title	Description	Issues	SWC ID
Delegatecall to Untrusted Callee	Calling untrusted contracts is very dangerous, so the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default, meaning a malicious user can make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The <code>assert()</code> function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used.	0	SWC-111
Unused Variables	Unused variables reduce code quality	0	SWC-131

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Manual Review Notes

Review Details

Summary

CertiK was chosen by SkyPeople to audit the design and implementation of its soon to be released smart contract. To ensure comprehensive protection, the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.

Refer to [White Paper](#):

MNR is the primary currency of the Mineral Hub ecosystem. It will act as a primary currency across multiple games, allowing players to buy and sell in-game items as non-fungible tokens.

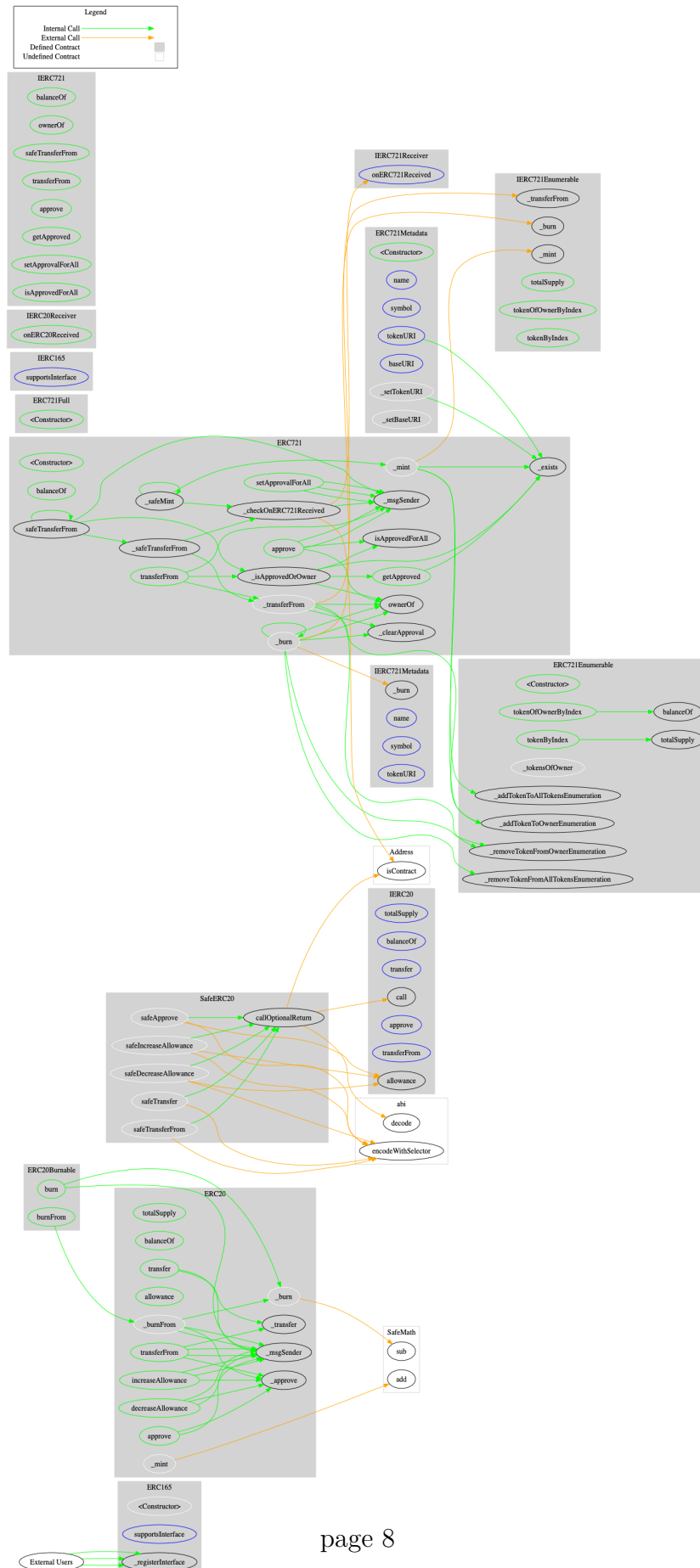
Source Code SHA-256 Checksum

- **MineralNFTMarket.sol**
81deb468a0c5d72679190c0737735f78d0254662c9bb58a0106cd31a4b73ce12
- **Counters.sol**
9bf7f72914f5d87d3f10e51b077ba4ef591330be2ce2f20f5eebea629c2c22ba
- **Context.sol**
48e51cda94a082cd59a2cb1309af279ba74fa061532ed4e2c191a25b68ad671b
- **SafeMath.sol**
ed701033e29cd4d639b0d8ab13c2e1554f6b4817e692b62c3dc269470a508d94
- **Mineral.sol**
b0d2c367c9f47b3906090cd5fba6d2406994194b5b39c9f7a60f7704433349ba
- **MineralNFT.sol**
1e56a06bab0dadac96f928dd99c086605b7de8db18ecc10ddc9de4c229afb9e1
- **ERC1132.sol**
e1b9bef8bd03022d3239bff75df220fe9a5cd156e5922024659167428a2d07ee
- **ERC165.sol**
9428389a4270beedb8a550aab3117a5e0ab5d6810ded1f37349dcaea9ce20848
- **ERC20.sol**
7af01925c2ef84f807d4a5c7d9bcb61a7f1650f22ff4dc70d57936ccbe6e0246
- **ERC20Burnable.sol**
47e1c6135c7215dfd9903f9145ce6ba263e0ebadd2ba84b3eaf4515eec5ff6ca
- **ERC721.sol**
ac454c8b1d9de703621af0cdbfaec6bc7f94b48ee74aab1aa18ac05908c8d3b0

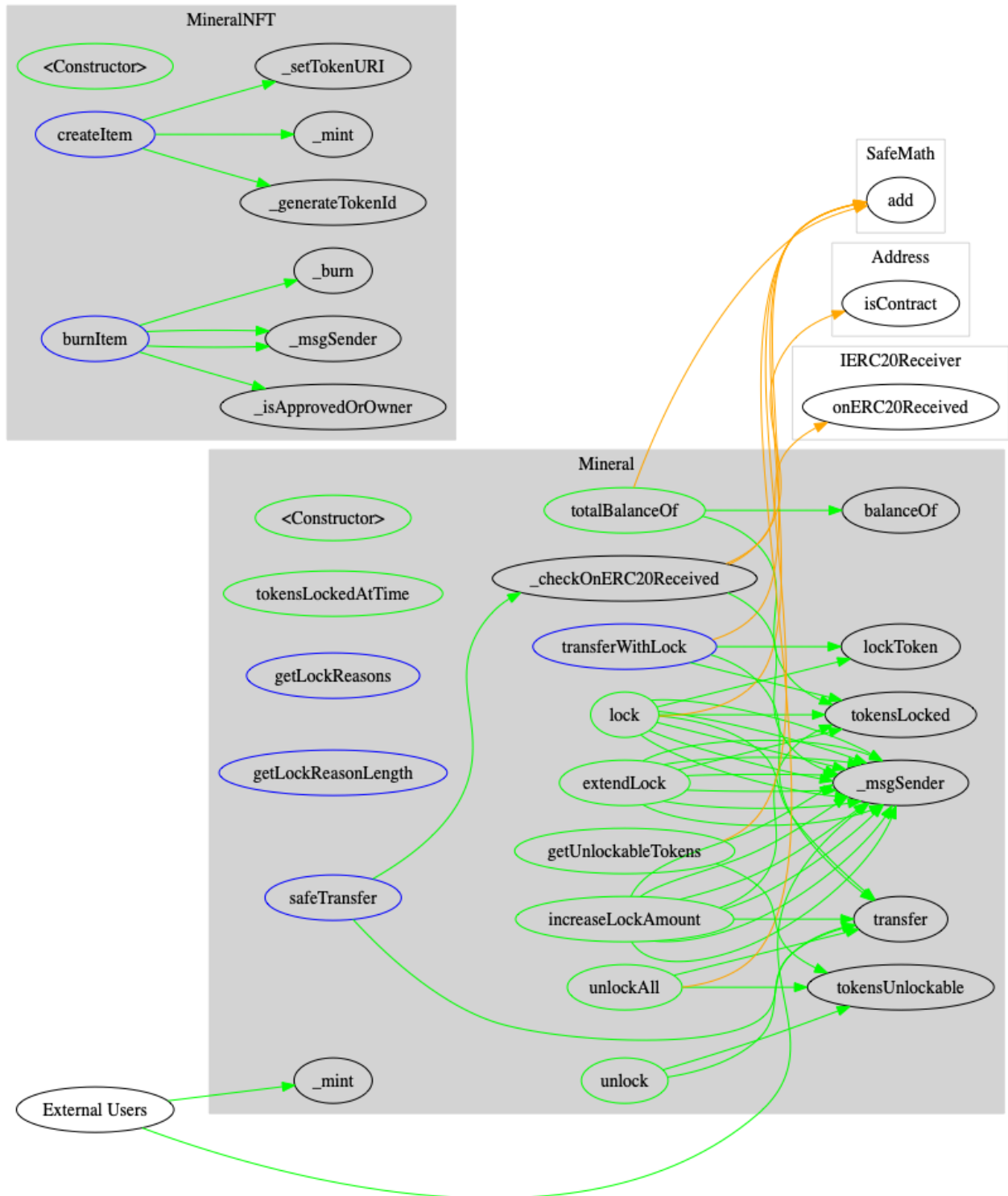
- **ERC721Enumerable.sol**
1084a0795b83499cfaa7d9549af34f5afd16ab3c504fdb50a8ad6b406ec86f1
- **ERC721Full.sol**
f4cc48196ca3c577a8d8f63475f909023277336fd49b11b056f632711c4100b8
- **ERC721Metadata.sol**
da24231ef6d1a912715caf4715a48d12deffede5d92c1d549261456a0a07b78f
- **IERC165.sol**
2c51255f992f9cb9af6f4a49258e1ea95461745f2793ad7bfa866fb3c32ef961
- **IERC20.sol**
a9125f9fed6949dfe63f0a681b611ba519042c54da222c00e97072895d690df2
- **IERC20Receiver.sol**
e9908f36e4aef61089720c9764a06e856bf988f853ded5727d30f30c18fc5f0c
- **IERC721.sol**
d24478871e68fe8017105c4bf07032033992220bfb8bfe667280b89a52f89a5f
- **IERC721Enumerable.sol**
b0b6915aaf8ee7f58bc04f68e4edb91ac3a2602fa9636a572319e478c977debe
- **IERC721Full.sol**
886bb7a12e412fde52cfe89672d71f14aa99910ac9f0765093f941d59f0608ac
- **IERC721Metadata.sol**
8971be3b30c2b5ee7473c6ff91531b3e42c74807f745947637fccc9d499f99b2
- **IERC721Receiver.sol**
9d682e2276bee3b73c4bce0547d3568890be97eea8084e5c6c67b8adce102a74
- **SafeERC20.sol**
22c89d5b7342d1f24d652f9462a8aed6c592d23f37db9c2f903e27f71de7e2e0
- **Address.sol**
47f17517d45f594e0c2aea25a7a7699fb11c9c146aeb276aaa03e4fb20ce068e
- **BytesLib.sol**
95da1be4fdf1a6eb1d35564f141a4b3073f312b580c374ed78ff56b181d103f4
- **Ownable.sol**
dda28aaf81b9acfec7d04ea23ff8dc04adbcee311a7406165766af727f691987

Design Architect

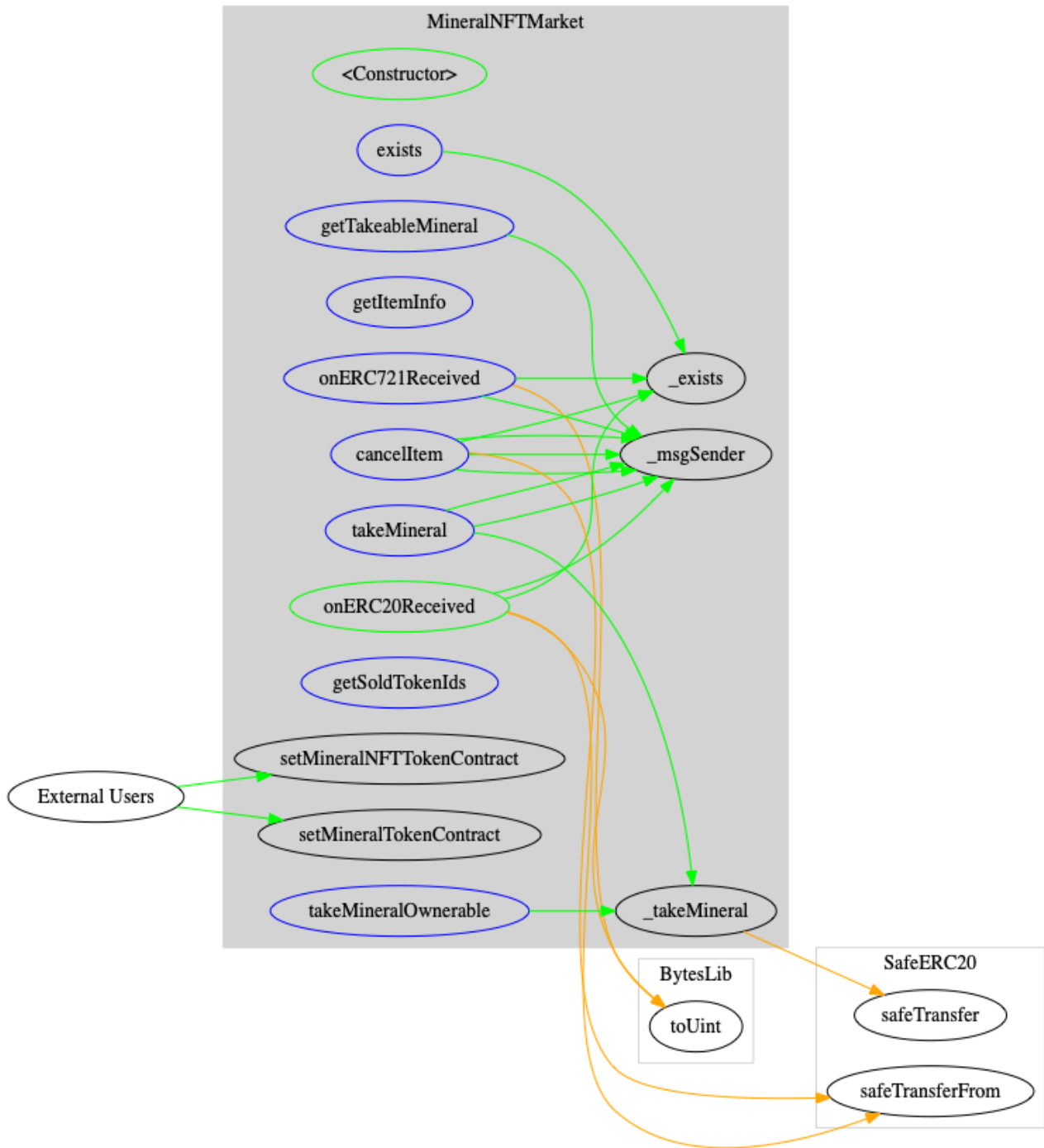
ERC Token Dependencies



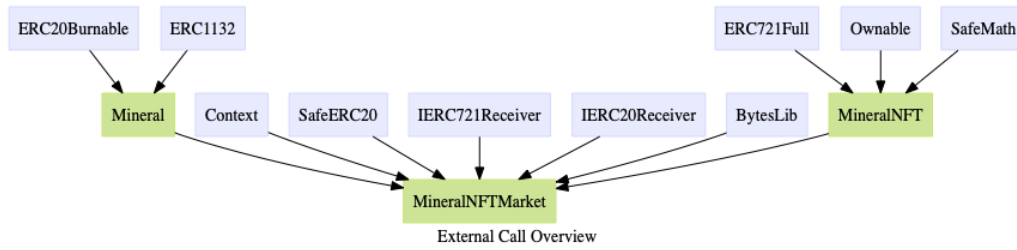
Mineral Dependencies



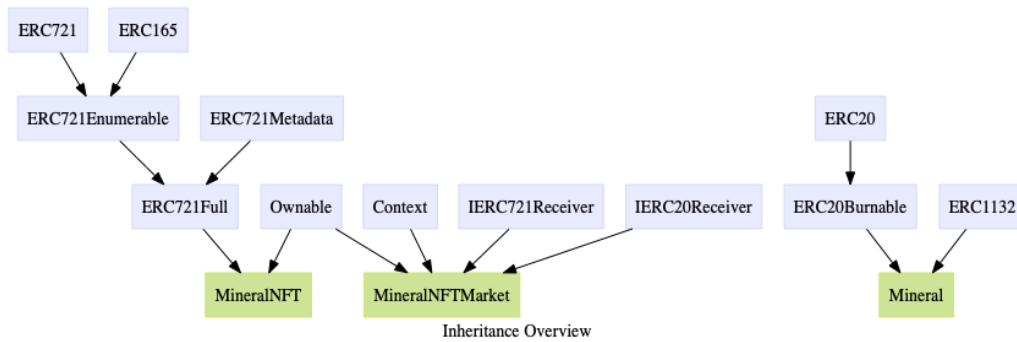
MineralNFTMarket Dependencies



Call Overview



Inheritance Overview



Recommendations

Changelog

- ✓ SkyPeople Upgraded to latest version of Openzeppelin contracts with usage of `safeTransfer()` and `safeTransferFrom()`.
- ✓ SkyPeople Added functions `getLockReasons()` and `getLockReasonLength()` as two getter functions, no security issues found.

Overview

- MINOR Recommend to use `safeTransfer()` and `safeTransferFrom()` when need to trigger transfer related logic.
 - ✓ SkyPeople The code is updated and reflected in the latest commit.
- MINOR Recommend to update to the latest and stable compiler version, or at least ensure the compiler versions of files in the project are the same.
 - ✓ SkyPeople The code is updated and reflected in the latest commit.
- MINOR Recommend to update to the latest version of ERC20 and ERC721 from Openzeppelin. Here is the diff comparing to [Openzeppelin](#) commit 33047ffddcc81ba7b0349431c5065c448603d098 and [EIP-1132 Proposer's](#) commit 82c68d3bd2d16c5ecda04d67b435ddaa69e5e7d4:
 - ERC1132.sol
 - * Derived from [EIP-1132 Proposer](#)
 - * SkyPeople: + `unlock()`
 - * ✓ SkyPeople This item is confirmed.
 - ERC20.sol
 - * Variables `_balances`, `_allowances` and `_totalSupply` are **public** in SkyPeople but **private** in openzeppelin.
 - * SkyPeople: + `safeTransfer()`
 - * ✓ SkyPeople The code is updated and reflected in the latest commit.
 - ERC721.sol
 - * SkyPeople: - `Counters` -> syntax diff for `increment()` or `decrement()`
 - * SkyPeople: - `_safeMint()`
 - * SkyPeople: - `_safeTransferFrom()`
 - * ✓ SkyPeople The code is updated and reflected in the latest commit.
 - ERC721Enumerable.sol
 - * SkyPeople: + `totalSupply()`
 - * ✓ SkyPeople This item is confirmed.

MineralNFTMarket.sol

- [MAJOR] Reentrancy in function `_takeMineral()`, `cancelItem()` and `onERC20Received()`. Recommend to use the [Check Effect Interaction Pattern](#) to protect the contract from being reentered.
 - ✓ [SkyPeople] The code is updated and reflected in the latest commit.
- [INFO] Variable `from` is unused in function `onERC721Received()`
 - ✓ [SkyPeople] The `from` parameter is derived from `IERC721Receiver` and the function is overriding `onERC721Received()`.
- [INFO] Local variable named `owner` shadows `Ownable.owner`.
- [INFO] Functions `onERC721Received()` and `onERC20Received()`: Recommend changing `public` to `external` to save gas.
- [INFO] Function `takeMineralOnwerable()`: Recommend renaming to `takeMineralOwnerable()`.
 - ✓ [SkyPeople] The code is updated and reflected in the latest commit.
- [INFO] Variable `selledTokenIds`: Recommend renaming to `soldTokenIds`.
 - ✓ [SkyPeople] The code is updated and reflected in the latest commit.
- [INFO] Struct `Item`: The various statuses should be enumerated for readability and used wherever you set statuses:


```
enum Status{enabled, sold, cancelled}
```

 - ✓ [SkyPeople] The code is updated and reflected in the latest commit.
- [INFO] Recommend changing wordings of error messages in `require`:
 - `require` same token `address` can be changed to `msg.sender is not nft token address`.
 - * ✓ [SkyPeople] The code is updated and reflected in the latest commit.
 - `exists item` can be changed to `item with input tokenId is existing`.
 - * ✓ [SkyPeople] The code is updated and reflected in the latest commit.
 - `require 0 < price` can be changed to `input price is not valid`.
 - * ✓ [SkyPeople] The code is updated and reflected in the latest commit.

/token/Mineral.sol

- [MINOR] Token name is set to be `"Mienral"`.
 - ✓ [SkyPeople] The code is updated and reflected in the latest commit.

- **INFO** `INITIAL_SUPPLY` initialization has many digits. Consider using scientific notation.
 - ✓ **SkyPeople** The code is updated and reflected in the latest commit.
- **INFO** Functions `lock()`, `tokensLockedAtTime()`, `totalBalanceOf()`, `extendLock()`, `increaseLockAmount()`, `unlockAll()`, `unlock()`, `getUnlockableTokens()` and `transferWithLock()`: Recommend changing `public` to `external` to save gas.
 - ✓ **SkyPeople** `transferWithLock()` is updated.
- **INFO** Recommend to initialize `locked` and `lockedReason` are mapped in `ERC1132.sol` for clarity.
 - ✓ **SkyPeople** This item is confirmed with SkyPeople that they aware the risk of using external library, although the risk is low on the this incident.
- **DISCUSSION** `Mineral` does not inherit from `Ownable`, so users couldn't directly give each other MNR in game. Is this intended functionality?
 - ✓ **SkyPeople** This item is confirmed.

`/token/MineralNFT.sol`

- **INFO** Function `createItem()`: Recommend changing `public` to `external` to save gas.
 - ✓ **SkyPeople** The code is updated and reflected in the latest commit.
- **MAJOR** When `id` is set to be the value of total supply, there is a possibility that the `createItem()` would never succeed, which might be considered as a Denial of Service.
 - ✓ **SkyPeople** The code is updated and reflected in the latest commit.
 - Approach:
 - * Let's say 10 NFTs exist, so they'll have tokenIDs 0-9.
 - * Now let's burn tokenIDs 3-5 so we're left with 7 total NFTs.
 - * When `createItem()` is called, the tokenID for this new NFT will be 7.
 - * Since an NFT with tokenID 7 already exists, minting will fail.
 - * Currently the issue is safe since all of the `_burn()` functions are internal and never being called. Recommend to add a similar logic like what `openzeppelin` does. Move the `id` of the latest valid element from the largest `id` value to the burned `id` value.

Static Analysis Results

TIMESTAMP_DEPENDENCY

Line 236 in File Mineral.sol

```
236      if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-  
          disable-line
```




! "now" can be influenced by miners to some degree

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	 20, Oct 2018
Verification timespan	 395.38 ms
CERTIK label location	Line 30-34 in File howtoread.sol
CERTIK label	<pre> 30 /*@CTK FAIL "transferFrom to same address" 31 @tag assume_completion 32 @pre from == to 33 @post __post.allowed[from][msg.sender] == 34 */ </pre>
Raw code location	Line 35-41 in File howtoread.sol
Raw code	<pre> 35 function transferFrom(address from, address to 36) { 37 balances[from] = balances[from].sub(tokens 38 allowed[from][msg.sender] = allowed[from][39 balances[to] = balances[to].add(tokens); 40 emit Transfer(from, to, tokens); 41 return true; 42 } </pre>
Counterexample	<div>  This code violates the specification </div> <div> <div> <div>Initial environment</div> <pre> 1 Counter Example: 2 Before Execution: 3 Input = { 4 from = 0x0 5 to = 0x0 6 tokens = 0x6c 7 } 8 This = 0 </pre> </div> <div> <pre> 52 } 53 balance: 0x0 54 } 55 } </pre> </div> </div> <div> <div>Post environment</div> <pre> 57 After Execution: 58 Input = { 59 from = 0x0 60 to = 0x0 61 tokens = 0x6c </pre> </div>

Formal Verification Request 1

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



239.03 ms

Line 41 in File MineralNFTMarket.sol

```
41 // @CTK_NO_BUF_OVERFLOW
```

Line 44-47 in File MineralNFTMarket.sol

```
44 constructor(address nft, address mineral) public {  
45     setMineralNFTTokenContract(nft);  
46     setMineralTokenContract(mineral);  
47 }
```

✓ The code meets the specification.

Formal Verification Request 2

If method completes, integer overflow would not happen.



23, Dec 2019



1.46 ms

Line 42 in File MineralNFTMarket.sol

```
42 // @CTK_NO_OVERFLOW
```

Line 44-47 in File MineralNFTMarket.sol

```
44 constructor(address nft, address mineral) public {  
45     setMineralNFTTokenContract(nft);  
46     setMineralTokenContract(mineral);  
47 }
```

✓ The code meets the specification.

Formal Verification Request 3

Method will not encounter an assertion failure.



23, Dec 2019



1.37 ms

Line 43 in File MineralNFTMarket.sol

```
43 // @CTK_NO_ASF
```

Line 44-47 in File MineralNFTMarket.sol

```
44 constructor(address nft, address mineral) public {  
45     setMineralNFTTokenContract(nft);  
46     setMineralTokenContract(mineral);  
47 }
```

✓ The code meets the specification.

Formal Verification Request 4

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



40.97 ms

Line 48 in File MineralNFTMarket.sol

```
48  // @CTK_NO_BUF_OVERFLOW
```

Line 55-57 in File MineralNFTMarket.sol

```
55  function exists(uint id) external view returns (bool) {  
56      return _exists(id);  
57  }
```

✓ The code meets the specification.

Formal Verification Request 5

If method completes, integer overflow would not happen.



23, Dec 2019



0.54 ms

Line 49 in File MineralNFTMarket.sol

```
49  // @CTK_NO_OVERFLOW
```

Line 55-57 in File MineralNFTMarket.sol

```
55  function exists(uint id) external view returns (bool) {  
56      return _exists(id);  
57  }
```

✓ The code meets the specification.

Formal Verification Request 6

Method will not encounter an assertion failure.



23, Dec 2019



0.59 ms

Line 50 in File MineralNFTMarket.sol

```
50  // @CTK_NO_ASF
```

Line 55-57 in File MineralNFTMarket.sol

```
55  function exists(uint id) external view returns (bool) {  
56      return _exists(id);  
57  }
```

✓ The code meets the specification.

Formal Verification Request 7

exists



23, Dec 2019



2.81 ms

Line 51-54 in File MineralNFTMarket.sol

```
51  /*@CTK exists
52     @post _items[id].price == 0 -> __return == false
53     @post _items[id].price != 0 -> __return == (_items[id].status == 0)
54  */
```

Line 55-57 in File MineralNFTMarket.sol

```
55  function exists(uint id) external view returns (bool) {
56      return _exists(id);
57  }
```

✓ The code meets the specification.

Formal Verification Request 8

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.42 ms

Line 58 in File MineralNFTMarket.sol

```
58  //@CTK NO_BUF_OVERFLOW
```

Line 65-70 in File MineralNFTMarket.sol

```
65  function _exists(uint id) internal view returns (bool) {
66      if (_items[id].price == 0)
67          return false;
68
69      return _items[id].status == uint8(ItemStatus.enable);
70  }
```

✓ The code meets the specification.

Formal Verification Request 9

If method completes, integer overflow would not happen.



23, Dec 2019



0.4 ms

Line 59 in File MineralNFTMarket.sol

```
59  //@CTK NO_OVERFLOW
```

Line 65-70 in File MineralNFTMarket.sol

```

65     function _exists(uint id) internal view returns (bool) {
66         if (_items[id].price == 0)
67             return false;
68
69         return _items[id].status == uint8(ItemStatus.enable);
70     }

```

✓ The code meets the specification.

Formal Verification Request 10

Method will not encounter an assertion failure.



23, Dec 2019



0.39 ms

Line 60 in File MineralNFTMarket.sol

```

60     //©TK NO_ASF

```

Line 65-70 in File MineralNFTMarket.sol

```

65     function _exists(uint id) internal view returns (bool) {
66         if (_items[id].price == 0)
67             return false;
68
69         return _items[id].status == uint8(ItemStatus.enable);
70     }

```

✓ The code meets the specification.

Formal Verification Request 11

__exists



23, Dec 2019



1.7 ms

Line 61-64 in File MineralNFTMarket.sol

```

61     /*©TK __exists
62         @post _items[id].price == 0 -> __return == false
63         @post _items[id].price != 0 -> __return == (_items[id].status == 0)
64     */

```

Line 65-70 in File MineralNFTMarket.sol

```

65     function _exists(uint id) internal view returns (bool) {
66         if (_items[id].price == 0)
67             return false;
68
69         return _items[id].status == uint8(ItemStatus.enable);
70     }

```

✓ The code meets the specification.

Formal Verification Request 12

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



22.68 ms

Line 71 in File MineralNFTMarket.sol

```
71 // @CTK_NO_BUF_OVERFLOW
```

Line 77-79 in File MineralNFTMarket.sol

```
77 function getTakeableMineral() external view returns (uint256) {  
78     return _takeableMineral[_msgSender()];  
79 }
```

✓ The code meets the specification.

Formal Verification Request 13

If method completes, integer overflow would not happen.



23, Dec 2019



0.48 ms

Line 72 in File MineralNFTMarket.sol

```
72 // @CTK_NO_OVERFLOW
```

Line 77-79 in File MineralNFTMarket.sol

```
77 function getTakeableMineral() external view returns (uint256) {  
78     return _takeableMineral[_msgSender()];  
79 }
```

✓ The code meets the specification.

Formal Verification Request 14

Method will not encounter an assertion failure.



23, Dec 2019



0.47 ms

Line 73 in File MineralNFTMarket.sol

```
73 // @CTK_NO_ASF
```

Line 77-79 in File MineralNFTMarket.sol

```
77 function getTakeableMineral() external view returns (uint256) {  
78     return _takeableMineral[_msgSender()];  
79 }
```

✓ The code meets the specification.

Formal Verification Request 15

getTakeableMineral



23, Dec 2019



0.66 ms

Line 74-76 in File MineralNFTMarket.sol

```
74  /*@CTK getTakeableMineral
75  @post !__reverted -> __return == _takeableMineral[msg.sender]
76  */
```

Line 77-79 in File MineralNFTMarket.sol

```
77  function getTakeableMineral() external view returns (uint256) {
78      return _takeableMineral[_msgSender()];
79  }
```

✓ The code meets the specification.

Formal Verification Request 16

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



8.52 ms

Line 80 in File MineralNFTMarket.sol

```
80  //@CTK NO_BUF_OVERFLOW
```

Line 86-88 in File MineralNFTMarket.sol

```
86  function getItemInfo(uint256 tokenId) external view returns (uint256 price, address
      owner, uint8 status) {
87      return (_items[tokenId].price, _items[tokenId].owner, _items[tokenId].status);
88  }
```

✓ The code meets the specification.

Formal Verification Request 17

If method completes, integer overflow would not happen.



23, Dec 2019



0.4 ms

Line 81 in File MineralNFTMarket.sol

```
81  //@CTK NO_OVERFLOW
```

Line 86-88 in File MineralNFTMarket.sol

```
86  function getItemInfo(uint256 tokenId) external view returns (uint256 price, address
      owner, uint8 status) {
87      return (_items[tokenId].price, _items[tokenId].owner, _items[tokenId].status);
88  }
```

✓ The code meets the specification.

Formal Verification Request 18

Method will not encounter an assertion failure.



23, Dec 2019



0.39 ms

Line 82 in File MineralNFTMarket.sol

82 `//@CTK NO_ASF`

Line 86-88 in File MineralNFTMarket.sol

```
86     function getItemInfo(uint256 tokenId) external view returns (uint256 price, address
      owner, uint8 status) {
87         return (_items[tokenId].price, _items[tokenId].owner, _items[tokenId].status);
88     }
```



The code meets the specification.

Formal Verification Request 19

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



125.67 ms

Line 91 in File MineralNFTMarket.sol

91 `//@CTK NO_BUF_OVERFLOW`

Line 101-116 in File MineralNFTMarket.sol

```
101     function onERC721Received(address operator, address from, uint256 tokenId, bytes
      calldata data) external returns (bytes4) {
102         require (_msgSender() == address(_nft), "msg.sender is not nft token address");
103         require (_exists(tokenId) == false, "item with input tokenId is existing");
104         uint256 price = data.toUint(0);
105         require (0 < price, "input price is not valid");
106         _items[tokenId] = Item({
107             id: tokenId,
108             price: price,
109             owner: operator,
110             status: uint8(ItemStatus.enable)
111         });
112         emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
113         return _ERC721_RECEIVED;
114     }
```



The code meets the specification.

Formal Verification Request 20

If method completes, integer overflow would not happen.



23, Dec 2019



3.86 ms

Line 92 in File MineralNFTMarket.sol

```
92  // @CTK_NO_OVERFLOW
```

Line 101-116 in File MineralNFTMarket.sol

```
101  function onERC721Received(address operator, address from, uint256 tokenId, bytes
      calldata data) external returns (bytes4) {
102      require (_msgSender() == address(_nft), "msg.sender is not nft token address");
103      require (_exists(tokenId) == false, "item with input tokenId is existing");
104      uint256 price = data.toUint(0);
105      require (0 < price, "input price is not valid");
106      _items[tokenId] = Item({
107          id: tokenId,
108          price: price,
109          owner: operator,
110          status: uint8(ItemStatus.enable)
111      });
112      emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
113      return _ERC721_RECEIVED;
114  }
```

✓ The code meets the specification.

Formal Verification Request 21

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 4.03 ms

Line 93 in File MineralNFTMarket.sol

```
93  // @CTK_NO_ASF
```

Line 101-116 in File MineralNFTMarket.sol

```
101  function onERC721Received(address operator, address from, uint256 tokenId, bytes
      calldata data) external returns (bytes4) {
102      require (_msgSender() == address(_nft), "msg.sender is not nft token address");
103      require (_exists(tokenId) == false, "item with input tokenId is existing");
104      uint256 price = data.toUint(0);
105      require (0 < price, "input price is not valid");
106      _items[tokenId] = Item({
107          id: tokenId,
108          price: price,
109          owner: operator,
110          status: uint8(ItemStatus.enable)
111      });
112      emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
113      return _ERC721_RECEIVED;
114  }
```

✓ The code meets the specification.

Formal Verification Request 22

onERC721Received



23, Dec 2019



4.03 ms

Line 94-100 in File MineralNFTMarket.sol

```

94      /*@CTK onERC721Received
95      @tag assume_completion
96      @pre msg.sender == address(_nft)
97      @pre _items[tokenId].price == 0 || (_items[tokenId].price != 0 && _items[tokenId].
          status != 0)
98      // @pre 0 < price
99      @post __return == 0x150b7a02
100     */

```

Line 101-116 in File MineralNFTMarket.sol

```

101     function onERC721Received(address operator, address from, uint256 tokenId, bytes
        calldata data) external returns (bytes4) {
102         require (_msgSender() == address(_nft), "msg.sender is not nft token address");
103         require (_exists(tokenId) == false, "item with input tokenId is existing");
104         uint256 price = data.toUint(0);
105         require (0 < price, "input price is not valid");
106         _items[tokenId] = Item({
107             id: tokenId,
108             price: price,
109             owner: operator,
110             status: uint8(ItemStatus.enable)
111         });
112         emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
113         return _ERC721_RECEIVED;
114     }

```

✓ The code meets the specification.

Formal Verification Request 23

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



190.39 ms

Line 119 in File MineralNFTMarket.sol

```

119     // @CTK NO_BUF_OVERFLOW

```

Line 132-157 in File MineralNFTMarket.sol

```

132     function onERC20Received(address from, uint256 amount, bytes memory data) public returns
        (bool) {
133         require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
            ");
134
135         uint256 id = data.toUint(0);
136         require (_exists(id), "item with input tokenId is existing");
137     }

```

```

138     Item storage item = _items[id];
139
140     require (item.price == amount, "input amount is not valid");
141     require (from != item.owner, "input buyer is not valid");
142     require (item.status == 0, "item is not available");
143     //ctk start
144     _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
145     //ctk end
146     _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
147     _soldTokenIds[item.owner].push(id);
148     item.status = uint8(ItemStatus.sold);
149     _nft.safeTransferFrom(address(this), from, id);
150
151     emit BuyItem(item.owner, from, id, amount);
152     return true;
153 }

```

✓ The code meets the specification.

Formal Verification Request 24

If method completes, integer overflow would not happen.



23, Dec 2019



17.7 ms

Line 120 in File MineralNFTMarket.sol

```

120     // @CTK_NO_OVERFLOW

```

Line 132-157 in File MineralNFTMarket.sol

```

132     function onERC20Received(address from, uint256 amount, bytes memory data) public returns
        (bool) {
133         require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
            ");
134
135         uint256 id = data.toUint(0);
136         require (_exists(id), "item with input tokenId is existing");
137
138         Item storage item = _items[id];
139
140         require (item.price == amount, "input amount is not valid");
141         require (from != item.owner, "input buyer is not valid");
142         require (item.status == 0, "item is not available");
143         //ctk start
144         _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
145         //ctk end
146         _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
147         _soldTokenIds[item.owner].push(id);
148         item.status = uint8(ItemStatus.sold);
149         _nft.safeTransferFrom(address(this), from, id);
150
151         emit BuyItem(item.owner, from, id, amount);
152         return true;
153     }


```

✓ The code meets the specification.

Formal Verification Request 25

Method will not encounter an assertion failure.

 23, Dec 2019

 12.02 ms

Line 121 in File MineralNFTMarket.sol

121 `//@CTK NO_ASF`

Line 132-157 in File MineralNFTMarket.sol

```


132  function onERC20Received(address from, uint256 amount, bytes memory data) public returns
      (bool) {
133      require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
          ");
134
135      uint256 id = data.toUint(0);
136      require (_exists(id), "item with input tokenId is existing");
137
138      Item storage item = _items[id];
139
140      require (item.price == amount, "input amount is not valid");
141      require (from != item.owner, "input buyer is not valid");
142      require (item.status == 0, "item is not available");
143      //ctk start
144      _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
145      //ctk end
146      _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
147      _soldTokenIds[item.owner].push(id);
148      item.status = uint8(ItemStatus.sold);
149      _nft.safeTransferFrom(address(this), from, id);
150
151      emit BuyItem(item.owner, from, id, amount);
152      return true;
153  }
```

 The code meets the specification.

Formal Verification Request 26

onERC20Received

 23, Dec 2019

 0.57 ms

Line 122-131 in File MineralNFTMarket.sol

```

122  /*@CTK onERC20Received
123      @pre msg.sender == address(_mineral)
124      @pre msg.sender == address(_nft)
125      // @pre _items[id].price == 0 || (_items[id].price != 0 && _items[id].status != 0)
126      // @pre _items[id].price == amount
127      // @pre from != _items[id].owner
128      // @pre _items[id].status == 0
129      // @post __post._takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner
          ].add(amount)
```

```
130 // @post __post._items[id].status == 1
131 */
```

Line 132-157 in File MineralNFTMarket.sol

```
132 function onERC20Received(address from, uint256 amount, bytes memory data) public returns
    (bool) {
133     require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
        ");
134
135     uint256 id = data.toUint(0);
136     require (_exists(id), "item with input tokenId is existing");
137
138     Item storage item = _items[id];
139
140     require (item.price == amount, "input amount is not valid");
141     require (from != item.owner, "input buyer is not valid");
142     require (item.status == 0, "item is not available");
143     //ctk start
144     _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
145     //ctk end
146     _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
147     _soldTokenIds[item.owner].push(id);
148     item.status = uint8(ItemStatus.sold);
149     _nft.safeTransferFrom(address(this), from, id);
150
151     emit BuyItem(item.owner, from, id, amount);
152     return true;
153 }
```

✓ The code meets the specification.

Formal Verification Request 27

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



83.91 ms

Line 159 in File MineralNFTMarket.sol

```
159 //CTK NO_BUF_OVERFLOW
```

Line 167-181 in File MineralNFTMarket.sol

```
167 function cancelItem(uint256 tokenId) external {
168     require (_exists(tokenId), "item with input tokenId is existing");
169
170     Item item = _items[tokenId];
171
172     require (_msgSender() == item.owner, "msg.sender is not token owner");
173     require (item.status != uint8(ItemStatus.canceled), "item is already canceled");
174
175     item.status = uint8(ItemStatus.canceled);
176     _nft.safeTransferFrom(address(this), _msgSender(), tokenId);
177
178     emit CancelItem(_msgSender(), tokenId);
179 }
```

✓ The code meets the specification.

Formal Verification Request 28

If method completes, integer overflow would not happen.

📅 23, Dec 2019

🕒 3.35 ms

Line 160 in File MineralNFTMarket.sol

160 `//@CTK NO_OVERFLOW`

Line 167-181 in File MineralNFTMarket.sol

```
167     function cancelItem(uint256 tokenId) external {
168         require (_exists(tokenId), "item with input tokenId is existing");
169
170         Item item = _items[tokenId];
171
172         require (_msgSender() == item.owner, "msg.sender is not token owner");
173         require (item.status != uint8(ItemStatus.canceled), "item is already canceled");
174
175         item.status = uint8(ItemStatus.canceled);
176         _nft.safeTransferFrom(address(this), _msgSender(), tokenId);
177
178         emit CancelItem(_msgSender(), tokenId);
179     }
```

✓ The code meets the specification.

Formal Verification Request 29

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 3.47 ms

Line 161 in File MineralNFTMarket.sol

161 `//@CTK NO_ASF`

Line 167-181 in File MineralNFTMarket.sol

```
167     function cancelItem(uint256 tokenId) external {
168         require (_exists(tokenId), "item with input tokenId is existing");
169
170         Item item = _items[tokenId];
171
172         require (_msgSender() == item.owner, "msg.sender is not token owner");
173         require (item.status != uint8(ItemStatus.canceled), "item is already canceled");
174
175         item.status = uint8(ItemStatus.canceled);
176         _nft.safeTransferFrom(address(this), _msgSender(), tokenId);
177
178         emit CancelItem(_msgSender(), tokenId);
179     }
```


✓ The code meets the specification.

Formal Verification Request 30

cancelItem

📅 23, Dec 2019

🕒 0.42 ms

Line 162-166 in File MineralNFTMarket.sol

```
162  /*@CTK "cancelItem"
163      @pre _items[tokenId].price != 0 && _items[tokenId].price == 0 || _items[tokenId].
          status == 0
164      @pre msg.sender == _items[tokenId].owner
165      @pre _items[tokenId].status != 2
166  */
```

Line 167-181 in File MineralNFTMarket.sol

```
167  function cancelItem(uint256 tokenId) external {
168      require (_exists(tokenId), "item with input tokenId is existing");
169
170      Item item = _items[tokenId];
171
172      require (_msgSender() == item.owner, "msg.sender is not token owner");
173      require (item.status != uint8(ItemStatus.canceled), "item is already canceled");
174
175      item.status = uint8(ItemStatus.canceled);
176      _nft.safeTransferFrom(address(this), _msgSender(), tokenId);
177
178      emit CancelItem(_msgSender(), tokenId);
179  }
```

✓ The code meets the specification.

Formal Verification Request 31

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 106.32 ms

Line 183 in File MineralNFTMarket.sol

```
183  //@CTK NO_BUF_OVERFLOW
```

Line 192-195 in File MineralNFTMarket.sol

```
192  function takeMineral() external {
193      require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
          take");
194      _takeMineral(_msgSender());
195  }
```

✓ The code meets the specification.

Formal Verification Request 32

If method completes, integer overflow would not happen.



23, Dec 2019



0.93 ms

Line 184 in File MineralNFTMarket.sol

```
184  // @CTK_NO_OVERFLOW
```

Line 192-195 in File MineralNFTMarket.sol

```
192  function takeMineral() external {
193      require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
        take");
194      _takeMineral(_msgSender());
195  }
```



The code meets the specification.

Formal Verification Request 33

Method will not encounter an assertion failure.



23, Dec 2019



0.88 ms

Line 185 in File MineralNFTMarket.sol

```
185  // @CTK_NO_ASF
```

Line 192-195 in File MineralNFTMarket.sol

```
192  function takeMineral() external {
193      require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
        take");
194      _takeMineral(_msgSender());
195  }
```



The code meets the specification.

Formal Verification Request 34

takeMineral



23, Dec 2019



4.07 ms

Line 186-191 in File MineralNFTMarket.sol

```
186  /* @CTK_takeMineral
187      @pre 0 < _takeableMineral[msg.sender]
188      @pre _soldTokenIds[msg.sender].length > 0
189      @post __post._takeableMineral[msg.sender] == 0
190      @post __post._soldTokenIds[msg.sender].length == 0
191  */
```

Line 192-195 in File MineralNFTMarket.sol


```
192     function takeMineral() external {
193         require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
            take");
194         _takeMineral(_msgSender());
195     }
```

✓ The code meets the specification.

Formal Verification Request 35

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.52 ms

Line 197 in File MineralNFTMarket.sol

```
197     //@CTK NO_BUF_OVERFLOW
```

Line 205-217 in File MineralNFTMarket.sol


```
205     function _takeMineral(address addr) internal {
206         require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");
207
208         uint256 amount = _takeableMineral[addr];
209         uint256[] memory tokenIds = _soldTokenIds[addr];
210         _takeableMineral[addr] = 0;
211         _soldTokenIds[addr].length = 0;
212
213         _mineral.safeTransfer(addr, amount);
214         emit TakeMineral(addr, amount, tokenIds);
215     }
```

✓ The code meets the specification.

Formal Verification Request 36

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.48 ms

Line 198 in File MineralNFTMarket.sol

```
198     //@CTK NO_OVERFLOW
```

Line 205-217 in File MineralNFTMarket.sol

```
205     function _takeMineral(address addr) internal {
206         require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");
207
208         uint256 amount = _takeableMineral[addr];
209         uint256[] memory tokenIds = _soldTokenIds[addr];
210         _takeableMineral[addr] = 0;
211         _soldTokenIds[addr].length = 0;
```

```

212
213     _mineral.safeTransfer(addr, amount);
214     emit TakeMineral(addr, amount, tokenIds);
215 }


```

✓ The code meets the specification.

Formal Verification Request 37

Method will not encounter an assertion failure.

 23, Dec 2019

 0.47 ms

Line 199 in File MineralNFTMarket.sol

```

199 // @CTK NO_ASF

```

Line 205-217 in File MineralNFTMarket.sol

```

205 function _takeMineral(address addr) internal {
206     require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");
207
208     uint256 amount = _takeableMineral[addr];
209     uint256[] memory tokenIds = _soldTokenIds[addr];
210     _takeableMineral[addr] = 0;
211     _soldTokenIds[addr].length = 0;
212
213     _mineral.safeTransfer(addr, amount);
214     emit TakeMineral(addr, amount, tokenIds);
215 }


```

✓ The code meets the specification.

Formal Verification Request 38

_takeMineral

 23, Dec 2019

 2.39 ms

Line 200-204 in File MineralNFTMarket.sol

```

200 /* @CTK _takeMineral
201     @pre _soldTokenIds[addr].length > 0
202     @post __post._takeableMineral[addr] == 0
203     @post __post._soldTokenIds[addr].length == 0
204 */

```

Line 205-217 in File MineralNFTMarket.sol

```

205 function _takeMineral(address addr) internal {
206     require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");
207
208     uint256 amount = _takeableMineral[addr];
209     uint256[] memory tokenIds = _soldTokenIds[addr];
210     _takeableMineral[addr] = 0;

```

```
211     _soldTokenIds[addr].length = 0;
212
213     _mineral.safeTransfer(addr, amount);
214     emit TakeMineral(addr, amount, tokenIds);
215 }
```

✓ The code meets the specification.

Formal Verification Request 39

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 6.3 ms

Line 219 in File MineralNFTMarket.sol

```
219 // @CTK_NO_BUF_OVERFLOW
```

Line 225-227 in File MineralNFTMarket.sol

```
225 function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
226     return _soldTokenIds[addr];
227 }
```

✓ The code meets the specification.

Formal Verification Request 40

If method completes, integer overflow would not happen.

📅 23, Dec 2019

🕒 0.38 ms

Line 220 in File MineralNFTMarket.sol

```
220 // @CTK_NO_OVERFLOW
```

Line 225-227 in File MineralNFTMarket.sol

```
225 function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
226     return _soldTokenIds[addr];
227 }
```

✓ The code meets the specification.

Formal Verification Request 41

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.37 ms

Line 221 in File MineralNFTMarket.sol

```
221 // @CTK_NO_ASF
```

Line 225-227 in File MineralNFTMarket.sol


```
225     function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
226         return _soldTokenIds[addr];
227     }
```

✓ The code meets the specification.

Formal Verification Request 42

getSoldTokenIds

 23, Dec 2019

 0.39 ms

Line 222-224 in File MineralNFTMarket.sol

```
222     /*@CTK getSoldTokenIds
223         @post !__reverted -> __return == _soldTokenIds[addr]
224     */
```

Line 225-227 in File MineralNFTMarket.sol


```
225     function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
226         return _soldTokenIds[addr];
227     }
```

✓ The code meets the specification.

Formal Verification Request 43

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.65 ms

Line 228 in File MineralNFTMarket.sol

```
228     //@CTK NO_BUF_OVERFLOW
```

Line 231-233 in File MineralNFTMarket.sol


```
231     function setMineralNFTTokenContract(address addr) public onlyOwner {
232         _nft = MineralNFT(addr);
233     }
```

✓ The code meets the specification.

Formal Verification Request 44

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.64 ms

Line 229 in File MineralNFTMarket.sol

229 `//@CTK NO_OVERFLOW`

Line 231-233 in File MineralNFTMarket.sol


```
231     function setMineralNFTTokenContract(address addr) public onlyOwner {
232         _nft = MineralNFT(addr);
233     }
```

✓ The code meets the specification.

Formal Verification Request 45

Method will not encounter an assertion failure.

 23, Dec 2019

 0.61 ms

Line 230 in File MineralNFTMarket.sol

230 `//@CTK NO_ASF`

Line 231-233 in File MineralNFTMarket.sol


```
231     function setMineralNFTTokenContract(address addr) public onlyOwner {
232         _nft = MineralNFT(addr);
233     }
```

✓ The code meets the specification.

Formal Verification Request 46

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.63 ms

Line 234 in File MineralNFTMarket.sol

234 `//@CTK NO_BUF_OVERFLOW`

Line 237-239 in File MineralNFTMarket.sol


```
237     function setMineralTokenContract(address addr) public onlyOwner {
238         _mineral = IERC20(addr);
239     }
```

✓ The code meets the specification.

Formal Verification Request 47

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.63 ms

Line 235 in File MineralNFTMarket.sol

235 `//@CTK NO_OVERFLOW`

Line 237-239 in File MineralNFTMarket.sol


```
237 function setMineralTokenContract(address addr) public onlyOwner {
238     _mineral = IERC20(addr);
239 }
```

✓ The code meets the specification.

Formal Verification Request 48

Method will not encounter an assertion failure.

 23, Dec 2019

 0.62 ms

Line 236 in File MineralNFTMarket.sol

236 `//@CTK NO_ASF`

Line 237-239 in File MineralNFTMarket.sol


```
237 function setMineralTokenContract(address addr) public onlyOwner {
238     _mineral = IERC20(addr);
239 }
```

✓ The code meets the specification.

Formal Verification Request 49

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 48.5 ms

Line 242 in File MineralNFTMarket.sol

242 `//@CTK NO_BUF_OVERFLOW`

Line 249-251 in File MineralNFTMarket.sol


```
249 function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {
250     return _takeableMineral[addr];
251 }
```

✓ The code meets the specification.

Formal Verification Request 50

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.64 ms

Line 243 in File MineralNFTMarket.sol

243 `//@CTK NO_OVERFLOW`

Line 249-251 in File MineralNFTMarket.sol


```
249   function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {  
250       return _takeableMineral[addr];  
251   }
```

✓ The code meets the specification.

Formal Verification Request 51

Method will not encounter an assertion failure.

 23, Dec 2019

 0.65 ms

Line 244 in File MineralNFTMarket.sol

244 `//@CTK NO_ASF`

Line 249-251 in File MineralNFTMarket.sol


```
249   function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {  
250       return _takeableMineral[addr];  
251   }
```

✓ The code meets the specification.

Formal Verification Request 52

getTakeableMineral

 23, Dec 2019

 0.7 ms

Line 245-248 in File MineralNFTMarket.sol

```
245   /*@CTK getTakeableMineral  
246       @pre msg.sender == _owner  
247       @post !__reverted -> __return == _takeableMineral[addr]  
248   */
```

Line 249-251 in File MineralNFTMarket.sol


```
249   function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {  
250       return _takeableMineral[addr];  
251   }
```

✓ The code meets the specification.

Formal Verification Request 53

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 80.42 ms

Line 253 in File MineralNFTMarket.sol

253 `//@CTK NO_BUF_OVERFLOW`

Line 262-264 in File MineralNFTMarket.sol


```
262 function takeMineralOwnerable(address addr) external onlyOwner {
263     _takeMineral(addr);
264 }
```

 The code meets the specification.

Formal Verification Request 54

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.97 ms

Line 254 in File MineralNFTMarket.sol

254 `//@CTK NO_OVERFLOW`

Line 262-264 in File MineralNFTMarket.sol


```
262 function takeMineralOwnerable(address addr) external onlyOwner {
263     _takeMineral(addr);
264 }
```

 The code meets the specification.

Formal Verification Request 55

Method will not encounter an assertion failure.

 23, Dec 2019

 0.91 ms

Line 255 in File MineralNFTMarket.sol

255 `//@CTK NO_ASF`

Line 262-264 in File MineralNFTMarket.sol


```
262 function takeMineralOwnerable(address addr) external onlyOwner {
263     _takeMineral(addr);
264 }
```

 The code meets the specification.

Formal Verification Request 56

takeMineralOwnerable

 23, Dec 2019

 3.84 ms

Line 256-261 in File MineralNFTMarket.sol

```
256  /*@CTK takeMineralOwnerable
257  @pre msg.sender == _owner
258  @pre _soldTokenIds[addr].length > 0
259  @post __post._takeableMineral[addr] == 0
260  @post __post._soldTokenIds[addr].length == 0
261  */
```

Line 262-264 in File MineralNFTMarket.sol


```
262  function takeMineralOwnerable(address addr) external onlyOwner {
263      _takeMineral(addr);
264  }
```

 The code meets the specification.

Formal Verification Request 57

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 255.73 ms

Line 27 in File Mineral.sol

```
27  //@CTK NO_BUF_OVERFLOW
```

Line 35-57 in File Mineral.sol

```
35  function lock(bytes32 _reason, uint256 _amount, uint256 _time)
36  public
37  returns (bool)
38  {
39      uint256 validUntil = now.add(_time); //solhint-disable-line
40
41      // If tokens are already locked, then functions extendLock or
42      // increaseLockAmount should be used to make any changes
43      require(tokensLocked(msgSender(), _reason) == 0, ALREADY_LOCKED);
44      require(_amount != 0, AMOUNT_ZERO);
45
46      if (locked[msgSender()][_reason].amount == 0)
47          lockReason[msgSender()].push(_reason);
48
49      transfer(address(this), _amount);
50
51      locked[msgSender()][_reason] = lockToken(_amount, validUntil, false);
52
53      emit Locked(msgSender(), _reason, _amount, validUntil);
54      return true;
55  }
```

✓ The code meets the specification.

Formal Verification Request 58

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 13.61 ms

Line 28 in File Mineral.sol

28 `//@CTK NO_ASF`

Line 35-57 in File Mineral.sol

```

35  function lock(bytes32 _reason, uint256 _amount, uint256 _time)
36      public
37      returns (bool)
38  {
39      uint256 validUntil = now.add(_time); //solhint-disable-line
40
41      // If tokens are already locked, then functions extendLock or
42      // increaseLockAmount should be used to make any changes
43      require(tokensLocked(_msgSender(), _reason) == 0, ALREADY_LOCKED);
44      require(_amount != 0, AMOUNT_ZERO);
45
46      if (locked[_msgSender()][_reason].amount == 0)
47          lockReason[_msgSender()].push(_reason);
48
49      transfer(address(this), _amount);
50
51      locked[_msgSender()][_reason] = lockToken(_amount, validUntil, false);
52
53      emit Locked(_msgSender(), _reason, _amount, validUntil);
54      return true;
55  }

```

✓ The code meets the specification.

Formal Verification Request 59

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 92.58 ms

Line 67 in File Mineral.sol

67 `//@CTK NO_BUF_OVERFLOW`

Line 75-95 in File Mineral.sol

```

75  function transferWithLock(address _to, bytes32 _reason, uint256 _amount, uint256 _time)
76      external
77      returns (bool)
78  {
79      uint256 validUntil = now.add(_time); //solhint-disable-line

```

```

80
81     require(tokensLocked(_to, _reason) == 0, ALREADY_LOCKED);
82     require(_amount != 0, AMOUNT_ZERO);
83
84     if (locked[_to][_reason].amount == 0)
85         lockReason[_to].push(_reason);
86
87     transfer(address(this), _amount);
88
89     locked[_to][_reason] = lockToken(_amount, validUntil, false);
90
91     emit Locked(_to, _reason, _amount, validUntil);
92     return true;
93 }


```

✓ The code meets the specification.

Formal Verification Request 60

Method will not encounter an assertion failure.

 23, Dec 2019

 5.86 ms

Line 68 in File Mineral.sol

```
68 // @CTK NO_ASF
```

Line 75-95 in File Mineral.sol

```

75     function transferWithLock(address _to, bytes32 _reason, uint256 _amount, uint256 _time)
76         external
77         returns (bool)
78     {
79         uint256 validUntil = now.add(_time); //solhint-disable-line
80
81         require(tokensLocked(_to, _reason) == 0, ALREADY_LOCKED);
82         require(_amount != 0, AMOUNT_ZERO);
83
84         if (locked[_to][_reason].amount == 0)
85             lockReason[_to].push(_reason);
86
87         transfer(address(this), _amount);
88
89         locked[_to][_reason] = lockToken(_amount, validUntil, false);
90
91         emit Locked(_to, _reason, _amount, validUntil);
92         return true;
93     }


```

✓ The code meets the specification.

Formal Verification Request 61

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.44 ms

Line 104 in File Mineral.sol

104 `//@CTK NO_BUF_OVERFLOW`


Line 111-118 in File Mineral.sol

```
111     function tokensLocked(address _of, bytes32 _reason)
112     public
113     view
114     returns (uint256 amount)
115     {
116         if (!locked[_of][_reason].claimed)
117             amount = locked[_of][_reason].amount;
118     }
```

 The code meets the specification.

Formal Verification Request 62

If method completes, integer overflow would not happen.

 23, Dec 2019 0.4 ms

Line 105 in File Mineral.sol

105 `//@CTK NO_OVERFLOW`


Line 111-118 in File Mineral.sol

```
111     function tokensLocked(address _of, bytes32 _reason)
112     public
113     view
114     returns (uint256 amount)
115     {
116         if (!locked[_of][_reason].claimed)
117             amount = locked[_of][_reason].amount;
118     }
```

 The code meets the specification.

Formal Verification Request 63

Method will not encounter an assertion failure.

 23, Dec 2019 0.39 ms

Line 106 in File Mineral.sol

106 `//@CTK NO_ASF`

Line 111-118 in File Mineral.sol


```
111 function tokensLocked(address _of, bytes32 _reason)
112     public
113     view
114     returns (uint256 amount)
115 {
116     if (!locked[_of][_reason].claimed)
117         amount = locked[_of][_reason].amount;
118 }
```

✓ The code meets the specification.

Formal Verification Request 64

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 7.74 ms

Line 128 in File Mineral.sol

```
128 // @CTK_NO_BUF_OVERFLOW
```

Line 135-142 in File Mineral.sol


```
135 function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
136     public
137     view
138     returns (uint256 amount)
139 {
140     if (locked[_of][_reason].validity > _time)
141         amount = locked[_of][_reason].amount;
142 }
```

✓ The code meets the specification.

Formal Verification Request 65

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.44 ms

Line 129 in File Mineral.sol

```
129 // @CTK_NO_OVERFLOW
```

Line 135-142 in File Mineral.sol


```
135 function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
136     public
137     view
138     returns (uint256 amount)
139 {
140     if (locked[_of][_reason].validity > _time)
141         amount = locked[_of][_reason].amount;
142 }
```

✓ The code meets the specification.

Formal Verification Request 66

Method will not encounter an assertion failure.

 23, Dec 2019

 0.4 ms

Line 130 in File Mineral.sol

130 `//@CTK NO_ASF`

Line 135-142 in File Mineral.sol


```
135     function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
136     public
137     view
138     returns (uint256 amount)
139     {
140         if (locked[_of][_reason].validity > _time)
141             amount = locked[_of][_reason].amount;
142     }
```

 The code meets the specification.

Formal Verification Request 67

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 34.14 ms

Line 148 in File Mineral.sol

148 `//@CTK NO_BUF_OVERFLOW`

Line 154-165 in File Mineral.sol


```
154     function totalBalanceOf(address _of)
155     public
156     view
157     returns (uint256 amount)
158     {
159         amount = balanceOf(_of);
160         for (uint256 i = 0; i < lockReason[_of].length; i++) {
161             amount = amount.add(tokensLocked(_of, lockReason[_of][i]));
162         }
163     }
```

 The code meets the specification.

Formal Verification Request 68

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.54 ms

Line 149 in File Mineral.sol

149 `//@CTK NO_OVERFLOW`

Line 154-165 in File Mineral.sol


```
154 function totalBalanceOf(address _of)
155     public
156     view
157     returns (uint256 amount)
158 {
159     amount = balanceOf(_of);
160     for (uint256 i = 0; i < lockReason[_of].length; i++) {
161         amount = amount.add(tokensLocked(_of, lockReason[_of][i]));
162     }
163 }
```

✓ The code meets the specification.

Formal Verification Request 69

Method will not encounter an assertion failure.

 23, Dec 2019

 0.5 ms

Line 150 in File Mineral.sol

150 `//@CTK NO_ASF`

Line 154-165 in File Mineral.sol


```
154 function totalBalanceOf(address _of)
155     public
156     view
157     returns (uint256 amount)
158 {
159     amount = balanceOf(_of);
160     for (uint256 i = 0; i < lockReason[_of].length; i++) {
161         amount = amount.add(tokensLocked(_of, lockReason[_of][i]));
162     }
163 }
```

✓ The code meets the specification.

Formal Verification Request 70

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 163.18 ms

Line 172 in File Mineral.sol

172 `//@CTK NO_BUF_OVERFLOW`

Line 180-190 in File Mineral.sol

```

180     function extendLock(bytes32 _reason, uint256 _time)
181     public
182     returns (bool)
183     {
184         require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
185
186         locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
187             _time);
188
189         emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
190             _msgSender()][_reason].validity);
191         return true;
192     }

```

✓ The code meets the specification.

Formal Verification Request 71

If method completes, integer overflow would not happen.



23, Dec 2019



3.48 ms

Line 173 in File Mineral.sol

```

173     // @CTK NO_OVERFLOW

```

Line 180-190 in File Mineral.sol

```

180     function extendLock(bytes32 _reason, uint256 _time)
181     public
182     returns (bool)
183     {
184         require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
185
186         locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
187             _time);
188
189         emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
190             _msgSender()][_reason].validity);
191         return true;
192     }

```

✓ The code meets the specification.

Formal Verification Request 72

Method will not encounter an assertion failure.



23, Dec 2019



1.42 ms

Line 174 in File Mineral.sol

```

174     // @CTK NO_ASF

```

Line 180-190 in File Mineral.sol

```

180     function extendLock(bytes32 _reason, uint256 _time)
181     public
182     returns (bool)
183     {
184         require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
185
186         locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
187             _time);
188
189         emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
190             _msgSender()][_reason].validity);
191         return true;
192     }

```

✓ The code meets the specification.

Formal Verification Request 73

extendLock



23, Dec 2019



19.36 ms

Line 175-179 in File Mineral.sol

```

175     /*@CTK extendLock
176     @tag assume_completion
177     @pre (!locked[msg.sender][_reason].claimed && locked[msg.sender][_reason].amount > 0)
178     @post __post.locked[msg.sender][_reason].validity == locked[msg.sender][_reason].
179         validity + _time
180     */

```

Line 180-190 in File Mineral.sol

```

180     function extendLock(bytes32 _reason, uint256 _time)
181     public
182     returns (bool)
183     {
184         require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
185
186         locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
187             _time);
188
189         emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
190             _msgSender()][_reason].validity);
191         return true;
192     }

```

✓ The code meets the specification.

Formal Verification Request 74

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



183.22 ms

Line 197 in File Mineral.sol

```
197 // @CTK_NO_BUF_OVERFLOW
```

Line 205-218 in File Mineral.sol

```
205 function increaseLockAmount(bytes32 _reason, uint256 _amount)
206 public
207 returns (bool)
208 {
209     require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
210     transfer(address(this), _amount);
211
212     locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
213         _amount);
214
215     emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
216         _msgSender()][_reason].validity);
217     return true;
218 }
```

✓ The code meets the specification.

Formal Verification Request 75

If method completes, integer overflow would not happen.



23, Dec 2019



4.03 ms

Line 198 in File Mineral.sol

```
198 // @CTK_NO_OVERFLOW
```

Line 205-218 in File Mineral.sol


```
205 function increaseLockAmount(bytes32 _reason, uint256 _amount)
206 public
207 returns (bool)
208 {
209     require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
210     transfer(address(this), _amount);
211
212     locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
213         _amount);
214
215     emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
216         _msgSender()][_reason].validity);
217     return true;
218 }
```

✓ The code meets the specification.

Formal Verification Request 76

Method will not encounter an assertion failure.

 23, Dec 2019

 1.81 ms

Line 199 in File Mineral.sol

199 `//@CTK NO_ASF`

Line 205-218 in File Mineral.sol

```

205     function increaseLockAmount(bytes32 _reason, uint256 _amount)
206     public
207     returns (bool)
208     {
209         require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
210         transfer(address(this), _amount);
211
212         locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
213             _amount);
214
215         emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
216             _msgSender()][_reason].validity);
217         return true;
218     }


```

 The code meets the specification.

Formal Verification Request 77

increaseLockAmount

 23, Dec 2019

 24.71 ms

Line 200-204 in File Mineral.sol

```

200     /*@CTK increaseLockAmount
201     @tag assume_completion
202     @pre (!locked[msg.sender][_reason].claimed && locked[msg.sender][_reason].amount > 0)
203     @post __post.locked[msg.sender][_reason].amount == locked[msg.sender][_reason].amount
204           + _amount
205     */

```

Line 205-218 in File Mineral.sol

```

205     function increaseLockAmount(bytes32 _reason, uint256 _amount)
206     public
207     returns (bool)
208     {
209         require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
210         transfer(address(this), _amount);
211
212         locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
213             _amount);
214
215         emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
216             _msgSender()][_reason].validity);
217         return true;
218     }

```

```

214     emit Locked(msgSender(), _reason, locked[msgSender()][_reason].amount, locked[
215         msgSender()][_reason].validity);
216     return true;
    }

```

✓ The code meets the specification.

Formal Verification Request 78

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 10.68 ms

Line 225 in File Mineral.sol

```

225     //@CTK NO_BUF_OVERFLOW

```

Line 231-238 in File Mineral.sol

```

231     function tokensUnlockable(address _of, bytes32 _reason)
232     public
233     view
234     returns (uint256 amount)
235     {
236         if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-
237             disable-line
238             amount = locked[_of][_reason].amount;
    }

```

✓ The code meets the specification.

Formal Verification Request 79

If method completes, integer overflow would not happen.

📅 23, Dec 2019

🕒 0.92 ms

Line 226 in File Mineral.sol

```

226     //@CTK NO_OVERFLOW

```

Line 231-238 in File Mineral.sol

```

231     function tokensUnlockable(address _of, bytes32 _reason)
232     public
233     view
234     returns (uint256 amount)
235     {
236         if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-
237             disable-line
238             amount = locked[_of][_reason].amount;
    }

```

✓ The code meets the specification.

Formal Verification Request 80

Method will not encounter an assertion failure.

23, Dec 2019

1.2 ms

Line 227 in File Mineral.sol

227 `//@CTK NO_ASF`

Line 231-238 in File Mineral.sol

```

231     function tokensUnlockable(address _of, bytes32 _reason)
232     public
233     view
234     returns (uint256 amount)
235     {
236         if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-
            disable-line
237             amount = locked[_of][_reason].amount;
238     }

```

The code meets the specification.

Formal Verification Request 81

Buffer overflow / array index out of bound would never happen.

23, Dec 2019

10.4 ms

Line 244 in File Mineral.sol

244 `//@CTK NO_BUF_OVERFLOW`

Line 247-275 in File Mineral.sol

```

247     function unlockAll(address _of)
248     public
249     returns (uint256 unlockableTokens)
250     {
251         uint256 lockedTokens;
252
253         /*#CTK "loop_unlockAll"
254         @inv i <= lockReason[_of].length
255         @inv forall j: uint. (j >= 0 /\ j < i /\ (locked[_of][lockReason[_of][i]].validity
            <= now && !locked[_of][lockReason[_of][i]].claimed && locked[_of][lockReason[
                _of][i]].amount > 0)) -> locked[_of][lockReason[_of][i]].claimed
256         @post i == lockReason[_of].length
257         @post !__should_return
258         */
259         /*@CTK loop
260         @inv true
261         */
262         for (uint256 i = 0; i < lockReason[_of].length; i++) {
263             lockedTokens = tokensUnlockable(_of, lockReason[_of][i]);
264             if (lockedTokens > 0) {
265                 unlockableTokens = unlockableTokens.add(lockedTokens);

```

```

266         locked[_of][lockReason[_of][i]].claimed = true;
267         emit Unlocked(_of, lockReason[_of][i], lockedTokens);
268     }
269 }
270
271 if (unlockableTokens > 0)
272     this.transfer(_of, unlockableTokens);
273 }

```

✓ The code meets the specification.

Formal Verification Request 82

If method completes, integer overflow would not happen.



23, Dec 2019



0.58 ms

Line 245 in File Mineral.sol

```

245 //CTK NO_OVERFLOW

```

Line 247-275 in File Mineral.sol

```

247 function unlockAll(address _of)
248 public
249 returns (uint256 unlockableTokens)
250 {
251     uint256 lockedTokens;
252
253     /*CTK "loop_unlockAll"
254     @inv i <= lockReason[_of].length
255     @inv forall j: uint. (j >= 0 /\ j < i /\ (locked[_of][lockReason[_of][i]].validity
256         <= now && !locked[_of][lockReason[_of][i]].claimed && locked[_of][lockReason[
257         _of][i]].amount > 0)) -> locked[_of][lockReason[_of][i]].claimed
258     @post i == lockReason[_of].length
259     @post !__should_return
260     */
261     /*
262     /*CTK loop
263     @inv true
264     */
265     for (uint256 i = 0; i < lockReason[_of].length; i++) {
266         lockedTokens = tokensUnlocked(_of, lockReason[_of][i]);
267         if (lockedTokens > 0) {
268             unlockableTokens = unlockableTokens.add(lockedTokens);
269             locked[_of][lockReason[_of][i]].claimed = true;
270             emit Unlocked(_of, lockReason[_of][i], lockedTokens);
271         }
272     }
273
274     if (unlockableTokens > 0)
275         this.transfer(_of, unlockableTokens);
276 }

```

✓ The code meets the specification.

Formal Verification Request 83

Method will not encounter an assertion failure.

23, Dec 2019

0.51 ms

Line 246 in File Mineral.sol

246 `/*@CTK NO_ASF`

Line 247-275 in File Mineral.sol

```

247     function unlockAll(address _of)
248     public
249     returns (uint256 unlockableTokens)
250     {
251         uint256 lockedTokens;
252
253         /*#CTK "loop_unlockAll"
254         @inv i <= lockReason[_of].length
255         @inv forall j: uint. (j >= 0 /\ j < i /\ (locked[_of][lockReason[_of][i]].validity
                <= now && !locked[_of][lockReason[_of][i]].claimed && locked[_of][lockReason[
                _of][i]].amount > 0)) -> locked[_of][lockReason[_of][i]].claimed
256         @post i == lockReason[_of].length
257         @post !__should_return
258         */
259         /*@CTK loop
260         @inv true
261         */
262         for (uint256 i = 0; i < lockReason[_of].length; i++) {
263             lockedTokens = tokensUnlockable(_of, lockReason[_of][i]);
264             if (lockedTokens > 0) {
265                 unlockableTokens = unlockableTokens.add(lockedTokens);
266                 locked[_of][lockReason[_of][i]].claimed = true;
267                 emit Unlocked(_of, lockReason[_of][i], lockedTokens);
268             }
269         }
270
271         if (unlockableTokens > 0)
272             this.transfer(_of, unlockableTokens);
273     }

```

The code meets the specification.

Formal Verification Request 84

Buffer overflow / array index out of bound would never happen.

23, Dec 2019

52.85 ms

Line 282 in File Mineral.sol

282 `/*@CTK NO_BUF_OVERFLOW`

Line 288-300 in File Mineral.sol

```
288     function unlock(address _of, bytes32 _reason)
289     public
290     returns (uint256 unlocked)
291     {
292         unlocked = tokensUnlockable(_of, _reason);
293         if (unlocked > 0) {
294             locked[_of][_reason].claimed = true;
295             emit Unlocked(_of, _reason, unlocked);
296             this.transfer(_of, unlocked);
297         }
298     }
```

✓ The code meets the specification.

Formal Verification Request 85

If method completes, integer overflow would not happen.



23, Dec 2019



0.64 ms

Line 283 in File Mineral.sol

```
283     //@CTK NO_OVERFLOW
```

Line 288-300 in File Mineral.sol

```
288     function unlock(address _of, bytes32 _reason)
289     public
290     returns (uint256 unlocked)
291     {
292         unlocked = tokensUnlockable(_of, _reason);
293         if (unlocked > 0) {
294             locked[_of][_reason].claimed = true;
295             emit Unlocked(_of, _reason, unlocked);
296             this.transfer(_of, unlocked);
297         }
298     }
```

✓ The code meets the specification.

Formal Verification Request 86

Method will not encounter an assertion failure.



23, Dec 2019



0.61 ms

Line 284 in File Mineral.sol

```
284     //@CTK NO_ASF
```

Line 288-300 in File Mineral.sol

```
288     function unlock(address _of, bytes32 _reason)
289     public
290     returns (uint256 unlocked)
```

```

291 {
292     unlocked = tokensUnlockable(_of, _reason);
293     if (unlocked > 0) {
294         locked[_of][_reason].claimed = true;
295         emit Unlocked(_of, _reason, unlocked);
296         this.transfer(_of, unlocked);
297     }
298 }


```

✓ The code meets the specification.

Formal Verification Request 87

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 3.92 ms

Line 306 in File Mineral.sol

```

306 // @CTK NO_BUF_OVERFLOW

```

Line 309-319 in File Mineral.sol

```

309 function getUnlockableTokens(address _of)
310     public
311     view
312     returns (uint256 unlockableTokens)
313 {
314     for (uint256 i = 0; i < lockReason[_of].length; i++) {
315         unlockableTokens = unlockableTokens.add(tokensUnlockable(_of, lockReason[_of][i]))
316     }
317 }


```

✓ The code meets the specification.

Formal Verification Request 88

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.38 ms

Line 307 in File Mineral.sol

```

307 // @CTK NO_OVERFLOW

```

Line 309-319 in File Mineral.sol

```

309 function getUnlockableTokens(address _of)
310     public
311     view
312     returns (uint256 unlockableTokens)
313 {
314     for (uint256 i = 0; i < lockReason[_of].length; i++) {

```

```

315         unlockableTokens = unlockableTokens.add(tokensUnlockable(_of, lockReason[_of][i])
316             );
317     }

```

✓ The code meets the specification.

Formal Verification Request 89

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.37 ms

Line 308 in File Mineral.sol

```

308     //@CTK NO_ASF

```

Line 309-319 in File Mineral.sol

```

309     function getUnlockableTokens(address _of)
310     public
311     view
312     returns (uint256 unlockableTokens)
313     {
314         for (uint256 i = 0; i < lockReason[_of].length; i++) {
315             unlockableTokens = unlockableTokens.add(tokensUnlockable(_of, lockReason[_of][i])
316                 );
317         }
318     }

```

✓ The code meets the specification.

Formal Verification Request 90

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 27.29 ms

Line 320 in File Mineral.sol

```

320     //@CTK NO_BUF_OVERFLOW

```

Line 322-339 in File Mineral.sol

```

322     function getLockReasons(address _of, uint256 _start, uint256 _end)
323     external
324     view
325     returns (bytes32[] memory reasons)
326     {
327         uint256 length = _end - _start;
328         reasons = new bytes32[](length);
329         /*@CTK loop_getLockReasons
330             @inv i <= length
331             @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
332                 j]

```

```

332     @post i == length
333     @post !__should_return
334     */
335     for (uint256 i = 0; i < length; i++) {
336         reasons[i] = lockReason[_of][_start + i];
337     }
338     return reasons;
339 }

```

✓ The code meets the specification.

Formal Verification Request 91

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.62 ms

Line 321 in File Mineral.sol

```

321 // @CTK NO_ASF

```

Line 322-339 in File Mineral.sol

```

322 function getLockReasons(address _of, uint256 _start, uint256 _end)
323     external
324     view
325     returns (bytes32[] memory reasons)
326 {
327     uint256 length = _end - _start;
328     reasons = new bytes32[](length);
329     /* @CTK loop_getLockReasons
330        @inv i <= length
331        @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
332            j]
333        @post i == length
334        @post !__should_return
335        */
336     for (uint256 i = 0; i < length; i++) {
337         reasons[i] = lockReason[_of][_start + i];
338     }
339     return reasons;
340 }

```

✓ The code meets the specification.

Formal Verification Request 92

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 6.49 ms

Line 340 in File Mineral.sol

```

340 // @CTK NO_BUF_OVERFLOW

```

Line 343-349 in File Mineral.sol


```
343     function getLockReasonLength(address _of)
344         external
345         view
346         returns (uint256 length)
347     {
348         return lockReason[_of].length;
349     }
```

✓ The code meets the specification.

Formal Verification Request 93

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.4 ms

Line 341 in File Mineral.sol

```
341     //@CTK NO_OVERFLOW
```

Line 343-349 in File Mineral.sol


```
343     function getLockReasonLength(address _of)
344         external
345         view
346         returns (uint256 length)
347     {
348         return lockReason[_of].length;
349     }
```

✓ The code meets the specification.

Formal Verification Request 94

Method will not encounter an assertion failure.

 23, Dec 2019

 0.38 ms

Line 342 in File Mineral.sol

```
342     //@CTK NO_ASF
```

Line 343-349 in File Mineral.sol


```
343     function getLockReasonLength(address _of)
344         external
345         view
346         returns (uint256 length)
347     {
348         return lockReason[_of].length;
349     }
```

✓ The code meets the specification.

Formal Verification Request 95

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 3.75 ms

Line 350 in File Mineral.sol

```
350  // @CTK_NO_BUF_OVERFLOW
```

Line 353-360 in File Mineral.sol


```
353  function safeTransfer(address _to, uint256 _amount, bytes calldata _data)
354      external
355  {
356      require(transfer(_to, _amount), "ERC20: failed transfer");
357      require(_checkOnERC20Received(_to, _amount, _data), "ERC20: transfer to non
          ERC20Receiver implementer");
358  }
```

 The code meets the specification.

Formal Verification Request 96

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.38 ms

Line 351 in File Mineral.sol

```
351  // @CTK_NO_OVERFLOW
```

Line 353-360 in File Mineral.sol


```
353  function safeTransfer(address _to, uint256 _amount, bytes calldata _data)
354      external
355  {
356      require(transfer(_to, _amount), "ERC20: failed transfer");
357      require(_checkOnERC20Received(_to, _amount, _data), "ERC20: transfer to non
          ERC20Receiver implementer");
358  }
```

 The code meets the specification.

Formal Verification Request 97

Method will not encounter an assertion failure.

 23, Dec 2019

 0.38 ms

Line 352 in File Mineral.sol

```
352  // @CTK_NO_ASF
```

Line 353-360 in File Mineral.sol

```

353     function safeTransfer(address _to, uint256 _amount, bytes calldata _data)
354     external
355     {
356         require(transfer(_to, _amount), "ERC20: failed transfer");
357         require(_checkOnERC20Received(_to, _amount, _data), "ERC20: transfer to non
            ERC20Receiver implementer");
358     }

```

✓ The code meets the specification.

Formal Verification Request 98

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 4.49 ms

Line 362 in File Mineral.sol

```

362     //@CTK NO_BUF_OVERFLOW

```

Line 363-374 in File Mineral.sol

```

363     function _checkOnERC20Received(address _to, uint256 _amount, bytes memory _data)
364     internal
365     returns (bool)
366     {
367         if (!_to.isContract()) {
368             return true;
369         }
370
371         return IERC20Receiver(_to).onERC20Received(_msgSender(), _amount, _data);
372     }

```

✓ The code meets the specification.

Formal Verification Request 99

loop__Generated

📅 23, Dec 2019

🕒 86.62 ms

(Loop) Line 259-261 in File Mineral.sol

```

259     /*@CTK loop
260         @inv true
261     */

```

(Loop) Line 259-269 in File Mineral.sol

```

259     /*@CTK loop
260         @inv true
261     */
262     for (uint256 i = 0; i < lockReason[_of].length; i++) {
263         lockedTokens = tokensUnlocked(_of, lockReason[_of][i]);
264         if (lockedTokens > 0) {

```



```

265         unlockableTokens = unlockableTokens.add(lockedTokens);
266         locked[_of][lockReason[_of][i]].claimed = true;
267         emit Unlocked(_of, lockReason[_of][i], lockedTokens);
268     }
269 }

```

✓ The code meets the specification.

Formal Verification Request 100

loop_getLockReasons___Generated

23, Dec 2019

586.58 ms

(Loop) Line 329-334 in File Mineral.sol

```

329     /*@CTK loop_getLockReasons
330     @inv i <= length
331     @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
        j]
332     @post i == length
333     @post !__should_return
334     */

```

(Loop) Line 329-337 in File Mineral.sol

```

329     /*@CTK loop_getLockReasons
330     @inv i <= length
331     @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
        j]
332     @post i == length
333     @post !__should_return
334     */
335     for (uint256 i = 0; i < length; i++) {
336         reasons[i] = lockReason[_of][_start + i];
337     }

```

✓ The code meets the specification.

Formal Verification Request 101

Buffer overflow / array index out of bound would never happen.

23, Dec 2019

13.67 ms

Line 14 in File MineralNFT.sol

```

14     /*@CTK NO_BUF_OVERFLOW

```

Line 20-22 in File MineralNFT.sol

```

20     function _generateTokenId() internal returns (uint256) {
21         return _finalTokenId++;
22     }


```

✓ The code meets the specification.

Formal Verification Request 102

Method will not encounter an assertion failure.

 23, Dec 2019

 0.49 ms

Line 15 in File MineralNFT.sol

```
15 // @CTK NO_ASF
```

Line 20-22 in File MineralNFT.sol


```
20 function _generateTokenId() internal returns (uint256) {  
21     return _finalTokenId++;  
22 }
```

 The code meets the specification.

Formal Verification Request 103

`__generateTokenId`

 23, Dec 2019

 1.13 ms

Line 16-19 in File MineralNFT.sol

```
16 /* @CTK __generateTokenId  
17     @post __post._finalTokenId == _finalTokenId + 1  
18     @post !__reverted -> __return == _finalTokenId  
19 */
```

Line 20-22 in File MineralNFT.sol


```
20 function _generateTokenId() internal returns (uint256) {  
21     return _finalTokenId++;  
22 }
```

 The code meets the specification.

Formal Verification Request 104

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 388.82 ms

Line 24 in File MineralNFT.sol

```
24 // @CTK NO_BUF_OVERFLOW
```

Line 29-34 in File MineralNFT.sol

```
29 function createItem(address to, string calldata jsonUrl) external onlyOwner returns (  
30     uint256 id = _generateTokenId();  
31     _mint(to, id);
```

```
32     _setTokenURI(id, jsonUrl);
33     return id;
34 }
```

✓ The code meets the specification.

Formal Verification Request 105

Method will not encounter an assertion failure.



23, Dec 2019



19.82 ms

Line 25 in File MineralNFT.sol

```
25 // @CTK NO_ASF
```

Line 29-34 in File MineralNFT.sol

```
29     function createItem(address to, string calldata jsonUrl) external onlyOwner returns (
        uint256) {
30         uint256 id = _generateTokenId();
31         _mint(to, id);
32         _setTokenURI(id, jsonUrl);
33         return id;
34     }
```

✓ The code meets the specification.

Formal Verification Request 106

createItem



23, Dec 2019



24.71 ms

Line 26-28 in File MineralNFT.sol

```
26 /* @CTK createItem
27    @post !__reverted -> __return == _finalTokenId
28    */
```

Line 29-34 in File MineralNFT.sol


```
29     function createItem(address to, string calldata jsonUrl) external onlyOwner returns (
        uint256) {
30         uint256 id = _generateTokenId();
31         _mint(to, id);
32         _setTokenURI(id, jsonUrl);
33         return id;
34     }
```

✓ The code meets the specification.

Formal Verification Request 107

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 4.24 ms

Line 35 in File MineralNFT.sol

```
35  // @CTK_NO_BUF_OVERFLOW
```

Line 38-43 in File MineralNFT.sol


```
38  function burnItem(uint256 tokenId) external {
39      require(_isApprovedOrOwner(_msgSender(), tokenId), "msg.sender is not token owner");
40      _burn(_msgSender(), tokenId);
41  }
```

 The code meets the specification.

Formal Verification Request 108

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.42 ms

Line 36 in File MineralNFT.sol

```
36  // @CTK_NO_OVERFLOW
```

Line 38-43 in File MineralNFT.sol


```
38  function burnItem(uint256 tokenId) external {
39      require(_isApprovedOrOwner(_msgSender(), tokenId), "msg.sender is not token owner");
40      _burn(_msgSender(), tokenId);
41  }
```

 The code meets the specification.

Formal Verification Request 109

Method will not encounter an assertion failure.

 23, Dec 2019

 0.44 ms

Line 37 in File MineralNFT.sol

```
37  // @CTK_NO_ASF
```

Line 38-43 in File MineralNFT.sol

```
38  function burnItem(uint256 tokenId) external {
39      require(_isApprovedOrOwner(_msgSender(), tokenId), "msg.sender is not token owner");
40      _burn(_msgSender(), tokenId);
41  }
```

 The code meets the specification.

Formal Verification Request 110

Method will not encounter an assertion failure.



23, Dec 2019



5.66 ms

Line 46 in File ERC20.sol

```
46  //@CTK NO_ASF
```

Line 49-51 in File ERC20.sol

```
49  function totalSupply() public view returns (uint256) {  
50      return _totalSupply;  
51  }
```

✓ The code meets the specification.

Formal Verification Request 111

If method completes, integer overflow would not happen.



23, Dec 2019



0.4 ms

Line 47 in File ERC20.sol

```
47  //@CTK NO_OVERFLOW
```

Line 49-51 in File ERC20.sol

```
49  function totalSupply() public view returns (uint256) {  
50      return _totalSupply;  
51  }
```

✓ The code meets the specification.

Formal Verification Request 112

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.36 ms

Line 48 in File ERC20.sol

```
48  //@CTK NO_BUF_OVERFLOW
```

Line 49-51 in File ERC20.sol

```
49  function totalSupply() public view returns (uint256) {  
50      return _totalSupply;  
51  }
```

✓ The code meets the specification.

Formal Verification Request 113

Method will not encounter an assertion failure.



23, Dec 2019



6.02 ms

Line 56 in File ERC20.sol

```
56    //@CTK NO_ASF
```

Line 59-61 in File ERC20.sol

```
59    function balanceOf(address account) public view returns (uint256) {  
60        return _balances[account];  
61    }
```

✓ The code meets the specification.

Formal Verification Request 114

If method completes, integer overflow would not happen.



23, Dec 2019



0.38 ms

Line 57 in File ERC20.sol

```
57    //@CTK NO_OVERFLOW
```

Line 59-61 in File ERC20.sol

```
59    function balanceOf(address account) public view returns (uint256) {  
60        return _balances[account];  
61    }
```

✓ The code meets the specification.

Formal Verification Request 115

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.36 ms

Line 58 in File ERC20.sol

```
58    //@CTK NO_BUF_OVERFLOW
```

Line 59-61 in File ERC20.sol

```
59    function balanceOf(address account) public view returns (uint256) {  
60        return _balances[account];  
61    }
```

✓ The code meets the specification.

Formal Verification Request 116

Method will not encounter an assertion failure.



23, Dec 2019



6.28 ms

Line 79 in File ERC20.sol

79 `//@CTK NO_ASF`

Line 82-84 in File ERC20.sol

```
82 function allowance(address owner, address spender) public view returns (uint256) {  
83     return _allowances[owner][spender];  
84 }
```

✓ The code meets the specification.

Formal Verification Request 117

If method completes, integer overflow would not happen.



23, Dec 2019



0.39 ms

Line 80 in File ERC20.sol

80 `//@CTK NO_OVERFLOW`

Line 82-84 in File ERC20.sol

```
82 function allowance(address owner, address spender) public view returns (uint256) {  
83     return _allowances[owner][spender];  
84 }
```

✓ The code meets the specification.

Formal Verification Request 118

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.4 ms

Line 81 in File ERC20.sol

81 `//@CTK NO_BUF_OVERFLOW`

Line 82-84 in File ERC20.sol

```
82 function allowance(address owner, address spender) public view returns (uint256) {  
83     return _allowances[owner][spender];  
84 }
```

✓ The code meets the specification.

Formal Verification Request 119

Method will not encounter an assertion failure.



23, Dec 2019



88.32 ms

Line 93 in File ERC20.sol

```
93  // @CTK_NO_ASF
```

Line 96-99 in File ERC20.sol

```
96  function approve(address spender, uint256 amount) public returns (bool) {
97      _approve(_msgSender(), spender, amount);
98      return true;
99  }
```

✓ The code meets the specification.

Formal Verification Request 120

If method completes, integer overflow would not happen.



23, Dec 2019



0.76 ms

Line 94 in File ERC20.sol

```
94  // @CTK_NO_OVERFLOW
```

Line 96-99 in File ERC20.sol

```
96  function approve(address spender, uint256 amount) public returns (bool) {
97      _approve(_msgSender(), spender, amount);
98      return true;
99  }
```

✓ The code meets the specification.

Formal Verification Request 121

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.68 ms

Line 95 in File ERC20.sol

```
95  // @CTK_NO_BUF_OVERFLOW
```

Line 96-99 in File ERC20.sol

```
96  function approve(address spender, uint256 amount) public returns (bool) {
97      _approve(_msgSender(), spender, amount);
98      return true;
99  }
```

✓ The code meets the specification.

Formal Verification Request 122

Method will not encounter an assertion failure.



23, Dec 2019



99.21 ms

Line 131 in File ERC20.sol

```
131 // @CTK_NO_ASF
```

Line 134-137 in File ERC20.sol

```
134 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
135     _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
136     return true;
137 }
```

✓ The code meets the specification.

Formal Verification Request 123

If method completes, integer overflow would not happen.



23, Dec 2019



3.23 ms

Line 132 in File ERC20.sol

```
132 // @CTK_NO_OVERFLOW
```

Line 134-137 in File ERC20.sol

```
134 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
135     _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
136     return true;
137 }
```

✓ The code meets the specification.

Formal Verification Request 124

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



1.15 ms

Line 133 in File ERC20.sol

```
133 // @CTK_NO_BUF_OVERFLOW
```

Line 134-137 in File ERC20.sol


```
134 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
135     _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
136     return true;
137 }
```

✓ The code meets the specification.

Formal Verification Request 125

Method will not encounter an assertion failure.

 23, Dec 2019

 72.44 ms

Line 190 in File ERC20.sol

190 `//@CTK NO_ASF`

Line 193-199 in File ERC20.sol


```
193     function _mint(address account, uint256 amount) internal {
194         require(account != address(0), "ERC20: mint to the zero address");
195
196         _totalSupply = _totalSupply.add(amount);
197         _balances[account] = _balances[account].add(amount);
198         emit Transfer(address(0), account, amount);
199     }
```

 The code meets the specification.

Formal Verification Request 126

If method completes, integer overflow would not happen.

 23, Dec 2019

 6.43 ms

Line 191 in File ERC20.sol

191 `//@CTK NO_OVERFLOW`

Line 193-199 in File ERC20.sol


```
193     function _mint(address account, uint256 amount) internal {
194         require(account != address(0), "ERC20: mint to the zero address");
195
196         _totalSupply = _totalSupply.add(amount);
197         _balances[account] = _balances[account].add(amount);
198         emit Transfer(address(0), account, amount);
199     }
```

 The code meets the specification.

Formal Verification Request 127

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 3.26 ms

Line 192 in File ERC20.sol

192 `//@CTK NO_BUF_OVERFLOW`

Line 193-199 in File ERC20.sol

```

193     function _mint(address account, uint256 amount) internal {
194         require(account != address(0), "ERC20: mint to the zero address");
195
196         _totalSupply = _totalSupply.add(amount);
197         _balances[account] = _balances[account].add(amount);
198         emit Transfer(address(0), account, amount);
199     }

```

✓ The code meets the specification.

Formal Verification Request 128

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.57 ms

Line 233 in File ERC20.sol

```

233     //@CTK NO_ASF

```

Line 236-242 in File ERC20.sol

```

236     function _approve(address owner, address spender, uint256 amount) internal {
237         require(owner != address(0), "ERC20: approve from the zero address");
238         require(spender != address(0), "ERC20: approve to the zero address");
239
240         _allowances[owner][spender] = amount;
241         emit Approval(owner, spender, amount);
242     }

```

✓ The code meets the specification.

Formal Verification Request 129

If method completes, integer overflow would not happen.

📅 23, Dec 2019

🕒 0.52 ms

Line 234 in File ERC20.sol

```

234     //@CTK NO_OVERFLOW

```

Line 236-242 in File ERC20.sol

```

236     function _approve(address owner, address spender, uint256 amount) internal {
237         require(owner != address(0), "ERC20: approve from the zero address");
238         require(spender != address(0), "ERC20: approve to the zero address");
239
240         _allowances[owner][spender] = amount;
241         emit Approval(owner, spender, amount);
242     }


```

✓ The code meets the specification.

Formal Verification Request 130

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.52 ms

Line 235 in File ERC20.sol

235 `//@CTK_NO_BUF_OVERFLOW`

Line 236-242 in File ERC20.sol


```
236     function _approve(address owner, address spender, uint256 amount) internal {
237         require(owner != address(0), "ERC20: approve from the zero address");
238         require(spender != address(0), "ERC20: approve to the zero address");
239
240         _allowances[owner][spender] = amount;
241         emit Approval(owner, spender, amount);
242     }
```

 The code meets the specification.

Formal Verification Request 131

Method will not encounter an assertion failure.

 23, Dec 2019

 65.53 ms

Line 37 in File ERC721Enumerable.sol

37 `//@CTK_NO_ASF`

Line 40-43 in File ERC721Enumerable.sol


```
40     constructor () public {
41         // register the supported interface to conform to ERC721Enumerable via ERC165
42         _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
43     }
```

 The code meets the specification.

Formal Verification Request 132

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.74 ms

Line 38 in File ERC721Enumerable.sol

38 `//@CTK_NO_OVERFLOW`

Line 40-43 in File ERC721Enumerable.sol

```

40     constructor () public {
41         // register the supported interface to conform to ERC721Enumerable via ERC165
42         _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
43     }

```

✓ The code meets the specification.

Formal Verification Request 133

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 0.69 ms

Line 39 in File ERC721Enumerable.sol

```

39     // @CTK_NO_BUF_OVERFLOW

```

Line 40-43 in File ERC721Enumerable.sol

```

40     constructor () public {
41         // register the supported interface to conform to ERC721Enumerable via ERC165
42         _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
43     }

```

✓ The code meets the specification.

Formal Verification Request 134

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 7.49 ms

Line 61 in File ERC721Enumerable.sol

```

61     // @CTK_NO_ASF

```

Line 64-66 in File ERC721Enumerable.sol

```

64     function totalSupply() public view returns (uint256) {
65         return _allTokens.length;
66     }

```

✓ The code meets the specification.

Formal Verification Request 135

If method completes, integer overflow would not happen.

📅 23, Dec 2019

🕒 0.39 ms

Line 62 in File ERC721Enumerable.sol

```

62     // @CTK_NO_OVERFLOW

```

Line 64-66 in File ERC721Enumerable.sol


```
64     function totalSupply() public view returns (uint256) {  
65         return _allTokens.length;  
66     }
```

✓ The code meets the specification.

Formal Verification Request 136

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.4 ms

Line 63 in File ERC721Enumerable.sol

```
63     //@CTK NO_BUF_OVERFLOW
```

Line 64-66 in File ERC721Enumerable.sol


```
64     function totalSupply() public view returns (uint256) {  
65         return _allTokens.length;  
66     }
```

✓ The code meets the specification.

Formal Verification Request 137

Method will not encounter an assertion failure.

 23, Dec 2019

 40.38 ms

Line 74 in File ERC721Enumerable.sol

```
74     //@CTK NO_ASF
```

Line 77-80 in File ERC721Enumerable.sol


```
77     function tokenByIndex(uint256 index) public view returns (uint256) {  
78         require(index < totalSupply(), "ERC721Enumerable: global index out of bounds");  
79         return _allTokens[index];  
80     }
```

✓ The code meets the specification.

Formal Verification Request 138

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.64 ms

Line 75 in File ERC721Enumerable.sol

75 `//@CTK NO_OVERFLOW`

Line 77-80 in File ERC721Enumerable.sol


```
77     function tokenByIndex(uint256 index) public view returns (uint256) {  
78         require(index < totalSupply(), "ERC721Enumerable: global index out of bounds");  
79         return _allTokens[index];  
80     }
```

✓ The code meets the specification.

Formal Verification Request 139

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 1.5 ms

Line 76 in File ERC721Enumerable.sol

76 `//@CTK NO_BUF_OVERFLOW`

Line 77-80 in File ERC721Enumerable.sol


```
77     function tokenByIndex(uint256 index) public view returns (uint256) {  
78         require(index < totalSupply(), "ERC721Enumerable: global index out of bounds");  
79         return _allTokens[index];  
80     }
```

✓ The code meets the specification.

Formal Verification Request 140

Method will not encounter an assertion failure.

 23, Dec 2019

 273.39 ms

Line 103 in File ERC721Enumerable.sol

103 `//@CTK NO_ASF`

Line 105-111 in File ERC721Enumerable.sol


```
105     function _mint(address to, uint256 tokenId) internal {  
106         super._mint(to, tokenId);  
107  
108         _addTokenToOwnerEnumeration(to, tokenId);  
109  
110         _addTokenToAllTokensEnumeration(tokenId);  
111     }
```

✓ The code meets the specification.

Formal Verification Request 141

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 8.61 ms

Line 104 in File ERC721Enumerable.sol

104 `//@CTK NO_BUF_OVERFLOW`

Line 105-111 in File ERC721Enumerable.sol


```
105     function _mint(address to, uint256 tokenId) internal {
106         super._mint(to, tokenId);
107
108         _addTokenToOwnerEnumeration(to, tokenId);
109
110         _addTokenToAllTokensEnumeration(tokenId);
111     }
```

 The code meets the specification.

Formal Verification Request 142

Method will not encounter an assertion failure.

 23, Dec 2019

 6.48 ms

Line 136 in File ERC721Enumerable.sol

136 `//@CTK NO_ASF`

Line 139-141 in File ERC721Enumerable.sol


```
139     function _tokensOfOwner(address owner) internal view returns (uint256[] storage) {
140         return _ownedTokens[owner];
141     }
```

 The code meets the specification.

Formal Verification Request 143

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.38 ms

Line 137 in File ERC721Enumerable.sol

137 `//@CTK NO_OVERFLOW`

Line 139-141 in File ERC721Enumerable.sol


```
139     function _tokensOfOwner(address owner) internal view returns (uint256[] storage) {
140         return _ownedTokens[owner];
141     }
```

 The code meets the specification.

Formal Verification Request 144

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.36 ms

Line 138 in File ERC721Enumerable.sol

138 `//@CTK NO_BUF_OVERFLOW`

Line 139-141 in File ERC721Enumerable.sol


```
139     function _tokensOfOwner(address owner) internal view returns (uint256[] storage) {
140         return _ownedTokens[owner];
141     }
```

 The code meets the specification.

Formal Verification Request 145

Method will not encounter an assertion failure.

 23, Dec 2019

 0.44 ms

Line 148 in File ERC721Enumerable.sol

148 `//@CTK NO_ASF`

Line 150-153 in File ERC721Enumerable.sol


```
150     function _addTokenToOwnerEnumeration(address to, uint256 tokenId) private {
151         _ownedTokensIndex[tokenId] = _ownedTokens[to].length;
152         _ownedTokens[to].push(tokenId);
153     }
```

 The code meets the specification.

Formal Verification Request 146

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.47 ms

Line 149 in File ERC721Enumerable.sol

149 `//@CTK NO_BUF_OVERFLOW`

Line 150-153 in File ERC721Enumerable.sol


```
150     function _addTokenToOwnerEnumeration(address to, uint256 tokenId) private {
151         _ownedTokensIndex[tokenId] = _ownedTokens[to].length;
152         _ownedTokens[to].push(tokenId);
153     }
```

 The code meets the specification.

Formal Verification Request 147

Method will not encounter an assertion failure.

 23, Dec 2019

 0.43 ms

Line 159 in File ERC721Enumerable.sol

159 `//@CTK NO_ASF`

Line 161-164 in File ERC721Enumerable.sol


```
161 function _addTokenToAllTokensEnumeration(uint256 tokenId) private {  
162     _allTokensIndex[tokenId] = _allTokens.length;  
163     _allTokens.push(tokenId);  
164 }
```

 The code meets the specification.

Formal Verification Request 148

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.45 ms

Line 160 in File ERC721Enumerable.sol

160 `//@CTK NO_BUF_OVERFLOW`

Line 161-164 in File ERC721Enumerable.sol


```
161 function _addTokenToAllTokensEnumeration(uint256 tokenId) private {  
162     _allTokensIndex[tokenId] = _allTokens.length;  
163     _allTokens.push(tokenId);  
164 }
```

 The code meets the specification.

Formal Verification Request 149

Method will not encounter an assertion failure.

 23, Dec 2019

 75.01 ms

Line 33 in File ERC721Metadata.sol

33 `//@CTK NO_ASF`

Line 36-42 in File ERC721Metadata.sol

```
36 constructor (string memory name, string memory symbol) public {  
37     _name = name;  
38     _symbol = symbol;  
39   
40     // register the supported interfaces to conform to ERC721 via ERC165  
41     _registerInterface(_INTERFACE_ID_ERC721_METADATA);  
42 }
```

✓ The code meets the specification.

Formal Verification Request 150

If method completes, integer overflow would not happen.



23, Dec 2019



0.99 ms

Line 34 in File ERC721Metadata.sol

```
34    //@CTK_NO_OVERFLOW
```

Line 36-42 in File ERC721Metadata.sol

```
36    constructor (string memory name, string memory symbol) public {
37        _name = name;
38        _symbol = symbol;
39
40        // register the supported interfaces to conform to ERC721 via ERC165
41        _registerInterface(_INTERFACE_ID_ERC721_METADATA);
42    }
```

✓ The code meets the specification.

Formal Verification Request 151

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.82 ms

Line 35 in File ERC721Metadata.sol

```
35    //@CTK_NO_BUF_OVERFLOW
```

Line 36-42 in File ERC721Metadata.sol

```
36    constructor (string memory name, string memory symbol) public {
37        _name = name;
38        _symbol = symbol;
39
40        // register the supported interfaces to conform to ERC721 via ERC165
41        _registerInterface(_INTERFACE_ID_ERC721_METADATA);
42    }
```

✓ The code meets the specification.

Formal Verification Request 152

Method will not encounter an assertion failure.



23, Dec 2019



6.01 ms

Line 48 in File ERC721Metadata.sol

48 `//@CTK NO_ASF`

Line 51-53 in File ERC721Metadata.sol


```
51 function name() external view returns (string memory) {  
52     return _name;  
53 }
```

✓ The code meets the specification.

Formal Verification Request 153

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.44 ms

Line 49 in File ERC721Metadata.sol

49 `//@CTK NO_OVERFLOW`

Line 51-53 in File ERC721Metadata.sol


```
51 function name() external view returns (string memory) {  
52     return _name;  
53 }
```

✓ The code meets the specification.

Formal Verification Request 154

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.39 ms

Line 50 in File ERC721Metadata.sol

50 `//@CTK NO_BUF_OVERFLOW`

Line 51-53 in File ERC721Metadata.sol


```
51 function name() external view returns (string memory) {  
52     return _name;  
53 }
```

✓ The code meets the specification.

Formal Verification Request 155

Method will not encounter an assertion failure.

 23, Dec 2019

 6.47 ms

Line 59 in File ERC721Metadata.sol

59 `//@CTK NO_ASF`

Line 62-64 in File ERC721Metadata.sol


```
62     function symbol() external view returns (string memory) {  
63         return _symbol;  
64     }
```

✓ The code meets the specification.

Formal Verification Request 156

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.43 ms

Line 60 in File ERC721Metadata.sol

60 `//@CTK NO_OVERFLOW`

Line 62-64 in File ERC721Metadata.sol


```
62     function symbol() external view returns (string memory) {  
63         return _symbol;  
64     }
```

✓ The code meets the specification.

Formal Verification Request 157

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.45 ms

Line 61 in File ERC721Metadata.sol

61 `//@CTK NO_BUF_OVERFLOW`

Line 62-64 in File ERC721Metadata.sol


```
62     function symbol() external view returns (string memory) {  
63         return _symbol;  
64     }
```

✓ The code meets the specification.

Formal Verification Request 158

Method will not encounter an assertion failure.

 23, Dec 2019

 49.59 ms

Line 74 in File ERC721Metadata.sol

74 //CTK NO_ASF

Line 77-91 in File ERC721Metadata.sol


```
77 function tokenURI(uint256 tokenId) external view returns (string memory) {
78     require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");
79
80     string memory _tokenURI = _tokenURIs[tokenId];
81
82     // Even if there is a base URI, it is only appended to non-empty token-specific URIs
83     if (bytes(_tokenURI).length == 0) {
84         return "";
85     } else {
86         // abi.encodePacked is being used to concatenate strings
87         return string(abi.encodePacked(_baseURI, _tokenURI));
88     }
89 }
```

✓ The code meets the specification.

Formal Verification Request 159

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.85 ms

Line 75 in File ERC721Metadata.sol

75 //CTK NO_OVERFLOW

Line 77-91 in File ERC721Metadata.sol


```
77 function tokenURI(uint256 tokenId) external view returns (string memory) {
78     require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");
79
80     string memory _tokenURI = _tokenURIs[tokenId];
81
82     // Even if there is a base URI, it is only appended to non-empty token-specific URIs
83     if (bytes(_tokenURI).length == 0) {
84         return "";
85     } else {
86         // abi.encodePacked is being used to concatenate strings
87         return string(abi.encodePacked(_baseURI, _tokenURI));
88     }
89 }
```

✓ The code meets the specification.

Formal Verification Request 160

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.62 ms

Line 76 in File ERC721Metadata.sol

76 `//@CTK NO_BUF_OVERFLOW`

Line 77-91 in File ERC721Metadata.sol


```
77     function tokenURI(uint256 tokenId) external view returns (string memory) {
78         require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");
79
80         string memory _tokenURI = _tokenURIs[tokenId];
81
82         // Even if there is a base URI, it is only appended to non-empty token-specific URIs
83         if (bytes(_tokenURI).length == 0) {
84             return "";
85         } else {
86             // abi.encodePacked is being used to concatenate strings
87             return string(abi.encodePacked(_baseURI, _tokenURI));
88         }
89     }
```

✓ The code meets the specification.

Formal Verification Request 161

Method will not encounter an assertion failure.

 23, Dec 2019

 5.79 ms

Line 98 in File ERC721Metadata.sol

98 `//@CTK NO_ASF`

Line 101-103 in File ERC721Metadata.sol


```
101     function baseURI() external view returns (string memory) {
102         return _baseURI;
103     }
```

✓ The code meets the specification.

Formal Verification Request 162

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.43 ms

Line 99 in File ERC721Metadata.sol

99 `//@CTK NO_OVERFLOW`

Line 101-103 in File ERC721Metadata.sol


```
101     function baseURI() external view returns (string memory) {
102         return _baseURI;
103     }
```

✓ The code meets the specification.

Formal Verification Request 163

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.37 ms

Line 100 in File ERC721Metadata.sol

100 `//@CTK NO_BUF_OVERFLOW`

Line 101-103 in File ERC721Metadata.sol


```
101     function baseURI() external view returns (string memory) {
102         return _baseURI;
103     }
```

 The code meets the specification.

Formal Verification Request 164

Method will not encounter an assertion failure.

 23, Dec 2019

 38.57 ms

Line 114 in File ERC721Metadata.sol

114 `//@CTK NO_ASF`

Line 117-120 in File ERC721Metadata.sol


```
117     function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal {
118         require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
119         _tokenURIs[tokenId] = _tokenURI;
120     }
```

 The code meets the specification.

Formal Verification Request 165

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.57 ms

Line 115 in File ERC721Metadata.sol

115 `//@CTK NO_OVERFLOW`

Line 117-120 in File ERC721Metadata.sol


```
117     function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal {
118         require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
119         _tokenURIs[tokenId] = _tokenURI;
120     }
```

 The code meets the specification.

Formal Verification Request 166

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.61 ms

Line 116 in File ERC721Metadata.sol

```
116    //@CTK_NO_BUF_OVERFLOW
```

Line 117-120 in File ERC721Metadata.sol


```
117    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal {
118        require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
119        _tokenURIs[tokenId] = _tokenURI;
120    }
```

 The code meets the specification.

Formal Verification Request 167

Method will not encounter an assertion failure.

 23, Dec 2019

 8.25 ms

Line 128 in File ERC721Metadata.sol

```
128    //@CTK_NO_ASF
```

Line 131-133 in File ERC721Metadata.sol


```
131    function _setBaseURI(string memory uri) internal {
132        _baseURI = uri;
133    }
```

 The code meets the specification.

Formal Verification Request 168

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.38 ms

Line 129 in File ERC721Metadata.sol

```
129    //@CTK_NO_OVERFLOW
```

Line 131-133 in File ERC721Metadata.sol


```
131    function _setBaseURI(string memory uri) internal {
132        _baseURI = uri;
133    }
```

 The code meets the specification.

Formal Verification Request 169

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.37 ms

Line 130 in File ERC721Metadata.sol

```
130    //@CTK NO_BUF_OVERFLOW
```

Line 131-133 in File ERC721Metadata.sol


```
131    function _setBaseURI(string memory uri) internal {  
132        _baseURI = uri;  
133    }
```

 The code meets the specification.

Formal Verification Request 170

Method will not encounter an assertion failure.

 23, Dec 2019

 12.9 ms

Line 142 in File ERC721Metadata.sol

```
142    //@CTK NO_ASF
```

Line 145-154 in File ERC721Metadata.sol


```
145    function _burn(address owner, uint256 tokenId) internal {  
146        super._burn(owner, tokenId);  
147  
148        // Clear metadata (if any)  
149        if (bytes(_tokenURIs[tokenId]).length != 0) {  
150            delete _tokenURIs[tokenId];  
151        }  
152    }
```

 The code meets the specification.

Formal Verification Request 171

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.4 ms

Line 143 in File ERC721Metadata.sol

```
143    //@CTK NO_OVERFLOW
```

Line 145-154 in File ERC721Metadata.sol


```
145 function _burn(address owner, uint256 tokenId) internal {
146     super._burn(owner, tokenId);
147
148     // Clear metadata (if any)
149     if (bytes(_tokenURIs[tokenId]).length != 0) {
150         delete _tokenURIs[tokenId];
151     }
152 }
```

✓ The code meets the specification.

Formal Verification Request 172

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.4 ms

Line 144 in File ERC721Metadata.sol

```
144 // @CTK_NO_BUF_OVERFLOW
```

Line 145-154 in File ERC721Metadata.sol


```
145 function _burn(address owner, uint256 tokenId) internal {
146     super._burn(owner, tokenId);
147
148     // Clear metadata (if any)
149     if (bytes(_tokenURIs[tokenId]).length != 0) {
150         delete _tokenURIs[tokenId];
151     }
152 }
```

✓ The code meets the specification.

Formal Verification Request 173

Method will not encounter an assertion failure.

 23, Dec 2019

 61.14 ms

Line 51 in File ERC721.sol

```
51 // @CTK_NO_ASF
```

Line 54-57 in File ERC721.sol

```
54 constructor () public {
55     // register the supported interfaces to conform to ERC721 via ERC165
56     _registerInterface(_INTERFACE_ID_ERC721);
57 }
```

✓ The code meets the specification.

Formal Verification Request 174

If method completes, integer overflow would not happen.



23, Dec 2019



0.62 ms

Line 52 in File ERC721.sol

52 `//@CTK NO_OVERFLOW`

Line 54-57 in File ERC721.sol

```
54     constructor () public {  
55         // register the supported interfaces to conform to ERC721 via ERC165  
56         _registerInterface(_INTERFACE_ID_ERC721);  
57     }
```



The code meets the specification.

Formal Verification Request 175

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.59 ms

Line 53 in File ERC721.sol

53 `//@CTK NO_BUF_OVERFLOW`

Line 54-57 in File ERC721.sol

```
54     constructor () public {  
55         // register the supported interfaces to conform to ERC721 via ERC165  
56         _registerInterface(_INTERFACE_ID_ERC721);  
57     }
```



The code meets the specification.

Formal Verification Request 176

Method will not encounter an assertion failure.



23, Dec 2019



44.66 ms

Line 64 in File ERC721.sol

64 `//@CTK NO_ASF`

Line 67-71 in File ERC721.sol

```
67     function balanceOf(address owner) public view returns (uint256) {  
68         require(owner != address(0), "ERC721: balance query for the zero address");  
69  
70         return _ownedTokensCount[owner].current();  
71     }
```



The code meets the specification.

Formal Verification Request 177

If method completes, integer overflow would not happen.



23, Dec 2019



0.6 ms

Line 65 in File ERC721.sol

```
65  // @CTK_NO_OVERFLOW
```

Line 67-71 in File ERC721.sol

```
67  function balanceOf(address owner) public view returns (uint256) {  
68      require(owner != address(0), "ERC721: balance query for the zero address");  
69  
70      return _ownedTokensCount[owner].current();  
71  }
```

✓ The code meets the specification.

Formal Verification Request 178

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.68 ms

Line 66 in File ERC721.sol

```
66  // @CTK_NO_BUF_OVERFLOW
```

Line 67-71 in File ERC721.sol

```
67  function balanceOf(address owner) public view returns (uint256) {  
68      require(owner != address(0), "ERC721: balance query for the zero address");  
69  
70      return _ownedTokensCount[owner].current();  
71  }
```

✓ The code meets the specification.

Formal Verification Request 179

Method will not encounter an assertion failure.



23, Dec 2019



21.51 ms

Line 78 in File ERC721.sol

```
78  // @CTK_NO_ASF
```

Line 81-86 in File ERC721.sol


```
81     function ownerOf(uint256 tokenId) public view returns (address) {  
82         address owner = _tokenOwner[tokenId];  
83         require(owner != address(0), "ERC721: owner query for nonexistent token");  
84  
85         return owner;  
86     }
```

✓ The code meets the specification.

Formal Verification Request 180

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.48 ms

Line 79 in File ERC721.sol

```
79     //@CTK NO_OVERFLOW
```

Line 81-86 in File ERC721.sol


```
81     function ownerOf(uint256 tokenId) public view returns (address) {  
82         address owner = _tokenOwner[tokenId];  
83         require(owner != address(0), "ERC721: owner query for nonexistent token");  
84  
85         return owner;  
86     }
```

✓ The code meets the specification.

Formal Verification Request 181

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.47 ms

Line 80 in File ERC721.sol

```
80     //@CTK NO_BUF_OVERFLOW
```

Line 81-86 in File ERC721.sol

```
81     function ownerOf(uint256 tokenId) public view returns (address) {  
82         address owner = _tokenOwner[tokenId];  
83         require(owner != address(0), "ERC721: owner query for nonexistent token");  
84  
85         return owner;  
86     }
```

✓ The code meets the specification.

Formal Verification Request 182

Method will not encounter an assertion failure.



23, Dec 2019



126.24 ms

Line 96 in File ERC721.sol

96 `//@CTK NO_ASF`

Line 99-109 in File ERC721.sol

```
99     function approve(address to, uint256 tokenId) public {
100         address owner = ownerOf(tokenId);
101         require(to != owner, "ERC721: approval to current owner");
102
103         require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
104             "ERC721: approve caller is not owner nor approved for all"
105         );
106
107         _tokenApprovals[tokenId] = to;
108         emit Approval(owner, to, tokenId);
109     }
```

✓ The code meets the specification.

Formal Verification Request 183

If method completes, integer overflow would not happen.



23, Dec 2019



3.4 ms

Line 97 in File ERC721.sol

97 `//@CTK NO_OVERFLOW`

Line 99-109 in File ERC721.sol


```
99     function approve(address to, uint256 tokenId) public {
100         address owner = ownerOf(tokenId);
101         require(to != owner, "ERC721: approval to current owner");
102
103         require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
104             "ERC721: approve caller is not owner nor approved for all"
105         );
106
107         _tokenApprovals[tokenId] = to;
108         emit Approval(owner, to, tokenId);
109     }
```

✓ The code meets the specification.

Formal Verification Request 184

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 3.49 ms

Line 98 in File ERC721.sol

98 `//@CTK NO_BUF_OVERFLOW`

Line 99-109 in File ERC721.sol


```
99     function approve(address to, uint256 tokenId) public {
100         address owner = ownerOf(tokenId);
101         require(to != owner, "ERC721: approval to current owner");
102
103         require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
104             "ERC721: approve caller is not owner nor approved for all"
105         );
106
107         _tokenApprovals[tokenId] = to;
108         emit Approval(owner, to, tokenId);
109     }
```

 The code meets the specification.

Formal Verification Request 185

Method will not encounter an assertion failure.

 23, Dec 2019

 41.18 ms

Line 117 in File ERC721.sol

117 `//@CTK NO_ASF`

Line 120-124 in File ERC721.sol


```
120     function getApproved(uint256 tokenId) public view returns (address) {
121         require(_exists(tokenId), "ERC721: approved query for nonexistent token");
122
123         return _tokenApprovals[tokenId];
124     }
```

 The code meets the specification.

Formal Verification Request 186

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.59 ms

Line 118 in File ERC721.sol

118 `//@CTK NO_OVERFLOW`

Line 120-124 in File ERC721.sol


```
120     function getApproved(uint256 tokenId) public view returns (address) {
121         require(_exists(tokenId), "ERC721: approved query for nonexistent token");
122
123         return _tokenApprovals[tokenId];
124     }
```

✓ The code meets the specification.

Formal Verification Request 187

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.53 ms

Line 119 in File ERC721.sol

119 `//@CTK NO_BUF_OVERFLOW`

Line 120-124 in File ERC721.sol


```
120     function getApproved(uint256 tokenId) public view returns (address) {
121         require(_exists(tokenId), "ERC721: approved query for nonexistent token");
122
123         return _tokenApprovals[tokenId];
124     }
```

✓ The code meets the specification.

Formal Verification Request 188

Method will not encounter an assertion failure.

 23, Dec 2019

 60.89 ms

Line 132 in File ERC721.sol

132 `//@CTK NO_ASF`

Line 135-140 in File ERC721.sol


```
135     function setApprovalForAll(address to, bool approved) public {
136         require(to != _msgSender(), "ERC721: approve to caller");
137
138         _operatorApprovals[_msgSender()][to] = approved;
139         emit ApprovalForAll(_msgSender(), to, approved);
140     }
```

✓ The code meets the specification.

Formal Verification Request 189

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.85 ms

Line 133 in File ERC721.sol

```
133  //@CTK_NO_OVERFLOW
```

Line 135-140 in File ERC721.sol


```
135  function setApprovalForAll(address to, bool approved) public {
136      require(to != _msgSender(), "ERC721: approve to caller");
137
138      _operatorApprovals[_msgSender()][to] = approved;
139      emit ApprovalForAll(_msgSender(), to, approved);
140  }
```

 The code meets the specification.

Formal Verification Request 190

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.78 ms

Line 134 in File ERC721.sol

```
134  //@CTK_NO_BUF_OVERFLOW
```

Line 135-140 in File ERC721.sol


```
135  function setApprovalForAll(address to, bool approved) public {
136      require(to != _msgSender(), "ERC721: approve to caller");
137
138      _operatorApprovals[_msgSender()][to] = approved;
139      emit ApprovalForAll(_msgSender(), to, approved);
140  }
```

 The code meets the specification.

Formal Verification Request 191

Method will not encounter an assertion failure.

 23, Dec 2019

 0.46 ms

Line 148 in File ERC721.sol

```
148  //@CTK_NO_ASF
```

Line 151-153 in File ERC721.sol

```
151     function isApprovedForAll(address owner, address operator) public view returns (bool) {
152         return _operatorApprovals[owner][operator];
153     }
```

✓ The code meets the specification.

Formal Verification Request 192

If method completes, integer overflow would not happen.

📅 23, Dec 2019

🕒 0.48 ms

Line 149 in File ERC721.sol

```
149     //@CTK_NO_OVERFLOW
```

Line 151-153 in File ERC721.sol

```
151     function isApprovedForAll(address owner, address operator) public view returns (bool) {
152         return _operatorApprovals[owner][operator];
153     }
```

✓ The code meets the specification.

Formal Verification Request 193

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 0.66 ms

Line 150 in File ERC721.sol

```
150     //@CTK_NO_BUF_OVERFLOW
```

Line 151-153 in File ERC721.sol

```
151     function isApprovedForAll(address owner, address operator) public view returns (bool) {
152         return _operatorApprovals[owner][operator];
153     }
```

✓ The code meets the specification.

Formal Verification Request 194

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.69 ms

Line 228 in File ERC721.sol

```
228     //@CTK_NO_ASF
```

Line 231-234 in File ERC721.sol


```
231     function _exists(uint256 tokenId) internal view returns (bool) {
232         address owner = _tokenOwner[tokenId];
233         return owner != address(0);
234     }
```

✓ The code meets the specification.

Formal Verification Request 195

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.42 ms

Line 229 in File ERC721.sol

```
229     //@CTK NO_OVERFLOW
```

Line 231-234 in File ERC721.sol


```
231     function _exists(uint256 tokenId) internal view returns (bool) {
232         address owner = _tokenOwner[tokenId];
233         return owner != address(0);
234     }
```

✓ The code meets the specification.

Formal Verification Request 196

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.38 ms

Line 230 in File ERC721.sol

```
230     //@CTK NO_BUF_OVERFLOW
```

Line 231-234 in File ERC721.sol


```
231     function _exists(uint256 tokenId) internal view returns (bool) {
232         address owner = _tokenOwner[tokenId];
233         return owner != address(0);
234     }
```

✓ The code meets the specification.

Formal Verification Request 197

Method will not encounter an assertion failure.

 23, Dec 2019

 128.92 ms

Line 243 in File ERC721.sol

243 //CTK NO_ASF

Line 246-250 in File ERC721.sol

```
246 function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (
    bool) {
247     require(_exists(tokenId), "ERC721: operator query for nonexistent token");
248     address owner = ownerOf(tokenId);
249     return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(
        owner, spender));
250 }
```

✓ The code meets the specification.

Formal Verification Request 198

If method completes, integer overflow would not happen.



23, Dec 2019



5.64 ms

Line 244 in File ERC721.sol

244 //CTK NO_OVERFLOW

Line 246-250 in File ERC721.sol

```
246 function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (
    bool) {
247     require(_exists(tokenId), "ERC721: operator query for nonexistent token");
248     address owner = ownerOf(tokenId);
249     return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(
        owner, spender));
250 }
```

✓ The code meets the specification.

Formal Verification Request 199

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



4.6 ms

Line 245 in File ERC721.sol

245 //CTK NO_BUF_OVERFLOW

Line 246-250 in File ERC721.sol


```
246 function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (
    bool) {
247     require(_exists(tokenId), "ERC721: operator query for nonexistent token");
248     address owner = ownerOf(tokenId);
249     return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(
        owner, spender));
250 }
```

✓ The code meets the specification.

Formal Verification Request 200

Method will not encounter an assertion failure.

 23, Dec 2019

 75.23 ms

Line 290 in File ERC721.sol

290 `//@CTK NO_ASF`

Line 292-300 in File ERC721.sol


```
292     function _mint(address to, uint256 tokenId) internal {
293         require(to != address(0), "ERC721: mint to the zero address");
294         require(!_exists(tokenId), "ERC721: token already minted");
295
296         _tokenOwner[tokenId] = to;
297         _ownedTokensCount[to].increment();
298
299         emit Transfer(address(0), to, tokenId);
300     }
```

 The code meets the specification.

Formal Verification Request 201

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 3.56 ms

Line 291 in File ERC721.sol

291 `//@CTK NO_BUF_OVERFLOW`

Line 292-300 in File ERC721.sol


```
292     function _mint(address to, uint256 tokenId) internal {
293         require(to != address(0), "ERC721: mint to the zero address");
294         require(!_exists(tokenId), "ERC721: token already minted");
295
296         _tokenOwner[tokenId] = to;
297         _ownedTokensCount[to].increment();
298
299         emit Transfer(address(0), to, tokenId);
300     }
```

 The code meets the specification.

Formal Verification Request 202

Method will not encounter an assertion failure.

 23, Dec 2019

 11.2 ms

Line 378 in File ERC721.sol

378 `//@CTK NO_ASF`

Line 381-385 in File ERC721.sol


```
381     function _clearApproval(uint256 tokenId) private {
382         if (_tokenApprovals[tokenId] != address(0)) {
383             _tokenApprovals[tokenId] = address(0);
384         }
385     }
```

✓ The code meets the specification.

Formal Verification Request 203

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.66 ms

Line 379 in File ERC721.sol

379 `//@CTK NO_OVERFLOW`

Line 381-385 in File ERC721.sol


```
381     function _clearApproval(uint256 tokenId) private {
382         if (_tokenApprovals[tokenId] != address(0)) {
383             _tokenApprovals[tokenId] = address(0);
384         }
385     }
```

✓ The code meets the specification.

Formal Verification Request 204

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.52 ms

Line 380 in File ERC721.sol

380 `//@CTK NO_BUF_OVERFLOW`

Line 381-385 in File ERC721.sol


```
381     function _clearApproval(uint256 tokenId) private {
382         if (_tokenApprovals[tokenId] != address(0)) {
383             _tokenApprovals[tokenId] = address(0);
384         }
385     }
```

✓ The code meets the specification.

Formal Verification Request 205

Method will not encounter an assertion failure.

 23, Dec 2019

 51.21 ms

Line 21 in File ERC165.sol

21 `//@CTK NO_ASF`

Line 24-28 in File ERC165.sol


```
24     constructor () internal {
25         // Derived contracts need only register support for their own interfaces,
26         // we register support for ERC165 itself here
27         _registerInterface(_INTERFACE_ID_ERC165);
28     }
```

 The code meets the specification.

Formal Verification Request 206

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.57 ms

Line 22 in File ERC165.sol

22 `//@CTK NO_OVERFLOW`

Line 24-28 in File ERC165.sol


```
24     constructor () internal {
25         // Derived contracts need only register support for their own interfaces,
26         // we register support for ERC165 itself here
27         _registerInterface(_INTERFACE_ID_ERC165);
28     }
```

 The code meets the specification.

Formal Verification Request 207

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.57 ms

Line 23 in File ERC165.sol

23 `//@CTK NO_BUF_OVERFLOW`

Line 24-28 in File ERC165.sol

```
24     constructor () internal {
25         // Derived contracts need only register support for their own interfaces,
26         // we register support for ERC165 itself here
27         _registerInterface(_INTERFACE_ID_ERC165);
28     }
```


✓ The code meets the specification.

Formal Verification Request 208

Method will not encounter an assertion failure.



23, Dec 2019



6.0 ms

Line 35 in File ERC165.sol

```
35  // @CTK NO_ASF
```

Line 38-40 in File ERC165.sol

```
38  function supportsInterface(bytes4 interfaceId) external view returns (bool) {
39      return _supportedInterfaces[interfaceId];
40  }
```

✓ The code meets the specification.

Formal Verification Request 209

If method completes, integer overflow would not happen.



23, Dec 2019



0.44 ms

Line 36 in File ERC165.sol

```
36  // @CTK NO_OVERFLOW
```

Line 38-40 in File ERC165.sol

```
38  function supportsInterface(bytes4 interfaceId) external view returns (bool) {
39      return _supportedInterfaces[interfaceId];
40  }
```

✓ The code meets the specification.

Formal Verification Request 210

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.37 ms

Line 37 in File ERC165.sol

```
37  // @CTK NO_BUF_OVERFLOW
```

Line 38-40 in File ERC165.sol


```
38  function supportsInterface(bytes4 interfaceId) external view returns (bool) {
39      return _supportedInterfaces[interfaceId];
40  }
```

✓ The code meets the specification.

Formal Verification Request 211

Method will not encounter an assertion failure.

 23, Dec 2019

 0.52 ms

Line 53 in File ERC165.sol

53 `//@CTK NO_ASF`

Line 56-59 in File ERC165.sol


```
56 function _registerInterface(bytes4 interfaceId) internal {  
57     require(interfaceId != 0xffffffff, "ERC165: invalid interface id");  
58     _supportedInterfaces[interfaceId] = true;  
59 }
```

 The code meets the specification.

Formal Verification Request 212

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.44 ms

Line 54 in File ERC165.sol

54 `//@CTK NO_OVERFLOW`

Line 56-59 in File ERC165.sol


```
56 function _registerInterface(bytes4 interfaceId) internal {  
57     require(interfaceId != 0xffffffff, "ERC165: invalid interface id");  
58     _supportedInterfaces[interfaceId] = true;  
59 }
```

 The code meets the specification.

Formal Verification Request 213

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.44 ms

Line 55 in File ERC165.sol

55 `//@CTK NO_BUF_OVERFLOW`

Line 56-59 in File ERC165.sol

```
56 function _registerInterface(bytes4 interfaceId) internal {  
57     require(interfaceId != 0xffffffff, "ERC165: invalid interface id");  
58     _supportedInterfaces[interfaceId] = true;  
59 }
```

 The code meets the specification.

Formal Verification Request 214

Method will not encounter an assertion failure.



23, Dec 2019



3.61 ms

Line 17 in File ERC20Burnable.sol

```
17  //@CTK NO_ASF
```

Line 20-24 in File ERC20Burnable.sol

```
20  function burn(uint256 amount) public {
21      _burn(_msgSender(), amount);
22  }
```

✓ The code meets the specification.

Formal Verification Request 215

If method completes, integer overflow would not happen.



23, Dec 2019



0.36 ms

Line 18 in File ERC20Burnable.sol

```
18  //@CTK NO_OVERFLOW
```

Line 20-24 in File ERC20Burnable.sol

```
20  function burn(uint256 amount) public {
21      _burn(_msgSender(), amount);
22  }
```

✓ The code meets the specification.

Formal Verification Request 216

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.34 ms

Line 19 in File ERC20Burnable.sol

```
19  //@CTK NO_BUF_OVERFLOW
```

Line 20-24 in File ERC20Burnable.sol

```
20  function burn(uint256 amount) public {
21      _burn(_msgSender(), amount);
22  }
```

✓ The code meets the specification.

Formal Verification Request 217

Method will not encounter an assertion failure.



23, Dec 2019



3.47 ms

Line 29 in File ERC20Burnable.sol

```
29  //@CTK NO_ASF
```

Line 32-36 in File ERC20Burnable.sol

```
32  function burnFrom(address account, uint256 amount) public {  
33      _burnFrom(account, amount);  
34  }
```

✓ The code meets the specification.

Formal Verification Request 218

If method completes, integer overflow would not happen.



23, Dec 2019



0.36 ms

Line 30 in File ERC20Burnable.sol

```
30  //@CTK NO_OVERFLOW
```

Line 32-36 in File ERC20Burnable.sol

```
32  function burnFrom(address account, uint256 amount) public {  
33      _burnFrom(account, amount);  
34  }
```

✓ The code meets the specification.

Formal Verification Request 219

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



0.37 ms

Line 31 in File ERC20Burnable.sol

```
31  //@CTK NO_BUF_OVERFLOW
```

Line 32-36 in File ERC20Burnable.sol


```
32  function burnFrom(address account, uint256 amount) public {  
33      _burnFrom(account, amount);  
34  }
```

✓ The code meets the specification.

Formal Verification Request 220

Method will not encounter an assertion failure.

 23, Dec 2019

 3.5 ms

Line 16 in File Context.sol

16 `//@CTK NO_ASF`

Line 19 in File Context.sol


19 `constructor () internal { }`

 The code meets the specification.

Formal Verification Request 221

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.35 ms

Line 17 in File Context.sol

17 `//@CTK NO_OVERFLOW`

Line 19 in File Context.sol


19 `constructor () internal { }`

 The code meets the specification.

Formal Verification Request 222

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.33 ms

Line 18 in File Context.sol

18 `//@CTK NO_BUF_OVERFLOW`

Line 19 in File Context.sol


19 `constructor () internal { }`

 The code meets the specification.

Formal Verification Request 223

Method will not encounter an assertion failure.

 23, Dec 2019

 5.28 ms

Line 21 in File Context.sol

21 `//@CTK NO_ASF`

Line 24-26 in File Context.sol


```
24 function _msgSender() internal view returns (address payable) {  
25     return msg.sender;  
26 }
```

 The code meets the specification.

Formal Verification Request 224

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.39 ms

Line 22 in File Context.sol

22 `//@CTK NO_OVERFLOW`

Line 24-26 in File Context.sol


```
24 function _msgSender() internal view returns (address payable) {  
25     return msg.sender;  
26 }
```

 The code meets the specification.

Formal Verification Request 225

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.46 ms

Line 23 in File Context.sol

23 `//@CTK NO_BUF_OVERFLOW`

Line 24-26 in File Context.sol


```
24 function _msgSender() internal view returns (address payable) {  
25     return msg.sender;  
26 }
```

 The code meets the specification.

Formal Verification Request 226

Method will not encounter an assertion failure.

 23, Dec 2019

 4.72 ms

Line 25 in File Counters.sol

25 `//@CTK NO_ASF`

Line 28-30 in File Counters.sol


```
28 function current(Counter storage counter) internal view returns (uint256) {  
29     return counter._value;  
30 }
```

 The code meets the specification.

Formal Verification Request 227

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.38 ms

Line 26 in File Counters.sol

26 `//@CTK NO_OVERFLOW`

Line 28-30 in File Counters.sol


```
28 function current(Counter storage counter) internal view returns (uint256) {  
29     return counter._value;  
30 }
```

 The code meets the specification.

Formal Verification Request 228

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.39 ms

Line 27 in File Counters.sol

27 `//@CTK NO_BUF_OVERFLOW`

Line 28-30 in File Counters.sol


```
28 function current(Counter storage counter) internal view returns (uint256) {  
29     return counter._value;  
30 }
```

 The code meets the specification.

Formal Verification Request 229

Method will not encounter an assertion failure.

 23, Dec 2019

 5.44 ms

Line 31 in File Counters.sol

31 `//@CTK NO_ASF`

Line 33-36 in File Counters.sol


```
33 function increment(Counter storage counter) internal {  
34     // The {SafeMath} overflow check can be skipped here, see the comment at the top  
35     counter._value += 1;  
36 }
```

 The code meets the specification.

Formal Verification Request 230

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 0.38 ms

Line 32 in File Counters.sol

32 `//@CTK NO_BUF_OVERFLOW`

Line 33-36 in File Counters.sol


```
33 function increment(Counter storage counter) internal {  
34     // The {SafeMath} overflow check can be skipped here, see the comment at the top  
35     counter._value += 1;  
36 }
```

 The code meets the specification.

Formal Verification Request 231

Method will not encounter an assertion failure.

 04, Dec 2019

 5.3 ms

Line 5 in File Ownable.sol

5 `//@CTK NO_ASF`

Line 8-10 in File Ownable.sol


```
8 constructor() public {  
9     owner = msg.sender;  
10 }
```

 The code meets the specification.

Formal Verification Request 232

If method completes, integer overflow would not happen.

 04, Dec 2019

 0.41 ms

Line 6 in File Ownable.sol

```
6 // @CTK_NO_OVERFLOW
```

Line 8-10 in File Ownable.sol


```
8 constructor() public {  
9     owner = msg.sender;  
10 }
```

 The code meets the specification.

Formal Verification Request 233

Buffer overflow / array index out of bound would never happen.

 04, Dec 2019

 0.36 ms

Line 7 in File Ownable.sol

```
7 // @CTK_NO_BUF_OVERFLOW
```

Line 8-10 in File Ownable.sol


```
8 constructor() public {  
9     owner = msg.sender;  
10 }
```

 The code meets the specification.

Formal Verification Request 234

Method will not encounter an assertion failure.

 04, Dec 2019

 17.54 ms

Line 16 in File Ownable.sol

```
16 // @CTK_NO_ASF
```

Line 19-22 in File Ownable.sol


```
19 function transferOwnership(address newOwner) public onlyOwner {  
20     if (newOwner != address(0))  
21         owner = newOwner;  
22 }
```

 The code meets the specification.

Formal Verification Request 235

If method completes, integer overflow would not happen.

 04, Dec 2019

 0.47 ms

Line 17 in File Ownable.sol

```
17 // @CTK_NO_OVERFLOW
```

Line 19-22 in File Ownable.sol


```
19 function transferOwnership(address newOwner) public onlyOwner {
20     if (newOwner != address(0))
21         owner = newOwner;
22 }
```

 The code meets the specification.

Formal Verification Request 236

Buffer overflow / array index out of bound would never happen.

 04, Dec 2019

 0.48 ms

Line 18 in File Ownable.sol

```
18 // @CTK_NO_BUF_OVERFLOW
```

Line 19-22 in File Ownable.sol


```
19 function transferOwnership(address newOwner) public onlyOwner {
20     if (newOwner != address(0))
21         owner = newOwner;
22 }
```

 The code meets the specification.

Formal Verification Request 237

isContract

 04, Dec 2019

 6.21 ms

Line 18-20 in File Address.sol

```
18 /* @CTK_isContract
19     @post !__reverted -> __return == (account != msg.sender)
20 */
```

Line 24-40 in File Address.sol

```

24 function isContract(address account) internal view returns (bool) {
25     return (account != msg.sender);
26     /*
27     // This method relies in extcodesize, which returns 0 for contracts in
28     // construction, since the code is only stored at the end of the
29     // constructor execution.
30
31     // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
32     // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
33     // returned
34     // for accounts without code, i.e. `keccak256('')`
35     bytes32 codehash;
36     bytes32 accountHash = 0
37         xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
38     // solhint-disable-next-line no-inline-assembly
39     assembly { codehash := extcodehash(account) }
40     return (codehash != 0x0 && codehash != accountHash);
41     */
42 }

```

✓ The code meets the specification.

Formal Verification Request 238

Buffer overflow / array index out of bound would never happen.

📅 04, Dec 2019

🕒 0.4 ms

Line 21 in File Address.sol

```
21 //CTK NO_BUF_OVERFLOW
```

Line 24-40 in File Address.sol

```

24 function isContract(address account) internal view returns (bool) {
25     return (account != msg.sender);
26     /*
27     // This method relies in extcodesize, which returns 0 for contracts in
28     // construction, since the code is only stored at the end of the
29     // constructor execution.
30
31     // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
32     // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
33     // returned
34     // for accounts without code, i.e. `keccak256('')`
35     bytes32 codehash;
36     bytes32 accountHash = 0
37         xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
38     // solhint-disable-next-line no-inline-assembly
39     assembly { codehash := extcodehash(account) }
40     return (codehash != 0x0 && codehash != accountHash);
41     */
42 }


```

✓ The code meets the specification.

Formal Verification Request 239

If method completes, integer overflow would not happen.

 04, Dec 2019

 0.4 ms

Line 22 in File Address.sol

22 `//@CTK NO_OVERFLOW`

Line 24-40 in File Address.sol


```
24 function isContract(address account) internal view returns (bool) {
25     return (account != msg.sender);
26     /*
27     // This method relies in extcodesize, which returns 0 for contracts in
28     // construction, since the code is only stored at the end of the
29     // constructor execution.
30
31     // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
32     // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
33     // returned
34     // for accounts without code, i.e. `keccak256(')`
35     bytes32 codehash;
36     bytes32 accountHash = 0
37         xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
38     // solhint-disable-next-line no-inline-assembly
39     assembly { codehash := extcodehash(account) }
40     return (codehash != 0x0 && codehash != accountHash);
41     */
42 }
```

 The code meets the specification.

Formal Verification Request 240

Method will not encounter an assertion failure.

 04, Dec 2019

 0.38 ms

Line 23 in File Address.sol

23 `//@CTK NO_ASF`

Line 24-40 in File Address.sol

```
24 function isContract(address account) internal view returns (bool) {
25     return (account != msg.sender);
26     /*
27     // This method relies in extcodesize, which returns 0 for contracts in
28     // construction, since the code is only stored at the end of the
29     // constructor execution.
30
31     // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
32     // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
33     // returned
34     // for accounts without code, i.e. `keccak256(')`
35     bytes32 codehash;
36     bytes32 accountHash = 0
37         xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
38     // solhint-disable-next-line no-inline-assembly
39     assembly { codehash := extcodehash(account) }
40     return (codehash != 0x0 && codehash != accountHash);
41     */
42 }
```

```

34     bytes32 codehash;
35     bytes32 accountHash = 0
        xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
36     // solhint-disable-next-line no-inline-assembly
37     assembly { codehash := extcodehash(account) }
38     return (codehash != 0x0 && codehash != accountHash);
39     */
40 }

```

✓ The code meets the specification.

Formal Verification Request 241

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 7.45 ms

Line 13 in File BytesLib.sol

```
13 //CTK NO_BUF_OVERFLOW
```

Line 16-92 in File BytesLib.sol

```

16     function concat(
17         bytes memory _preBytes,
18         bytes memory _postBytes
19     )
20     internal
21     pure
22     returns (bytes memory)
23     {
24         bytes memory tempBytes;
25
26         assembly {
27             // Get a location of some free memory and store it in tempBytes as
28             // Solidity does for memory variables.
29             tempBytes := mload(0x40)
30
31             // Store the length of the first bytes array at the beginning of
32             // the memory for tempBytes.
33             let length := mload(_preBytes)
34             mstore(tempBytes, length)
35
36             // Maintain a memory counter for the current write location in the
37             // temp bytes array by adding the 32 bytes for the array length to
38             // the starting location.
39             let mc := add(tempBytes, 0x20)
40             // Stop copying when the memory counter reaches the length of the
41             // first bytes array.
42             let end := add(mc, length)
43
44             for {
45                 // Initialize a copy counter to the start of the _preBytes data,
46                 // 32 bytes into its memory.
47                 let cc := add(_preBytes, 0x20)
48             } lt(mc, end) {
49                 // Increase both counters by 32 bytes each iteration.

```

```

50         mc := add(mc, 0x20)
51         cc := add(cc, 0x20)
52     } {
53         // Write the _preBytes data into the tempBytes memory 32 bytes
54         // at a time.
55         mstore(mc, mload(cc))
56     }
57
58     // Add the length of _postBytes to the current length of tempBytes
59     // and store it as the new length in the first 32 bytes of the
60     // tempBytes memory.
61     length := mload(_postBytes)
62     mstore(tempBytes, add(length, mload(tempBytes)))
63
64     // Move the memory counter back from a multiple of 0x20 to the
65     // actual end of the _preBytes data.
66     mc := end
67     // Stop copying when the memory counter reaches the new combined
68     // length of the arrays.
69     end := add(mc, length)
70
71     for {
72         let cc := add(_postBytes, 0x20)
73     } lt(mc, end) {
74         mc := add(mc, 0x20)
75         cc := add(cc, 0x20)
76     } {
77         mstore(mc, mload(cc))
78     }
79
80     // Update the free-memory pointer by padding our last write location
81     // to 32 bytes: add 31 bytes to the end of tempBytes to move to the
82     // next 32 byte block, then round down to the nearest multiple of
83     // 32. If the sum of the length of the two arrays is zero then add
84     // one before rounding down to leave a blank 32 bytes (the length block with 0).
85     mstore(0x40, and(
86         add(add(end, iszero(add(length, mload(_preBytes)))), 31),
87         not(31) // Round down to the nearest 32 bytes.
88     ))
89 }
90
91 return tempBytes;
92 }

```

✓ The code meets the specification.

Formal Verification Request 242

If method completes, integer overflow would not happen.

📅 23, Dec 2019

🕒 0.41 ms

Line 14 in File BytesLib.sol

14 // @CTK NO_OVERFLOW

Line 16-92 in File BytesLib.sol

```

16  function concat(
17      bytes memory _preBytes,
18      bytes memory _postBytes
19  )
20      internal
21      pure
22      returns (bytes memory)
23  {
24      bytes memory tempBytes;
25
26      assembly {
27          // Get a location of some free memory and store it in tempBytes as
28          // Solidity does for memory variables.
29          tempBytes := mload(0x40)
30
31          // Store the length of the first bytes array at the beginning of
32          // the memory for tempBytes.
33          let length := mload(_preBytes)
34          mstore(tempBytes, length)
35
36          // Maintain a memory counter for the current write location in the
37          // temp bytes array by adding the 32 bytes for the array length to
38          // the starting location.
39          let mc := add(tempBytes, 0x20)
40          // Stop copying when the memory counter reaches the length of the
41          // first bytes array.
42          let end := add(mc, length)
43
44          for {
45              // Initialize a copy counter to the start of the _preBytes data,
46              // 32 bytes into its memory.
47              let cc := add(_preBytes, 0x20)
48          } lt(mc, end) {
49              // Increase both counters by 32 bytes each iteration.
50              mc := add(mc, 0x20)
51              cc := add(cc, 0x20)
52          } {
53              // Write the _preBytes data into the tempBytes memory 32 bytes
54              // at a time.
55              mstore(mc, mload(cc))
56          }
57
58          // Add the length of _postBytes to the current length of tempBytes
59          // and store it as the new length in the first 32 bytes of the
60          // tempBytes memory.
61          length := mload(_postBytes)
62          mstore(tempBytes, add(length, mload(tempBytes)))
63
64          // Move the memory counter back from a multiple of 0x20 to the
65          // actual end of the _preBytes data.
66          mc := end
67          // Stop copying when the memory counter reaches the new combined
68          // length of the arrays.
69          end := add(mc, length)
70
71          for {
72              let cc := add(_postBytes, 0x20)

```

```

73     } lt(mc, end) {
74         mc := add(mc, 0x20)
75         cc := add(cc, 0x20)
76     } {
77         mstore(mc, mload(cc))
78     }
79
80     // Update the free-memory pointer by padding our last write location
81     // to 32 bytes: add 31 bytes to the end of tempBytes to move to the
82     // next 32 byte block, then round down to the nearest multiple of
83     // 32. If the sum of the length of the two arrays is zero then add
84     // one before rounding down to leave a blank 32 bytes (the length block with 0).
85     mstore(0x40, and(
86         add(add(end, iszero(add(length, mload(_preBytes)))), 31),
87         not(31) // Round down to the nearest 32 bytes.
88     ))
89 }
90
91 return tempBytes;
92 }

```

✓ The code meets the specification.

Formal Verification Request 243

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.39 ms

Line 15 in File BytesLib.sol

```
15 //CTK NO_ASF
```

Line 16-92 in File BytesLib.sol

```

16 function concat(
17     bytes memory _preBytes,
18     bytes memory _postBytes
19 )
20     internal
21     pure
22     returns (bytes memory)
23 {
24     bytes memory tempBytes;
25
26     assembly {
27         // Get a location of some free memory and store it in tempBytes as
28         // Solidity does for memory variables.
29         tempBytes := mload(0x40)
30
31         // Store the length of the first bytes array at the beginning of
32         // the memory for tempBytes.
33         let length := mload(_preBytes)
34         mstore(tempBytes, length)
35
36         // Maintain a memory counter for the current write location in the
37         // temp bytes array by adding the 32 bytes for the array length to

```



```

38      // the starting location.
39      let mc := add(tempBytes, 0x20)
40      // Stop copying when the memory counter reaches the length of the
41      // first bytes array.
42      let end := add(mc, length)
43
44      for {
45          // Initialize a copy counter to the start of the _preBytes data,
46          // 32 bytes into its memory.
47          let cc := add(_preBytes, 0x20)
48      } lt(mc, end) {
49          // Increase both counters by 32 bytes each iteration.
50          mc := add(mc, 0x20)
51          cc := add(cc, 0x20)
52      } {
53          // Write the _preBytes data into the tempBytes memory 32 bytes
54          // at a time.
55          mstore(mc, mload(cc))
56      }
57
58      // Add the length of _postBytes to the current length of tempBytes
59      // and store it as the new length in the first 32 bytes of the
60      // tempBytes memory.
61      length := mload(_postBytes)
62      mstore(tempBytes, add(length, mload(tempBytes)))
63
64      // Move the memory counter back from a multiple of 0x20 to the
65      // actual end of the _preBytes data.
66      mc := end
67      // Stop copying when the memory counter reaches the new combined
68      // length of the arrays.
69      end := add(mc, length)
70
71      for {
72          let cc := add(_postBytes, 0x20)
73      } lt(mc, end) {
74          mc := add(mc, 0x20)
75          cc := add(cc, 0x20)
76      } {
77          mstore(mc, mload(cc))
78      }
79
80      // Update the free-memory pointer by padding our last write location
81      // to 32 bytes: add 31 bytes to the end of tempBytes to move to the
82      // next 32 byte block, then round down to the nearest multiple of
83      // 32. If the sum of the length of the two arrays is zero then add
84      // one before rounding down to leave a blank 32 bytes (the length block with 0).
85      mstore(0x40, and(
86          add(add(end, iszero(add(length, mload(_preBytes))))), 31),
87          not(31) // Round down to the nearest 32 bytes.
88      ))
89  }
90
91  return tempBytes;
92  }


```

✓ The code meets the specification.

Formal Verification Request 244

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 4.0 ms

Line 93 in File BytesLib.sol

93 `//@CTK NO_BUF_OVERFLOW`

Line 96-231 in File BytesLib.sol

```

96  function concatStorage(bytes storage _preBytes, bytes memory _postBytes) internal {
97      assembly {
98          // Read the first 32 bytes of _preBytes storage, which is the length
99          // of the array. (We don't need to use the offset into the slot
100         // because arrays use the entire slot.)
101         let fslot := sload(_preBytes_slot)
102         // Arrays of 31 bytes or less have an even value in their slot,
103         // while longer arrays have an odd value. The actual length is
104         // the slot divided by two for odd values, and the lowest order
105         // byte divided by two for even values.
106         // If the slot is even, bitwise and the slot with 255 and divide by
107         // two to get the length. If the slot is odd, bitwise and the slot
108         // with -1 and divide by two.
109         let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
110         let mlength := mload(_postBytes)
111         let newlength := add(slength, mlength)
112         // slength can contain both the length and contents of the array
113         // if length < 32 bytes so let's prepare for that
114         // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-state-variables-in-storage
115         switch add(lt(slength, 32), lt(newlength, 32))
116         case 2 {
117             // Since the new array still fits in the slot, we just need to
118             // update the contents of the slot.
119             // uint256(bytes_storage) = uint256(bytes_storage) + uint256(bytes_memory) +
120             //   new_length
121             sstore(
122                 _preBytes_slot,
123                 // all the modifications to the slot are inside this
124                 // next block
125                 add(
126                     // we can just add to the slot contents because the
127                     // bytes we want to change are the LSBs
128                     fslot,
129                     add(
130                         mul(
131                             div(
132                                 // load the bytes from memory
133                                 mload(add(_postBytes, 0x20)),
134                                 // zero all bytes to the right
135                                 exp(0x100, sub(32, mlength))
136                             ),
137                             // and now shift left the number of bytes to
138                             // leave space for the length in the slot
139                             exp(0x100, sub(32, newlength))
140                         )
141                     )
142                 )
143             )
144         }
145     }
146 }

```

```

140         // increase length by the double of the memory
141         // bytes length
142         mul(mlength, 2)
143     )
144 )
145 )
146 }
147 case 1 {
148     // The stored value fits in the slot, but the combined value
149     // will exceed it.
150     // get the keccak hash to get the contents of the array
151     mstore(0x0, _preBytes_slot)
152     let sc := add(keccak256(0x0, 0x20), div(slength, 32))
153
154     // save new length
155     sstore(_preBytes_slot, add(mul(newlength, 2), 1))
156
157     // The contents of the _postBytes array start 32 bytes into
158     // the structure. Our first read should obtain the `submod`
159     // bytes that can fit into the unused space in the last word
160     // of the stored array. To get this, we read 32 bytes starting
161     // from `submod`, so the data we read overlaps with the array
162     // contents by `submod` bytes. Masking the lowest-order
163     // `submod` bytes allows us to add that value directly to the
164     // stored value.
165
166     let submod := sub(32, slength)
167     let mc := add(_postBytes, submod)
168     let end := add(_postBytes, mlength)
169     let mask := sub(exp(0x100, submod), 1)
170
171     sstore(
172         sc,
173         add(
174             and(
175                 fslot,
176                 0xffffffffffffffffffffffffffffffffffffffffffffffff00
177             ),
178             and(mload(mc), mask)
179         )
180     )
181
182     for {
183         mc := add(mc, 0x20)
184         sc := add(sc, 1)
185     } lt(mc, end) {
186         sc := add(sc, 1)
187         mc := add(mc, 0x20)
188     } {
189         sstore(sc, mload(mc))
190     }
191
192     mask := exp(0x100, sub(mc, end))
193
194     sstore(sc, mul(div(mload(mc), mask), mask))
195 }
196 default {
197     // get the keccak hash to get the contents of the array

```

```

198     mstore(0x0, _preBytes_slot)
199     // Start copying to the last used word of the stored array.
200     let sc := add(keccak256(0x0, 0x20), div(slength, 32))
201
202     // save new length
203     sstore(_preBytes_slot, add(mul(newlength, 2), 1))
204
205     // Copy over the first `submod` bytes of the new data as in
206     // case 1 above.
207     let slengthmod := mod(slength, 32)
208     let mlengthmod := mod(mlength, 32)
209     let submod := sub(32, slengthmod)
210     let mc := add(_postBytes, submod)
211     let end := add(_postBytes, mlength)
212     let mask := sub(exp(0x100, submod), 1)
213
214     sstore(sc, add(sload(sc), and(mload(mc), mask)))
215
216     for {
217         sc := add(sc, 1)
218         mc := add(mc, 0x20)
219     } lt(mc, end) {
220         sc := add(sc, 1)
221         mc := add(mc, 0x20)
222     } {
223         sstore(sc, mload(mc))
224     }
225
226     mask := exp(0x100, sub(mc, end))
227
228     sstore(sc, mul(div(mload(mc), mask), mask))
229 }
230 }
231 }


```

✓ The code meets the specification.

Formal Verification Request 245

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.37 ms

Line 94 in File BytesLib.sol

```
94 // @CTK NO_OVERFLOW
```

Line 96-231 in File BytesLib.sol

```

96 function concatStorage(bytes storage _preBytes, bytes memory _postBytes) internal {
97     assembly {
98         // Read the first 32 bytes of _preBytes storage, which is the length
99         // of the array. (We don't need to use the offset into the slot
100         // because arrays use the entire slot.)
101         let fslot := sload(_preBytes_slot)
102         // Arrays of 31 bytes or less have an even value in their slot,
103         // while longer arrays have an odd value. The actual length is

```

```

104 // the slot divided by two for odd values, and the lowest order
105 // byte divided by two for even values.
106 // If the slot is even, bitwise and the slot with 255 and divide by
107 // two to get the length. If the slot is odd, bitwise and the slot
108 // with -1 and divide by two.
109 let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
110 let mlength := mload(_postBytes)
111 let newlength := add(slength, mlength)
112 // slength can contain both the length and contents of the array
113 // if length < 32 bytes so let's prepare for that
114 // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-state-variables-in-storage
115 switch add(1t(slength, 32), 1t(newlength, 32))
116 case 2 {
117     // Since the new array still fits in the slot, we just need to
118     // update the contents of the slot.
119     // uint256(bytes_storage) = uint256(bytes_storage) + uint256(bytes_memory) +
120     // new_length
121     sstore(
122         _preBytes_slot,
123         // all the modifications to the slot are inside this
124         // next block
125         add(
126             // we can just add to the slot contents because the
127             // bytes we want to change are the LSBs
128             fslot,
129             add(
130                 mul(
131                     div(
132                         // load the bytes from memory
133                         mload(add(_postBytes, 0x20)),
134                         // zero all bytes to the right
135                         exp(0x100, sub(32, mlength))
136                     ),
137                     // and now shift left the number of bytes to
138                     // leave space for the length in the slot
139                     exp(0x100, sub(32, newlength))
140                 ),
141                 // increase length by the double of the memory
142                 // bytes length
143                 mul(mlength, 2)
144             )
145         )
146     }
147 case 1 {
148     // The stored value fits in the slot, but the combined value
149     // will exceed it.
150     // get the keccak hash to get the contents of the array
151     mstore(0x0, _preBytes_slot)
152     let sc := add(keccak256(0x0, 0x20), div(slength, 32))
153
154     // save new length
155     sstore(_preBytes_slot, add(mul(newlength, 2), 1))
156
157     // The contents of the _postBytes array start 32 bytes into
158     // the structure. Our first read should obtain the `submod`
159     // bytes that can fit into the unused space in the last word

```

```

160      // of the stored array. To get this, we read 32 bytes starting
161      // from `submod`, so the data we read overlaps with the array
162      // contents by `submod` bytes. Masking the lowest-order
163      // `submod` bytes allows us to add that value directly to the
164      // stored value.
165
166      let submod := sub(32, slength)
167      let mc := add(_postBytes, submod)
168      let end := add(_postBytes, mlength)
169      let mask := sub(exp(0x100, submod), 1)
170
171      sstore(
172        sc,
173        add(
174          and(
175            fslot,
176            0xfffffffffffffffffffffffffffffffffffffffffffffffff00
177          ),
178          and(mload(mc), mask)
179        )
180      )
181
182      for {
183        mc := add(mc, 0x20)
184        sc := add(sc, 1)
185      } lt(mc, end) {
186        sc := add(sc, 1)
187        mc := add(mc, 0x20)
188      } {
189        sstore(sc, mload(mc))
190      }
191
192      mask := exp(0x100, sub(mc, end))
193
194      sstore(sc, mul(div(mload(mc), mask), mask))
195    }
196    default {
197      // get the keccak hash to get the contents of the array
198      mstore(0x0, _preBytes_slot)
199      // Start copying to the last used word of the stored array.
200      let sc := add(keccak256(0x0, 0x20), div(slength, 32))
201
202      // save new length
203      sstore(_preBytes_slot, add(mul(newlength, 2), 1))
204
205      // Copy over the first `submod` bytes of the new data as in
206      // case 1 above.
207      let slengthmod := mod(slength, 32)
208      let mlengthmod := mod(mlength, 32)
209      let submod := sub(32, slengthmod)
210      let mc := add(_postBytes, submod)
211      let end := add(_postBytes, mlength)
212      let mask := sub(exp(0x100, submod), 1)
213
214      sstore(sc, add(sload(sc), and(mload(mc), mask)))
215
216      for {
217        sc := add(sc, 1)

```

```

218         mc := add(mc, 0x20)
219     } lt(mc, end) {
220         sc := add(sc, 1)
221         mc := add(mc, 0x20)
222     } {
223         sstore(sc, mload(mc))
224     }
225
226     mask := exp(0x100, sub(mc, end))
227
228     sstore(sc, mul(div(mload(mc), mask), mask))
229 }
230 }
231 }

```

✓ The code meets the specification.

Formal Verification Request 246

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.35 ms

Line 95 in File BytesLib.sol

```
95  //CTK NO_ASF
```

Line 96-231 in File BytesLib.sol

```

96  function concatStorage(bytes storage _preBytes, bytes memory _postBytes) internal {
97      assembly {
98          // Read the first 32 bytes of _preBytes storage, which is the length
99          // of the array. (We don't need to use the offset into the slot
100         // because arrays use the entire slot.)
101         let fslot := sload(_preBytes_slot)
102         // Arrays of 31 bytes or less have an even value in their slot,
103         // while longer arrays have an odd value. The actual length is
104         // the slot divided by two for odd values, and the lowest order
105         // byte divided by two for even values.
106         // If the slot is even, bitwise and the slot with 255 and divide by
107         // two to get the length. If the slot is odd, bitwise and the slot
108         // with -1 and divide by two.
109         let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
110         let mlength := mload(_postBytes)
111         let newlength := add(slength, mlength)
112         // slength can contain both the length and contents of the array
113         // if length < 32 bytes so let's prepare for that
114         // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-state-
            variables-in-storage
115         switch add(lt(slength, 32), lt(newlength, 32))
116         case 2 {
117             // Since the new array still fits in the slot, we just need to
118             // update the contents of the slot.
119             // uint256(bytes_storage) = uint256(bytes_storage) + uint256(bytes_memory) +
                new_length
120             sstore(
121                 _preBytes_slot,

```

```

122         // all the modifications to the slot are inside this
123         // next block
124         add(
125             // we can just add to the slot contents because the
126             // bytes we want to change are the LSBs
127             fslot,
128             add(
129                 mul(
130                     div(
131                         // load the bytes from memory
132                         mload(add(_postBytes, 0x20)),
133                         // zero all bytes to the right
134                         exp(0x100, sub(32, mlength))
135                     ),
136                     // and now shift left the number of bytes to
137                     // leave space for the length in the slot
138                     exp(0x100, sub(32, newlength))
139                 ),
140                 // increase length by the double of the memory
141                 // bytes length
142                 mul(mlength, 2)
143             )
144         )
145     )
146 }
147 case 1 {
148     // The stored value fits in the slot, but the combined value
149     // will exceed it.
150     // get the keccak hash to get the contents of the array
151     mstore(0x0, _preBytes_slot)
152     let sc := add(keccak256(0x0, 0x20), div(slength, 32))
153
154     // save new length
155     sstore(_preBytes_slot, add(mul(newlength, 2), 1))
156
157     // The contents of the _postBytes array start 32 bytes into
158     // the structure. Our first read should obtain the `submod`
159     // bytes that can fit into the unused space in the last word
160     // of the stored array. To get this, we read 32 bytes starting
161     // from `submod`, so the data we read overlaps with the array
162     // contents by `submod` bytes. Masking the lowest-order
163     // `submod` bytes allows us to add that value directly to the
164     // stored value.
165
166     let submod := sub(32, slength)
167     let mc := add(_postBytes, submod)
168     let end := add(_postBytes, mlength)
169     let mask := sub(exp(0x100, submod), 1)
170
171     sstore(
172         sc,
173         add(
174             and(
175                 fslot,
176                 0xffffffffffffffffffffffffffffffffffffffffffffffff00
177             ),
178             and(mload(mc), mask)
179         )

```



```

180     )
181
182     for {
183         mc := add(mc, 0x20)
184         sc := add(sc, 1)
185     } lt(mc, end) {
186         sc := add(sc, 1)
187         mc := add(mc, 0x20)
188     } {
189         sstore(sc, mload(mc))
190     }
191
192     mask := exp(0x100, sub(mc, end))
193
194     sstore(sc, mul(div(mload(mc), mask), mask))
195 }
196 default {
197     // get the keccak hash to get the contents of the array
198     mstore(0x0, _preBytes_slot)
199     // Start copying to the last used word of the stored array.
200     let sc := add(keccak256(0x0, 0x20), div(slength, 32))
201
202     // save new length
203     sstore(_preBytes_slot, add(mul(newlength, 2), 1))
204
205     // Copy over the first `submod` bytes of the new data as in
206     // case 1 above.
207     let slengthmod := mod(slength, 32)
208     let mlengthmod := mod(mlength, 32)
209     let submod := sub(32, slengthmod)
210     let mc := add(_postBytes, submod)
211     let end := add(_postBytes, mlength)
212     let mask := sub(exp(0x100, submod), 1)
213
214     sstore(sc, add(sload(sc), and(mload(mc), mask)))
215
216     for {
217         sc := add(sc, 1)
218         mc := add(mc, 0x20)
219     } lt(mc, end) {
220         sc := add(sc, 1)
221         mc := add(mc, 0x20)
222     } {
223         sstore(sc, mload(mc))
224     }
225
226     mask := exp(0x100, sub(mc, end))
227
228     sstore(sc, mul(div(mload(mc), mask), mask))
229 }
230 }
231


```

✓ The code meets the specification.

Formal Verification Request 247

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 20.97 ms

Line 232 in File BytesLib.sol

232 `//@CTK NO_BUF_OVERFLOW`

Line 234-297 in File BytesLib.sol

```

234     function slice(
235         bytes memory _bytes,
236         uint _start,
237         uint _length
238     )
239     internal
240     pure
241     returns (bytes memory)
242     {
243         require(_bytes.length >= (_start + _length), "_bytes.length >= (_start + _length)");
244
245         bytes memory tempBytes;
246
247         assembly {
248             switch iszero(_length)
249             case 0 {
250                 // Get a location of some free memory and store it in tempBytes as
251                 // Solidity does for memory variables.
252                 tempBytes := mload(0x40)
253
254                 // The first word of the slice result is potentially a partial
255                 // word read from the original array. To read it, we calculate
256                 // the length of that partial word and start copying that many
257                 // bytes into the array. The first word we copy will start with
258                 // data we don't care about, but the last `lengthmod` bytes will
259                 // land at the beginning of the contents of the new array. When
260                 // we're done copying, we overwrite the full first word with
261                 // the actual length of the slice.
262                 let lengthmod := and(_length, 31)
263
264                 // The multiplication in the next line is necessary
265                 // because when slicing multiples of 32 bytes (lengthmod == 0)
266                 // the following copy loop was copying the origin's length
267                 // and then ending prematurely not copying everything it should.
268                 let mc := add(add(tempBytes, lengthmod), mul(0x20, iszero(lengthmod)))
269                 let end := add(mc, _length)
270
271                 for {
272                     // The multiplication in the next line has the same exact purpose
273                     // as the one above.
274                     let cc := add(add(add(_bytes, lengthmod), mul(0x20, iszero(lengthmod))),
275                                 _start)
276                 } lt(mc, end) {
277                     mc := add(mc, 0x20)
278                     cc := add(cc, 0x20)
279                 }

```

```

279         mstore(mc, mload(cc))
280     }
281
282     mstore(tempBytes, _length)
283
284     //update free-memory pointer
285     //allocating the array padded to 32 bytes like the compiler does now
286     mstore(0x40, and(add(mc, 31), not(31)))
287 }
288 //if we want a zero-length slice let's just return a zero-length array
289 default {
290     tempBytes := mload(0x40)
291
292     mstore(0x40, add(tempBytes, 0x20))
293 }
294 }
295
296 return tempBytes;
297 }

```

✓ The code meets the specification.

Formal Verification Request 248

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.74 ms

Line 233 in File BytesLib.sol

```

233 //CTK NO_ASF

```

Line 234-297 in File BytesLib.sol

```

234 function slice(
235     bytes memory _bytes,
236     uint _start,
237     uint _length
238 )
239     internal
240     pure
241     returns (bytes memory)
242 {
243     require(_bytes.length >= (_start + _length), "_bytes.length >= (_start + _length)");
244
245     bytes memory tempBytes;
246
247     assembly {
248         switch iszero(_length)
249         case 0 {
250             // Get a location of some free memory and store it in tempBytes as
251             // Solidity does for memory variables.
252             tempBytes := mload(0x40)
253
254             // The first word of the slice result is potentially a partial
255             // word read from the original array. To read it, we calculate
256             // the length of that partial word and start copying that many

```

```

257 // bytes into the array. The first word we copy will start with
258 // data we don't care about, but the last `lengthmod` bytes will
259 // land at the beginning of the contents of the new array. When
260 // we're done copying, we overwrite the full first word with
261 // the actual length of the slice.
262 let lengthmod := and(_length, 31)
263
264 // The multiplication in the next line is necessary
265 // because when slicing multiples of 32 bytes (lengthmod == 0)
266 // the following copy loop was copying the origin's length
267 // and then ending prematurely not copying everything it should.
268 let mc := add(add(tempBytes, lengthmod), mul(0x20, iszero(lengthmod)))
269 let end := add(mc, _length)
270
271 for {
272     // The multiplication in the next line has the same exact purpose
273     // as the one above.
274     let cc := add(add(add(_bytes, lengthmod), mul(0x20, iszero(lengthmod))),
275                     _start)
276 } lt(mc, end) {
277     mc := add(mc, 0x20)
278     cc := add(cc, 0x20)
279 } {
280     mstore(mc, mload(cc))
281 }
282 mstore(tempBytes, _length)
283
284 //update free-memory pointer
285 //allocating the array padded to 32 bytes like the compiler does now
286 mstore(0x40, and(add(mc, 31), not(31)))
287 }
288 //if we want a zero-length slice let's just return a zero-length array
289 default {
290     tempBytes := mload(0x40)
291
292     mstore(0x40, add(tempBytes, 0x20))
293 }
294 }
295
296 return tempBytes;
297 }

```

✓ The code meets the specification.

Formal Verification Request 249

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 20.46 ms

Line 298 in File BytesLib.sol

298 // @CTK NO_BUF_OVERFLOW

Line 300-309 in File BytesLib.sol

```

300     function toAddress(bytes memory _bytes, uint _start) internal pure returns (address) {
301         require(_bytes.length >= (_start + 20), "_bytes.length >= (_start + 20)");
302         address tempAddress;
303
304         assembly {
305             tempAddress := div(mload(add(add(_bytes, 0x20), _start)), 0
                               x100000000000000000000000000000000)
306         }
307
308         return tempAddress;
309     }

```

✓ The code meets the specification.

Formal Verification Request 250

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.59 ms

Line 299 in File BytesLib.sol

```
299     // @CTK NO_ASF
```

Line 300-309 in File BytesLib.sol

```

300     function toAddress(bytes memory _bytes, uint _start) internal pure returns (address) {
301         require(_bytes.length >= (_start + 20), "_bytes.length >= (_start + 20)");
302         address tempAddress;
303
304         assembly {
305             tempAddress := div(mload(add(add(_bytes, 0x20), _start)), 0
                               x100000000000000000000000000000000)
306         }
307
308         return tempAddress;
309     }

```

✓ The code meets the specification.

Formal Verification Request 251

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 21.18 ms

Line 310 in File BytesLib.sol

```
310     // @CTK NO_BUF_OVERFLOW
```

Line 312-321 in File BytesLib.sol

```

312     function toUint8(bytes memory _bytes, uint _start) internal pure returns (uint8) {
313         require(_bytes.length >= (_start + 1), "_bytes.length >= (_start + 1)");
314         uint8 tempUint;

```

```

315
316     assembly {
317         tempUint := mload(add(add(_bytes, 0x1), _start))
318     }
319
320     return tempUint;
321 }

```

✓ The code meets the specification.

Formal Verification Request 252

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.62 ms

Line 311 in File BytesLib.sol

```

311 //©CTK NO_ASF

```

Line 312-321 in File BytesLib.sol

```

312 function toUint8(bytes memory _bytes, uint _start) internal pure returns (uint8) {
313     require(_bytes.length >= (_start + 1), "_bytes.length >= (_start + 1)");
314     uint8 tempUint;
315
316     assembly {
317         tempUint := mload(add(add(_bytes, 0x1), _start))
318     }
319
320     return tempUint;
321 }

```

✓ The code meets the specification.

Formal Verification Request 253

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 21.51 ms

Line 322 in File BytesLib.sol

```

322 //©CTK NO_BUF_OVERFLOW

```

Line 324-333 in File BytesLib.sol

```

324 function toUint16(bytes memory _bytes, uint _start) internal pure returns (uint16) {
325     require(_bytes.length >= (_start + 2), "_bytes.length >= (_start + 2)");
326     uint16 tempUint;
327
328     assembly {
329         tempUint := mload(add(add(_bytes, 0x2), _start))
330     }
331 }

```

```
332     return tempUint;
333 }
```

✓ The code meets the specification.

Formal Verification Request 254

Method will not encounter an assertion failure.



23, Dec 2019



0.53 ms

Line 323 in File BytesLib.sol

```
323 //©CTK NO_ASF
```

Line 324-333 in File BytesLib.sol

```
324 function toUint16(bytes memory _bytes, uint _start) internal pure returns (uint16) {
325     require(_bytes.length >= (_start + 2), "_bytes.length >= (_start + 2)");
326     uint16 tempUint;
327
328     assembly {
329         tempUint := mload(add(add(_bytes, 0x2), _start))
330     }
331
332     return tempUint;
333 }
```

✓ The code meets the specification.

Formal Verification Request 255

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



20.84 ms

Line 334 in File BytesLib.sol

```
334 //©CTK NO_BUF_OVERFLOW
```

Line 336-345 in File BytesLib.sol


```
336 function toUint32(bytes memory _bytes, uint _start) internal pure returns (uint32) {
337     require(_bytes.length >= (_start + 4), "_bytes.length >= (_start + 4)");
338     uint32 tempUint;
339
340     assembly {
341         tempUint := mload(add(add(_bytes, 0x4), _start))
342     }
343
344     return tempUint;
345 }
```

✓ The code meets the specification.

Formal Verification Request 256

Method will not encounter an assertion failure.

 23, Dec 2019

 0.79 ms

Line 335 in File BytesLib.sol

335 `//@CTK NO_ASF`

Line 336-345 in File BytesLib.sol


```
336 function toUint32(bytes memory _bytes, uint _start) internal pure returns (uint32) {
337     require(_bytes.length >= (_start + 4), "_bytes.length >= (_start + 4)");
338     uint32 tempUint;
339
340     assembly {
341         tempUint := mload(add(add(_bytes, 0x4), _start))
342     }
343
344     return tempUint;
345 }
```

 The code meets the specification.

Formal Verification Request 257

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 19.03 ms

Line 346 in File BytesLib.sol

346 `//@CTK NO_BUF_OVERFLOW`

Line 348-357 in File BytesLib.sol


```
348 function toUint64(bytes memory _bytes, uint _start) internal pure returns (uint64) {
349     require(_bytes.length >= (_start + 8), "_bytes.length >= (_start + 8)");
350     uint64 tempUint;
351
352     assembly {
353         tempUint := mload(add(add(_bytes, 0x8), _start))
354     }
355
356     return tempUint;
357 }
```

 The code meets the specification.

Formal Verification Request 258

Method will not encounter an assertion failure.

 23, Dec 2019

 0.55 ms

Line 347 in File BytesLib.sol

```
347 // @CTK NO_ASF
```

Line 348-357 in File BytesLib.sol


```
348 function toUint64(bytes memory _bytes, uint _start) internal pure returns (uint64) {
349     require(_bytes.length >= (_start + 8), "_bytes.length >= (_start + 8)");
350     uint64 tempUint;
351
352     assembly {
353         tempUint := mload(add(add(_bytes, 0x8), _start))
354     }
355
356     return tempUint;
357 }
```

✓ The code meets the specification.

Formal Verification Request 259

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 23.43 ms

Line 358 in File BytesLib.sol

```
358 // @CTK NO_BUF_OVERFLOW
```

Line 360-369 in File BytesLib.sol


```
360 function toUint96(bytes memory _bytes, uint _start) internal pure returns (uint96) {
361     require(_bytes.length >= (_start + 12), "_bytes.length >= (_start + 12)");
362     uint96 tempUint;
363
364     assembly {
365         tempUint := mload(add(add(_bytes, 0xc), _start))
366     }
367
368     return tempUint;
369 }
```

✓ The code meets the specification.

Formal Verification Request 260

Method will not encounter an assertion failure.

 23, Dec 2019

 0.53 ms

Line 359 in File BytesLib.sol

```
359 // @CTK NO_ASF
```

Line 360-369 in File BytesLib.sol

```

360 function toUint96(bytes memory _bytes, uint _start) internal pure returns (uint96) {
361     require(_bytes.length >= (_start + 12), "_bytes.length >= (_start + 12)");
362     uint96 tempUint;
363
364     assembly {
365         tempUint := mload(add(add(_bytes, 0xc), _start))
366     }
367
368     return tempUint;
369 }

```

✓ The code meets the specification.

Formal Verification Request 261

Buffer overflow / array index out of bound would never happen.



23, Dec 2019



21.54 ms

Line 370 in File BytesLib.sol

```

370 // @CTK NO_BUF_OVERFLOW

```

Line 372-381 in File BytesLib.sol

```

372 function toUint128(bytes memory _bytes, uint _start) internal pure returns (uint128) {
373     require(_bytes.length >= (_start + 16), "_bytes.length >= (_start + 16)");
374     uint128 tempUint;
375
376     assembly {
377         tempUint := mload(add(add(_bytes, 0x10), _start))
378     }
379
380     return tempUint;
381 }

```

✓ The code meets the specification.

Formal Verification Request 262

Method will not encounter an assertion failure.



23, Dec 2019



0.54 ms

Line 371 in File BytesLib.sol

```

371 // @CTK NO_ASF

```

Line 372-381 in File BytesLib.sol

```

372 function toUint128(bytes memory _bytes, uint _start) internal pure returns (uint128) {
373     require(_bytes.length >= (_start + 16), "_bytes.length >= (_start + 16)");
374     uint128 tempUint;
375
376     assembly {

```

```

377     tempUint := mload(add(add(_bytes, 0x10), _start))
378 }
379
380     return tempUint;
381 }

```

✓ The code meets the specification.

Formal Verification Request 263

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 22.57 ms

Line 382 in File BytesLib.sol

```

382     //@CTK NO_BUF_OVERFLOW

```

Line 384-393 in File BytesLib.sol

```

384     function toUint(bytes memory _bytes, uint _start) internal pure returns (uint256) {
385         require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
386         uint256 tempUint;
387
388         assembly {
389             tempUint := mload(add(add(_bytes, 0x20), _start))
390         }
391
392         return tempUint;
393     }

```

✓ The code meets the specification.

Formal Verification Request 264

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.7 ms

Line 383 in File BytesLib.sol

```

383     //@CTK NO_ASF

```

Line 384-393 in File BytesLib.sol

```

384     function toUint(bytes memory _bytes, uint _start) internal pure returns (uint256) {
385         require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
386         uint256 tempUint;
387
388         assembly {
389             tempUint := mload(add(add(_bytes, 0x20), _start))
390         }
391
392         return tempUint;
393     }

```

✓ The code meets the specification.

Formal Verification Request 265

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019

🕒 22.1 ms

Line 394 in File BytesLib.sol

394 `//@CTK NO_BUF_OVERFLOW`

Line 396-405 in File BytesLib.sol

```
396   function toBytes32(bytes memory _bytes, uint _start) internal pure returns (bytes32) {
397       require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
398       bytes32 tempBytes32;
399
400       assembly {
401           tempBytes32 := mload(add(add(_bytes, 0x20), _start))
402       }
403
404       return tempBytes32;
405   }
```

✓ The code meets the specification.

Formal Verification Request 266

Method will not encounter an assertion failure.

📅 23, Dec 2019

🕒 0.59 ms

Line 395 in File BytesLib.sol

395 `//@CTK NO_ASF`

Line 396-405 in File BytesLib.sol


```
396   function toBytes32(bytes memory _bytes, uint _start) internal pure returns (bytes32) {
397       require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
398       bytes32 tempBytes32;
399
400       assembly {
401           tempBytes32 := mload(add(add(_bytes, 0x20), _start))
402       }
403
404       return tempBytes32;
405   }
```

✓ The code meets the specification.

Formal Verification Request 267

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 7.55 ms

Line 406 in File BytesLib.sol

406 `//@CTK NO_BUF_OVERFLOW`

Line 409-450 in File BytesLib.sol


```
409 function equal(bytes memory _preBytes, bytes memory _postBytes) internal pure returns (  
410     bool) {  
411     bool success = true;  
412     assembly {  
413         let length := mload(_preBytes)  
414   
415         // if lengths don't match the arrays are not equal  
416         switch eq(length, mload(_postBytes))  
417         case 1 {  
418             // cb is a circuit breaker in the for loop since there's  
419             // no said feature for inline assembly loops  
420             // cb = 1 - don't breaker  
421             // cb = 0 - break  
422             let cb := 1  
423   
424             let mc := add(_preBytes, 0x20)  
425             let end := add(mc, length)  
426   
427             for {  
428                 let cc := add(_postBytes, 0x20)  
429                 // the next line is the loop condition:  
430                 // while(uint(mc < end) + cb == 2)  
431             } eq(add(lt(mc, end), cb), 2) {  
432                 mc := add(mc, 0x20)  
433                 cc := add(cc, 0x20)  
434             } {  
435                 // if any of these checks fails then arrays are not equal  
436                 if iszero(eq(mload(mc), mload(cc))) {  
437                     // unsuccessful:  
438                     success := 0  
439                     cb := 0  
440                 }  
441             }  
442         }  
443         default {  
444             // unsuccessful:  
445             success := 0  
446         }  
447     }  
448   
449     return success;  
450 }
```

 The code meets the specification.

Formal Verification Request 268

If method completes, integer overflow would not happen.

 23, Dec 2019

 0.43 ms

Line 407 in File BytesLib.sol

407 `//@CTK NO_OVERFLOW`

Line 409-450 in File BytesLib.sol


```
409     function equal(bytes memory _preBytes, bytes memory _postBytes) internal pure returns (
410         bool) {
411         bool success = true;
412
413         assembly {
414             let length := mload(_preBytes)
415
416             // if lengths don't match the arrays are not equal
417             switch eq(length, mload(_postBytes))
418             case 1 {
419                 // cb is a circuit breaker in the for loop since there's
420                 // no said feature for inline assembly loops
421                 // cb = 1 - don't breaker
422                 // cb = 0 - break
423                 let cb := 1
424
425                 let mc := add(_preBytes, 0x20)
426                 let end := add(mc, length)
427
428                 for {
429                     let cc := add(_postBytes, 0x20)
430                     // the next line is the loop condition:
431                     // while(uint(mc < end) + cb == 2)
432                     } eq(add(lt(mc, end), cb), 2) {
433                         mc := add(mc, 0x20)
434                         cc := add(cc, 0x20)
435                     } {
436                         // if any of these checks fails then arrays are not equal
437                         if iszero(eq(mload(mc), mload(cc))) {
438                             // unsuccessful:
439                             success := 0
440                             cb := 0
441                         }
442                     }
443                 default {
444                     // unsuccessful:
445                     success := 0
446                 }
447             }
448
449             return success;
450         }
```

 The code meets the specification.

Formal Verification Request 269

Method will not encounter an assertion failure.

 23, Dec 2019

 0.45 ms

Line 408 in File BytesLib.sol

408 `//@CTK NO_ASF`

Line 409-450 in File BytesLib.sol


```
409     function equal(bytes memory _preBytes, bytes memory _postBytes) internal pure returns (
410         bool) {
411         bool success = true;
412
413         assembly {
414             let length := mload(_preBytes)
415
416             // if lengths don't match the arrays are not equal
417             switch eq(length, mload(_postBytes))
418             case 1 {
419                 // cb is a circuit breaker in the for loop since there's
420                 // no said feature for inline assembly loops
421                 // cb = 1 - don't breaker
422                 // cb = 0 - break
423                 let cb := 1
424
425                 let mc := add(_preBytes, 0x20)
426                 let end := add(mc, length)
427
428                 for {
429                     let cc := add(_postBytes, 0x20)
430                     // the next line is the loop condition:
431                     // while(uint(mc < end) + cb == 2)
432                     } eq(add(lt(mc, end), cb), 2) {
433                         mc := add(mc, 0x20)
434                         cc := add(cc, 0x20)
435                     } {
436                         // if any of these checks fails then arrays are not equal
437                         if iszero(eq(mload(mc), mload(cc))) {
438                             // unsuccessful:
439                             success := 0
440                             cb := 0
441                         }
442                     }
443                 default {
444                     // unsuccessful:
445                     success := 0
446                 }
447             }
448
449             return success;
450         }
```

 The code meets the specification.

Formal Verification Request 270

Buffer overflow / array index out of bound would never happen.

 23, Dec 2019

 6.82 ms

Line 451 in File BytesLib.sol

451 `//@CTK NO_BUF_OVERFLOW`

Line 454-524 in File BytesLib.sol

```

454     function equalStorage(
455         bytes storage _preBytes,
456         bytes memory _postBytes
457     )
458     internal
459     view
460     returns (bool)
461     {
462         bool success = true;
463
464         assembly {
465             // we know _preBytes_offset is 0
466             let fslot := sload(_preBytes_slot)
467             // Decode the length of the stored array like in concatStorage().
468             let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
469             let mlength := mload(_postBytes)
470
471             // if lengths don't match the arrays are not equal
472             switch eq(slength, mlength)
473             case 1 {
474                 // slength can contain both the length and contents of the array
475                 // if length < 32 bytes so let's prepare for that
476                 // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-
477                     state-variables-in-storage
478                 if iszero(iszero(slength)) {
479                     switch lt(slength, 32)
480                     case 1 {
481                         // blank the last byte which is the length
482                         fslot := mul(div(fslot, 0x100), 0x100)
483
484                         if iszero(eq(fslot, mload(add(_postBytes, 0x20)))) {
485                             // unsuccessful:
486                             success := 0
487                         }
488                     }
489                     default {
490                         // cb is a circuit breaker in the for loop since there's
491                         // no said feature for inline assembly loops
492                         // cb = 1 - don't breaker
493                         // cb = 0 - break
494                         let cb := 1
495
496                         // get the keccak hash to get the contents of the array
497                         mstore(0x0, _preBytes_slot)
498                         let sc := keccak256(0x0, 0x20)

```



```

499         let mc := add(_postBytes, 0x20)
500         let end := add(mc, mlength)
501
502         // the next line is the loop condition:
503         // while(uint(mc < end) + cb == 2)
504         for {} eq(add(lt(mc, end), cb), 2) {
505             sc := add(sc, 1)
506             mc := add(mc, 0x20)
507         } {
508             if iszero(eq(sload(sc), mload(mc))) {
509                 // unsuccessful:
510                 success := 0
511                 cb := 0
512             }
513         }
514     }
515 }
516 }
517 default {
518     // unsuccessful:
519     success := 0
520 }
521 }
522
523 return success;
524 }

```

✓ The code meets the specification.

Formal Verification Request 271

If method completes, integer overflow would not happen.

📅 23, Dec 2019

🕒 0.48 ms

Line 452 in File BytesLib.sol

```

452     // @CTK NO_OVERFLOW

```

Line 454-524 in File BytesLib.sol

```

454     function equalStorage(
455         bytes storage _preBytes,
456         bytes memory _postBytes
457     )
458     internal
459     view
460     returns (bool)
461     {
462         bool success = true;
463
464         assembly {
465             // we know _preBytes_offset is 0
466             let fslot := sload(_preBytes_slot)
467             // Decode the length of the stored array like in concatStorage().
468             let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
469             let mlength := mload(_postBytes)

```

```

470
471 // if lengths don't match the arrays are not equal
472 switch eq(slength, mlength)
473 case 1 {
474     // slength can contain both the length and contents of the array
475     // if length < 32 bytes so let's prepare for that
476     // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-
        state-variables-in-storage
477     if iszero(iszero(slength)) {
478         switch lt(slength, 32)
479         case 1 {
480             // blank the last byte which is the length
481             fslot := mul(div(fslot, 0x100), 0x100)
482
483             if iszero(eq(fslot, mload(add(_postBytes, 0x20)))) {
484                 // unsuccessful:
485                 success := 0
486             }
487         }
488         default {
489             // cb is a circuit breaker in the for loop since there's
490             // no said feature for inline assembly loops
491             // cb = 1 - don't breaker
492             // cb = 0 - break
493             let cb := 1
494
495             // get the keccak hash to get the contents of the array
496             mstore(0x0, _preBytes_slot)
497             let sc := keccak256(0x0, 0x20)
498
499             let mc := add(_postBytes, 0x20)
500             let end := add(mc, mlength)
501
502             // the next line is the loop condition:
503             // while(uint(mc < end) + cb == 2)
504             for {} eq(add(lt(mc, end), cb), 2) {
505                 sc := add(sc, 1)
506                 mc := add(mc, 0x20)
507             } {
508                 if iszero(eq(sload(sc), mload(mc))) {
509                     // unsuccessful:
510                     success := 0
511                     cb := 0
512                 }
513             }
514         }
515     }
516 }
517 default {
518     // unsuccessful:
519     success := 0
520 }
521 }
522
523 return success;
524


```

✓ The code meets the specification.

Formal Verification Request 272

Method will not encounter an assertion failure.

 23, Dec 2019

 0.4 ms

Line 453 in File BytesLib.sol

453 `//@CTK NO_ASF`

Line 454-524 in File BytesLib.sol

```

454     function equalStorage(
455         bytes storage _preBytes,
456         bytes memory _postBytes
457     )
458     internal
459     view
460     returns (bool)
461     {
462         bool success = true;
463
464         assembly {
465             // we know _preBytes_offset is 0
466             let fslot := sload(_preBytes_slot)
467             // Decode the length of the stored array like in concatStorage().
468             let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
469             let mlength := mload(_postBytes)
470
471             // if lengths don't match the arrays are not equal
472             switch eq(slength, mlength)
473             case 1 {
474                 // slength can contain both the length and contents of the array
475                 // if length < 32 bytes so let's prepare for that
476                 // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-
477                     state-variables-in-storage
478                 if iszero(iszero(slength)) {
479                     switch lt(slength, 32)
480                     case 1 {
481                         // blank the last byte which is the length
482                         fslot := mul(div(fslot, 0x100), 0x100)
483
484                         if iszero(eq(fslot, mload(add(_postBytes, 0x20)))) {
485                             // unsuccessful:
486                             success := 0
487                         }
488                     }
489                     default {
490                         // cb is a circuit breaker in the for loop since there's
491                         // no said feature for inline assembly loops
492                         // cb = 1 - don't breaker
493                         // cb = 0 - break
494                         let cb := 1
495
496                         // get the keccak hash to get the contents of the array
497                         mstore(0x0, _preBytes_slot)
498                         let sc := keccak256(0x0, 0x20)

```

```

499         let mc := add(_postBytes, 0x20)
500         let end := add(mc, mlength)
501
502         // the next line is the loop condition:
503         // while(uint(mc < end) + cb == 2)
504         for {} eq(add(lt(mc, end), cb), 2) {
505             sc := add(sc, 1)
506             mc := add(mc, 0x20)
507         } {
508             if iszero(eq(sload(sc), mload(mc))) {
509                 // unsuccessful:
510                 success := 0
511                 cb := 0
512             }
513         }
514     }
515 }
516 }
517 default {
518     // unsuccessful:
519     success := 0
520 }
521 }
522
523 return success;
524 }

```

✓ The code meets the specification.

Formal Verification Request 273

SafeMath add



23, Dec 2019



20.68 ms

Line 26-32 in File SafeMath.sol

```

26  /*@CTK "SafeMath add"
27  @post (a + b < a || a + b < b) == __reverted
28  @post !__reverted -> __return == a + b
29  @post !__reverted -> !__has_overflow
30  @post !__reverted -> !__has_assertion_failure
31  @post !(__has_buf_overflow)
32  */

```

Line 33-38 in File SafeMath.sol

```

33  function add(uint256 a, uint256 b) internal pure returns (uint256) {
34      uint256 c = a + b;
35      require(c >= a, "SafeMath: addition overflow");
36
37      return c;
38  }

```

✓ The code meets the specification.

Formal Verification Request 274

SafeMath sub



23, Dec 2019



15.96 ms

Line 64-70 in File SafeMath.sol

```

64      /*@CTK "SafeMath sub"
65      @post (a < b) == __reverted
66      @post !__reverted -> __return == a - b
67      @post !__reverted -> !__has_overflow
68      @post !__reverted -> !__has_assertion_failure
69      @post !(__has_buf_overflow)
70      */

```

Line 71-76 in File SafeMath.sol

```

71      function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
72          uint256) {
73          require(b <= a, errorMessage);
74          uint256 c = a - b;
75          return c;
76      }

```

✓ The code meets the specification.

Formal Verification Request 275

SafeMath mul



23, Dec 2019



221.26 ms

Line 87-93 in File SafeMath.sol

```

87      /*@CTK "SafeMath mul"
88      @post (((a) > (0)) && (((a) * (b)) / (a)) != (b))) == (__reverted)
89      @post !__reverted -> __return == a * b
90      @post !__reverted == !__has_overflow
91      @post !__reverted -> !__has_assertion_failure
92      @post !(__has_buf_overflow)
93      */

```

Line 94-106 in File SafeMath.sol

```

94      function mul(uint256 a, uint256 b) internal pure returns (uint256) {
95          // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
96          // benefit is lost if 'b' is also tested.
97          // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
98          if (a == 0) {
99              return 0;
100          }
101
102          uint256 c = a * b;
103          require(c / a == b, "SafeMath: multiplication overflow");
104      }

```

```
105     return c;
106 }
```

✓ The code meets the specification.

Formal Verification Request 276

SafeMath div



23, Dec 2019



16.85 ms

Line 136-142 in File SafeMath.sol

```
136  /*@CTK "SafeMath div"
137  @post (b <= 0) == __reverted
138  @post !__reverted -> __return == a / b
139  @post !__reverted -> !__has_overflow
140  @post !__reverted -> !__has_assertion_failure
141  @post !(__has_buf_overflow)
142  */
```

Line 143-150 in File SafeMath.sol

```
143  function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
144      uint256) {
145      // Solidity only automatically asserts when dividing by 0
146      require(b > 0, errorMessage);
147      uint256 c = a / b;
148      // assert(a == b * c + a % b); // There is no case in which this doesn't hold
149
150      return c;
151  }
```

✓ The code meets the specification.

Formal Verification Request 277

SafeMath mod



23, Dec 2019



14.05 ms

Line 180-186 in File SafeMath.sol

```
180  /*@CTK "SafeMath mod"
181  @post (b == 0) == __reverted
182  @post !__reverted -> __return == a % b
183  @post !__reverted -> !__has_overflow
184  @post !__reverted -> !__has_assertion_failure
185  @post !(__has_buf_overflow)
186  */
```

Line 187-190 in File SafeMath.sol

```
187  function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
      uint256) {
188      require(b != 0, errorMessage);
189      return a % b;
190  }
```

✓ The code meets the specification.

Source Code with CertiK Labels

File MineralNFTMarket.sol

```

1  pragma solidity ^0.5.13;
2
3  import "../GSN/Context.sol";
4  import "../token/MineralNFT.sol";
5  import "../token/Mineral.sol";
6  import "../token/ERC/SafeERC20.sol";
7  import "../token/ERC/IERC721Receiver.sol";
8  import "../token/ERC/IERC20Receiver.sol";
9  import "../utils/Ownable.sol";
10 import "../utils/BytesLib.sol";
11 import "../math/SafeMath.sol";
12
13 contract MineralNFTMarket is Context, IERC721Receiver, IERC20Receiver, Ownable {
14     using SafeMath for uint256;
15     using BytesLib for bytes;
16     using SafeERC20 for IERC20;
17
18     enum ItemStatus { enable, sold, canceled }
19
20     struct Item {
21         uint256 id;
22         uint256 price;
23         address owner;
24         uint8 status; // 0 : enable, 1 : sold, 2 : cancel
25     }
26
27     bytes4 private constant _ERC721_RECEIVED = 0x150b7a02;
28
29     event SellItem(address owner, uint256 id, uint256 price);
30     event BuyItem(address seller, address buyer, uint256 id, uint256 price);
31     event CancelItem(address owner, uint256 id);
32     event TakeMineral(address owner, uint256 mineral, uint256[] ids);
33
34     mapping(uint256 => Item) private _items;
35     mapping(address => uint256[]) private _soldTokenIds;
36     mapping(address => uint256) private _takeableMineral;
37
38     MineralNFT public _nft;
39     IERC20 public _mineral;
40
41     //@CTK NO_BUF_OVERFLOW
42     //@CTK NO_OVERFLOW
43     //@CTK NO_ASF
44     constructor(address nft, address mineral) public {
45         setMineralNFTTokenContract(nft);
46         setMineralTokenContract(mineral);
47     }
48     //@CTK NO_BUF_OVERFLOW
49     //@CTK NO_OVERFLOW
50     //@CTK NO_ASF
51     /*CTK exists
52     @post _items[id].price == 0 -> __return == false
53     @post _items[id].price != 0 -> __return == (_items[id].status == 0)
54     */

```



```

55 function exists(uint id) external view returns (bool) {
56     return _exists(id);
57 }
58 //@CTK NO_BUF_OVERFLOW
59 //@CTK NO_OVERFLOW
60 //@CTK NO_ASF
61 /*@CTK _exists
62     @post _items[id].price == 0 -> __return == false
63     @post _items[id].price != 0 -> __return == (_items[id].status == 0)
64 */
65 function _exists(uint id) internal view returns (bool) {
66     if (_items[id].price == 0)
67         return false;
68
69     return _items[id].status == uint8(ItemStatus.enable);
70 }
71 //@CTK NO_BUF_OVERFLOW
72 //@CTK NO_OVERFLOW
73 //@CTK NO_ASF
74 /*@CTK getTakeableMineral
75     @post !__reverted -> __return == _takeableMineral[msg.sender]
76 */
77 function getTakeableMineral() external view returns (uint256) {
78     return _takeableMineral[_msgSender()];
79 }
80 //@CTK NO_BUF_OVERFLOW
81 //@CTK NO_OVERFLOW
82 //@CTK NO_ASF
83 /*#CTK getItemInfo
84     @post !__reverted -> __return == (_items[tokenId].price, _items[tokenId].owner, _items
85         [tokenId].status)
86 */
87 function getItemInfo(uint256 tokenId) external view returns (uint256 price, address
88     owner, uint8 status) {
89     return (_items[tokenId].price, _items[tokenId].owner, _items[tokenId].status);
90 }
91 // sel
92 //@CTK NO_BUF_OVERFLOW
93 //@CTK NO_OVERFLOW
94 //@CTK NO_ASF
95 /*@CTK onERC721Received
96     @tag assume_completion
97     @pre msg.sender == address(_nft)
98     @pre _items[tokenId].price == 0 || (_items[tokenId].price != 0 && _items[tokenId].
99         status != 0)
100     // @pre 0 < price
101     @post __return == 0x150b7a02
102 */
103 function onERC721Received(address operator, address from, uint256 tokenId, bytes
104     calldata data) external returns (bytes4) {
105     require (_msgSender() == address(_nft), "msg.sender is not nft token address");
106     require (_exists(tokenId) == false, "item with input tokenId is existing");
107     uint256 price = data.toUint(0);
108     require (0 < price, "input price is not valid");
109     _items[tokenId] = Item({
110         id: tokenId,
111         price: price,

```

```

109         owner: operator,
110         status: uint8(ItemStatus.enable)
111     });
112     emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
113     return _ERC721_RECEIVED;
114 }
115
116 // buy
117 //@CTK NO_BUF_OVERFLOW
118 //@CTK NO_OVERFLOW
119 //@CTK NO_ASF
120 /*@CTK onERC20Received
121     @pre msg.sender == address(_mineral)
122     @pre msg.sender == address(_nft)
123     // @pre _items[id].price == 0 || (_items[id].price != 0 && _items[id].status != 0)
124     // @pre _items[id].price == amount
125     // @pre from != _items[id].owner
126     // @pre _items[id].status == 0
127     // @post __post._takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner]
128     // @post __post._items[id].status == 1
129 */
130 function onERC20Received(address from, uint256 amount, bytes memory data) public returns
    (bool) {
131     require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
    ");
132
133     uint256 id = data.toUint(0);
134     require (_exists(id), "item with input tokenId is existing");
135
136     Item storage item = _items[id];
137
138     require (item.price == amount, "input amount is not valid");
139     require (from != item.owner, "input buyer is not valid");
140     require (item.status == 0, "item is not available");
141     //ctk start
142     _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
143     //ctk end
144     _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
145     _soldTokenIds[item.owner].push(id);
146     item.status = uint8(ItemStatus.sold);
147     _nft.safeTransferFrom(address(this), from, id);
148
149     emit BuyItem(item.owner, from, id, amount);
150     return true;
151 }
152
153 //@CTK NO_BUF_OVERFLOW
154 //@CTK NO_OVERFLOW
155 //@CTK NO_ASF
156 /*@CTK "cancelItem"
157     @pre _items[tokenId].price != 0 && _items[tokenId].price == 0 || _items[tokenId].
158     status == 0
159     @pre msg.sender == _items[tokenId].owner
160     @pre _items[tokenId].status != 2
161 */
162 function cancelItem(uint256 tokenId) external {
    require (_exists(tokenId), "item with input tokenId is existing");

```

```

163
164     Item item = _items[tokenId];
165
166     require (_msgSender() == item.owner, "msg.sender is not token owner");
167     require (item.status != uint8(ItemStatus.canceled), "item is already canceled");
168
169     item.status = uint8(ItemStatus.canceled);
170     _nft.safeTransferFrom(address(this), _msgSender(), tokenId);
171
172     emit CancelItem(_msgSender(), tokenId);
173 }
174
175 //@CTK NO_BUF_OVERFLOW
176 //@CTK NO_OVERFLOW
177 //@CTK NO_ASF
178 /*@CTK takeMineral
179   @pre 0 < _takeableMineral[msg.sender]
180   @pre _soldTokenIds[msg.sender].length > 0
181   @post __post._takeableMineral[msg.sender] == 0
182   @post __post._soldTokenIds[msg.sender].length == 0
183 */
184 function takeMineral() external {
185     require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
186         take");
187     _takeMineral(_msgSender());
188 }
189
190 //@CTK NO_BUF_OVERFLOW
191 //@CTK NO_OVERFLOW
192 //@CTK NO_ASF
193 /*@CTK _takeMineral
194   @pre _soldTokenIds[addr].length > 0
195   @post __post._takeableMineral[addr] == 0
196   @post __post._soldTokenIds[addr].length == 0
197 */
198 function _takeMineral(address addr) internal {
199     require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");
200
201     uint256 amount = _takeableMineral[addr];
202     uint256[] memory tokenIds = _soldTokenIds[addr];
203     _takeableMineral[addr] = 0;
204     _soldTokenIds[addr].length = 0;
205
206     _mineral.safeTransfer(addr, amount);
207     emit TakeMineral(addr, amount, tokenIds);
208 }
209
210 //@CTK NO_BUF_OVERFLOW
211 //@CTK NO_OVERFLOW
212 //@CTK NO_ASF
213 /*@CTK getSoldTokenIds
214   @post !__reverted -> __return == _soldTokenIds[addr]
215 */
216 function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
217     return _soldTokenIds[addr];
218 }
219
220 //@CTK NO_BUF_OVERFLOW
221 //@CTK NO_OVERFLOW

```

```

220 // @CTK_NO_ASF
221 function setMineralNFTTokenContract(address addr) public onlyOwner {
222     _nft = MineralNFT(addr);
223 }
224 // @CTK_NO_BUF_OVERFLOW
225 // @CTK_NO_OVERFLOW
226 // @CTK_NO_ASF
227 function setMineralTokenContract(address addr) public onlyOwner {
228     _mineral = IERC20(addr);
229 }
230
231 // reset contract
232 // @CTK_NO_BUF_OVERFLOW
233 // @CTK_NO_OVERFLOW
234 // @CTK_NO_ASF
235 /* @CTK_getTakeableMineral
236     @pre msg.sender == _owner
237     @post !__reverted -> __return == _takeableMineral[addr]
238 */
239 function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {
240     return _takeableMineral[addr];
241 }
242
243 // @CTK_NO_BUF_OVERFLOW
244 // @CTK_NO_OVERFLOW
245 // @CTK_NO_ASF
246 /* @CTK_takeMineralOwnerable
247     @pre msg.sender == _owner
248     @pre _soldTokenIds[addr].length > 0
249     @post __post._takeableMineral[addr] == 0
250     @post __post._soldTokenIds[addr].length == 0
251 */
252 function takeMineralOwnerable(address addr) external onlyOwner {
253     _takeMineral(addr);
254 }
255 }

```

File token/Mineral.sol

```

1 pragma solidity ^0.5.13;
2
3 import "./ERC/ERC20Burnable.sol";
4 import "./ERC/ERC1132.sol";
5
6 contract Mineral is ERC1132, ERC20Burnable {
7     string internal constant ALREADY_LOCKED = 'Tokens already locked';
8     string internal constant NOT_LOCKED = 'No tokens locked';
9     string internal constant AMOUNT_ZERO = 'Amount can not be 0';
10
11     string public name = "Mineral";
12     string public symbol = "MNR";
13     uint public decimals = 6;
14     uint public INITIAL_SUPPLY = (10 ** 10) * (10 ** decimals);
15
16     constructor() public {
17         _mint(_msgSender(), INITIAL_SUPPLY);
18     }
19
20     /**

```

```

21  * @dev Locks a specified amount of tokens against an address,
22  *      for a specified reason and time
23  * @param _reason The reason to lock tokens
24  * @param _amount Number of tokens to be locked
25  * @param _time Lock time in seconds
26  */
27  //@CTK NO_BUF_OVERFLOW
28  //@CTK NO_ASF
29  /*#CTK lock
30  @tag assume_completion
31  @pre _amount != 0
32  @pre locked[msg.sender][_reason].claimed || (!locked[msg.sender][_reason].claimed &&
    locked[msg.sender][_reason].amount == 0)
33  @post locked[msg.sender][_reason].amount == 0 -> __post.lockReason[msg.sender].length
    == lockReason[msg.sender].length + 1
34  */
35  function lock(bytes32 _reason, uint256 _amount, uint256 _time)
36  public
37  returns (bool)
38  {
39      uint256 validUntil = now.add(_time); //solhint-disable-line
40
41      // If tokens are already locked, then functions extendLock or
42      // increaseLockAmount should be used to make any changes
43      require(tokensLocked(_msgSender(), _reason) == 0, ALREADY_LOCKED);
44      require(_amount != 0, AMOUNT_ZERO);
45
46      if (locked[_msgSender()][_reason].amount == 0)
47          lockReason[_msgSender()].push(_reason);
48
49      transfer(address(this), _amount);
50
51      locked[_msgSender()][_reason] = lockToken(_amount, validUntil, false);
52
53      emit Locked(_msgSender(), _reason, _amount, validUntil);
54      return true;
55  }
56
57  /**
58  * @dev Transfers and Locks a specified amount of tokens,
59  *      for a specified reason and time
60  * @param _to adress to which tokens are to be transfered
61  * @param _reason The reason to lock tokens
62  * @param _amount Number of tokens to be transfered and locked
63  * @param _time Lock time in seconds
64  */
65  //@CTK NO_BUF_OVERFLOW
66  //@CTK NO_ASF
67  /*#CTK transferWithLock
68  @tag assume_completion
69  @pre locked[_to][_reason].claimed || (!locked[_to][_reason].claimed && locked[_to][
    _reason].amount == 0)
70  @pre _amount != 0
71  @post locked[_to][_reason].amount == 0 -> __post.lockReason[_to].length == lockReason[
    _to].length + 1
72  */
73  function transferWithLock(address _to, bytes32 _reason, uint256 _amount, uint256 _time)
74  external

```

```

75     returns (bool)
76 {
77     uint256 validUntil = now.add(_time); //solhint-disable-line
78
79     require(tokensLocked(_to, _reason) == 0, ALREADY_LOCKED);
80     require(_amount != 0, AMOUNT_ZERO);
81
82     if (locked[_to][_reason].amount == 0)
83         lockReason[_to].push(_reason);
84
85     transfer(address(this), _amount);
86
87     locked[_to][_reason] = lockToken(_amount, validUntil, false);
88
89     emit Locked(_to, _reason, _amount, validUntil);
90     return true;
91 }
92
93 /**
94  * @dev Returns tokens locked for a specified address for a
95  *       specified reason
96  *
97  * @param _of The address whose tokens are locked
98  * @param _reason The reason to query the lock tokens for
99  */
100 //@CTK NO_BUF_OVERFLOW
101 //@CTK NO_OVERFLOW
102 //@CTK NO_ASF
103 /*#CTK tokensLocked
104  @tag assume_completion
105  @post !locked[_of][_reason].claimed -> __return == locked[_of][_reason].amount
106 */
107 function tokensLocked(address _of, bytes32 _reason)
108     public
109     view
110     returns (uint256 amount)
111 {
112     if (!locked[_of][_reason].claimed)
113         amount = locked[_of][_reason].amount;
114 }
115
116 /**
117  * @dev Returns tokens locked for a specified address for a
118  *       specified reason at a specific time
119  *
120  * @param _of The address whose tokens are locked
121  * @param _reason The reason to query the lock tokens for
122  * @param _time The timestamp to query the lock tokens for
123  */
124 //@CTK NO_BUF_OVERFLOW
125 //@CTK NO_OVERFLOW
126 //@CTK NO_ASF
127 /*#CTK tokensLockedAtTime
128  @tag assume_completion
129  @post locked[_of][_reason].validity > _time -> __return == locked[_of][_reason].amount
130 */
131 function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
132     public

```

```

133     view
134     returns (uint256 amount)
135     {
136         if (locked[_of][_reason].validity > _time)
137             amount = locked[_of][_reason].amount;
138     }
139
140     /**
141     * @dev Returns total tokens held by an address (locked + transferable)
142     * @param _of The address to query the total balance of
143     */
144     //@CTK NO_BUF_OVERFLOW
145     //@CTK NO_OVERFLOW
146     //@CTK NO_ASF
147     /*#CTK totalBalanceOf
148     @post !__reverted -> __return == balances[_of]
149     */
150     function totalBalanceOf(address _of)
151     public
152     view
153     returns (uint256 amount)
154     {
155         amount = balanceOf(_of);
156         for (uint256 i = 0; i < lockReason[_of].length; i++) {
157             amount = amount.add(tokensLocked(_of, lockReason[_of][i]));
158         }
159     }
160
161     /**
162     * @dev Extends lock for a specified reason and time
163     * @param _reason The reason to lock tokens
164     * @param _time Lock extension time in seconds
165     */
166     //@CTK NO_BUF_OVERFLOW
167     //@CTK NO_OVERFLOW
168     //@CTK NO_ASF
169     /*CTK extendLock
170     @tag assume_completion
171     @pre (!locked[msg.sender][_reason].claimed && locked[msg.sender][_reason].amount > 0)
172     @post __post.locked[msg.sender][_reason].validity == locked[msg.sender][_reason].
173           validity + _time
174     */
175     function extendLock(bytes32 _reason, uint256 _time)
176     public
177     returns (bool)
178     {
179         require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
180
181         locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
182             _time);
183
184         emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
185             _msgSender()][_reason].validity);
186         return true;
187     }
188
189     /**
190     * @dev Increase number of tokens locked for a specified reason

```

```

188  * @param _reason The reason to lock tokens
189  * @param _amount Number of tokens to be increased
190  */
191  //@CTK NO_BUF_OVERFLOW
192  //@CTK NO_OVERFLOW
193  //@CTK NO_ASF
194  /*@CTK increaseLockAmount
195   @tag assume_completion
196   @pre (!locked[msg.sender][_reason].claimed && locked[msg.sender][_reason].amount > 0)
197   @post __post.locked[msg.sender][_reason].amount == locked[msg.sender][_reason].amount
        + _amount
198  */
199  function increaseLockAmount(bytes32 _reason, uint256 _amount)
200      public
201      returns (bool)
202  {
203      require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
204      transfer(address(this), _amount);
205
206      locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
        _amount);
207
208      emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
        _msgSender()][_reason].validity);
209      return true;
210  }
211
212  /**
213   * @dev Returns unlockable tokens for a specified address for a specified reason
214   * @param _of The address to query the the unlockable token count of
215   * @param _reason The reason to query the unlockable tokens for
216   */
217  //@CTK NO_BUF_OVERFLOW
218  //@CTK NO_OVERFLOW
219  //@CTK NO_ASF
220  /*#CTK tokensUnlockable
221   @post (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) ->
        __return == locked[_of][_reason].amount
222  */
223  function tokensUnlockable(address _of, bytes32 _reason)
224      public
225      view
226      returns (uint256 amount)
227  {
228      if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-
        disable-line
229          amount = locked[_of][_reason].amount;
230  }
231
232  /**
233   * @dev Unlocks the unlockable tokens of a specified address
234   * @param _of Address of user, claiming back unlockable tokens
235   */
236  //@CTK NO_BUF_OVERFLOW
237  //@CTK NO_OVERFLOW
238  //@CTK NO_ASF
239  function unlockAll(address _of)
240      public

```



```

241     returns (uint256 unlockableTokens)
242 {
243     uint256 lockedTokens;
244
245     /*#CTK "loop_unlockAll"
246     @inv i <= lockReason[_of].length
247     @inv forall j: uint. (j >= 0 /\ j < i /\ (locked[_of][lockReason[_of][i]].validity
248         <= now && !locked[_of][lockReason[_of][i]].claimed && locked[_of][lockReason[
249         _of][i]].amount > 0)) -> locked[_of][lockReason[_of][i]].claimed
250     @post i == lockReason[_of].length
251     @post !__should_return
252     */
253     /*@CTK loop
254     @inv true
255     */
256     for (uint256 i = 0; i < lockReason[_of].length; i++) {
257         lockedTokens = tokensUnlockable(_of, lockReason[_of][i]);
258         if (lockedTokens > 0) {
259             unlockableTokens = unlockableTokens.add(lockedTokens);
260             locked[_of][lockReason[_of][i]].claimed = true;
261             emit Unlocked(_of, lockReason[_of][i], lockedTokens);
262         }
263     }
264
265     if (unlockableTokens > 0)
266         this.transfer(_of, unlockableTokens);
267 }
268
269 /**
270  * @dev Unlock once
271  * @param _of Address of user, claiming back unlockable tokens
272  * @param _reason Once reason
273  */
274 // @CTK NO_BUF_OVERFLOW
275 // @CTK NO_OVERFLOW
276 // @CTK NO_ASF
277 /*#CTK unlock
278 @post locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed && locked[
279     _of][_reason].amount > 0 -> __post.locked[_of][_reason].claimed
280 */
281 function unlock(address _of, bytes32 _reason)
282     public
283     returns (uint256 unlocked)
284 {
285     unlocked = tokensUnlockable(_of, _reason);
286     if (unlocked > 0) {
287         locked[_of][_reason].claimed = true;
288         emit Unlocked(_of, _reason, unlocked);
289         this.transfer(_of, unlocked);
290     }
291 }
292
293 /**
294  * @dev Gets the unlockable tokens of a specified address
295  * @param _of The address to query the the unlockable token count of
296  */
297 // @CTK NO_BUF_OVERFLOW
298 // @CTK NO_OVERFLOW

```

```

296 // @CTK NO_ASF
297 function getUnlockableTokens(address _of)
298     public
299     view
300     returns (uint256 unlockableTokens)
301 {
302     for (uint256 i = 0; i < lockReason[_of].length; i++) {
303         unlockableTokens = unlockableTokens.add(tokensUnlockable(_of, lockReason[_of][i])
304         );
305     }
306 }
307 // @CTK NO_BUF_OVERFLOW
308 // @CTK NO_ASF
309 function getLockReasons(address _of, uint256 _start, uint256 _end)
310     external
311     view
312     returns (bytes32[] memory reasons)
313 {
314     uint256 length = _end - _start;
315     reasons = new bytes32[](length);
316     /* @CTK loop_getLockReasons
317        @inv i <= length
318        @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
319        j]
320        @post i == length
321        @post !__should_return
322    */
323     for (uint256 i = 0; i < length; i++) {
324         reasons[i] = lockReason[_of][_start + i];
325     }
326     return reasons;
327 }
328 // @CTK NO_BUF_OVERFLOW
329 // @CTK NO_OVERFLOW
330 // @CTK NO_ASF
331 function getLockReasonLength(address _of)
332     external
333     view
334     returns (uint256 length)
335 {
336     return lockReason[_of].length;
337 }
338 // @CTK NO_BUF_OVERFLOW
339 // @CTK NO_OVERFLOW
340 // @CTK NO_ASF
341 function safeTransfer(address _to, uint256 _amount, bytes calldata _data)
342     external
343 {
344     require(transfer(_to, _amount), "ERC20: failed transfer");
345     require(_checkOnERC20Received(_to, _amount, _data), "ERC20: transfer to non
346         ERC20Receiver implementer");
347 }
348 // @CTK NO_BUF_OVERFLOW
349 // @CTK NO_OVERFLOW
350 // @CTK NO_ASF
351 function _checkOnERC20Received(address _to, uint256 _amount, bytes memory _data)
352     internal
353     returns (bool)
354 {

```

```

351     if (!_to.isContract()) {
352         return true;
353     }
354
355     return IERC20Receiver(_to).onERC20Received(_msgSender(), _amount, _data);
356 }
357 }

```

File token/MineralNFT.sol

```

1  pragma solidity ^0.5.13;
2
3  import "../ERC/ERC721Full.sol";
4  import "../utils/Ownable.sol";
5  import "../math/SafeMath.sol";
6
7  contract MineralNFT is ERC721Full, Ownable {
8      using SafeMath for uint256;
9
10     uint256 private _finalTokenId = 0;
11
12     constructor (string memory name, string memory symbol) ERC721Full(name, symbol) public {
13     }
14     // @CTK NO_BUF_OVERFLOW
15     // @CTK NO_ASF
16     /* @CTK _generateTokenId
17         @post __post._finalTokenId == _finalTokenId + 1
18         @post !__reverted -> __return == _finalTokenId
19     */
20     function _generateTokenId() internal returns (uint256) {
21         return _finalTokenId++;
22     }
23
24     // @CTK NO_BUF_OVERFLOW
25     // @CTK NO_ASF
26     /* @CTK createItem
27         @post !__reverted -> __return == _finalTokenId
28     */
29     function createItem(address to, string calldata jsonUrl) external onlyOwner returns (
30         uint256) {
31         uint256 id = _generateTokenId();
32         _mint(to, id);
33         _setTokenURI(id, jsonUrl);
34         return id;
35     }
36     // @CTK NO_BUF_OVERFLOW
37     // @CTK NO_OVERFLOW
38     // @CTK NO_ASF
39     function burnItem(uint256 tokenId) external {
40         require(!_isApprovedOrOwner(_msgSender(), tokenId), "msg.sender is not token owner");
41         _burn(_msgSender(), tokenId);
42     }
43 }

```

File token/ERC/ERC20.sol

```

1  pragma solidity ^0.5.13;
2
3  import "../../GSN/Context.sol";
4  import "../IERC20.sol";

```

```

5 import "../IERC20Receiver.sol";
6 import "../math/SafeMath.sol";
7 import "../utils/Address.sol";
8
9 /**
10  * @dev Implementation of the {IERC20} interface.
11  *
12  * This implementation is agnostic to the way tokens are created. This means
13  * that a supply mechanism has to be added in a derived contract using {_mint}.
14  * For a generic mechanism see {ERC20Mintable}.
15  *
16  * TIP: For a detailed writeup see our guide
17  * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226 [How
18  * to implement supply mechanisms].
19  *
20  * We have followed general OpenZeppelin guidelines: functions revert instead
21  * of returning `false` on failure. This behavior is nonetheless conventional
22  * and does not conflict with the expectations of ERC20 applications.
23  *
24  * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
25  * This allows applications to reconstruct the allowance for all accounts just
26  * by listening to said events. Other implementations of the EIP may not emit
27  * these events, as it isn't required by the specification.
28  *
29  * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
30  * functions have been added to mitigate the well-known issues around setting
31  * allowances. See {IERC20-approve}.
32  */
33 contract ERC20 is Context, IERC20 {
34     using SafeMath for uint256;
35     using Address for address;
36
37     mapping (address => uint256) private _balances;
38
39     mapping (address => mapping (address => uint256)) private _allowances;
40
41     uint256 private _totalSupply;
42
43     /**
44      * @dev See {IERC20-totalSupply}.
45      */
46     //@CTK NO_ASF
47     //@CTK NO_OVERFLOW
48     //@CTK NO_BUF_OVERFLOW
49     function totalSupply() public view returns (uint256) {
50         return _totalSupply;
51     }
52
53     /**
54      * @dev See {IERC20-balanceOf}.
55      */
56     //@CTK NO_ASF
57     //@CTK NO_OVERFLOW
58     //@CTK NO_BUF_OVERFLOW
59     function balanceOf(address account) public view returns (uint256) {
60         return _balances[account];
61     }
62

```

```

63  /**
64  * @dev See {IERC20-transfer}.
65  *
66  * Requirements:
67  *
68  * - `recipient` cannot be the zero address.
69  * - the caller must have a balance of at least `amount`.
70  */
71  function transfer(address recipient, uint256 amount) public returns (bool) {
72      _transfer(_msgSender(), recipient, amount);
73      return true;
74  }
75
76  /**
77  * @dev See {IERC20-allowance}.
78  */
79  //@CTK NO_ASF
80  //@CTK NO_OVERFLOW
81  //@CTK NO_BUF_OVERFLOW
82  function allowance(address owner, address spender) public view returns (uint256) {
83      return _allowances[owner][spender];
84  }
85
86  /**
87  * @dev See {IERC20-approve}.
88  *
89  * Requirements:
90  *
91  * - `spender` cannot be the zero address.
92  */
93  //@CTK NO_ASF
94  //@CTK NO_OVERFLOW
95  //@CTK NO_BUF_OVERFLOW
96  function approve(address spender, uint256 amount) public returns (bool) {
97      _approve(_msgSender(), spender, amount);
98      return true;
99  }
100
101  /**
102  * @dev See {IERC20-transferFrom}.
103  *
104  * Emits an {Approval} event indicating the updated allowance. This is not
105  * required by the EIP. See the note at the beginning of {ERC20};
106  *
107  * Requirements:
108  * - `sender` and `recipient` cannot be the zero address.
109  * - `sender` must have a balance of at least `amount`.
110  * - the caller must have allowance for `sender`'s tokens of at least
111  *   `amount`.
112  */
113  function transferFrom(address sender, address recipient, uint256 amount) public returns
      (bool) {
114      _transfer(sender, recipient, amount);
115      _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20:
          transfer amount exceeds allowance"));
116      return true;
117  }
118

```

```

119  /**
120   * @dev Atomically increases the allowance granted to `spender` by the caller.
121   *
122   * This is an alternative to {approve} that can be used as a mitigation for
123   * problems described in {IERC20-approve}.
124   *
125   * Emits an {Approval} event indicating the updated allowance.
126   *
127   * Requirements:
128   *
129   * - `spender` cannot be the zero address.
130   */
131   //@CTK NO_ASF
132   //@CTK NO_OVERFLOW
133   //@CTK NO_BUF_OVERFLOW
134   function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
135       _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
136       return true;
137   }
138
139  /**
140   * @dev Atomically decreases the allowance granted to `spender` by the caller.
141   *
142   * This is an alternative to {approve} that can be used as a mitigation for
143   * problems described in {IERC20-approve}.
144   *
145   * Emits an {Approval} event indicating the updated allowance.
146   *
147   * Requirements:
148   *
149   * - `spender` cannot be the zero address.
150   * - `spender` must have allowance for the caller of at least
151   *   `subtractedValue`.
152   */
153   function decreaseAllowance(address spender, uint256 subtractedValue) public returns (
154       bool) {
155       _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(
156           subtractedValue, "ERC20: decreased allowance below zero"));
157       return true;
158   }
159
160  /**
161   * @dev Moves tokens `amount` from `sender` to `recipient`.
162   *
163   * This is internal function is equivalent to {transfer}, and can be used to
164   * e.g. implement automatic token fees, slashing mechanisms, etc.
165   *
166   * Emits a {Transfer} event.
167   *
168   * Requirements:
169   *
170   * - `sender` cannot be the zero address.
171   * - `recipient` cannot be the zero address.
172   * - `sender` must have a balance of at least `amount`.
173   */
174   function _transfer(address sender, address recipient, uint256 amount) internal {
175       require(sender != address(0), "ERC20: transfer from the zero address");
176       require(recipient != address(0), "ERC20: transfer to the zero address");

```

```

175
176     _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds
177         balance");
178     _balances[recipient] = _balances[recipient].add(amount);
179     emit Transfer(sender, recipient, amount);
180 }
181
182 /** @dev Creates `amount` tokens and assigns them to `account`, increasing
183     * the total supply.
184     *
185     * Emits a {Transfer} event with `from` set to the zero address.
186     *
187     * Requirements
188     *
189     * - `to` cannot be the zero address.
190     */
191 //@CTK NO_ASF
192 //@CTK NO_OVERFLOW
193 //@CTK NO_BUF_OVERFLOW
194 function _mint(address account, uint256 amount) internal {
195     require(account != address(0), "ERC20: mint to the zero address");
196
197     _totalSupply = _totalSupply.add(amount);
198     _balances[account] = _balances[account].add(amount);
199     emit Transfer(address(0), account, amount);
200 }
201
202 /**
203     * @dev Destroys `amount` tokens from `account`, reducing the
204     * total supply.
205     *
206     * Emits a {Transfer} event with `to` set to the zero address.
207     *
208     * Requirements
209     *
210     * - `account` cannot be the zero address.
211     * - `account` must have at least `amount` tokens.
212     */
213 function _burn(address account, uint256 amount) internal {
214     require(account != address(0), "ERC20: burn from the zero address");
215
216     _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds
217         balance");
218     _totalSupply = _totalSupply.sub(amount);
219     emit Transfer(account, address(0), amount);
220 }
221
222 /**
223     * @dev Sets `amount` as the allowance of `spender` over the `owner`'s tokens.
224     *
225     * This is internal function is equivalent to `approve`, and can be used to
226     * e.g. set automatic allowances for certain subsystems, etc.
227     *
228     * Emits an {Approval} event.
229     *
230     * Requirements:
231     *
232     * - `owner` cannot be the zero address.

```

```

231     * - `spender` cannot be the zero address.
232     */
233     //@CTK NO_ASF
234     //@CTK NO_OVERFLOW
235     //@CTK NO_BUF_OVERFLOW
236     function _approve(address owner, address spender, uint256 amount) internal {
237         require(owner != address(0), "ERC20: approve from the zero address");
238         require(spender != address(0), "ERC20: approve to the zero address");
239
240         _allowances[owner][spender] = amount;
241         emit Approval(owner, spender, amount);
242     }
243
244     /**
245     * @dev Destroys `amount` tokens from `account`. `amount` is then deducted
246     * from the caller's allowance.
247     *
248     * See {_burn} and {_approve}.
249     */
250     function _burnFrom(address account, uint256 amount) internal {
251         _burn(account, amount);
252         _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "
                ERC20: burn amount exceeds balance"));
253     }
254 }

```

File token/ERC/ERC721Enumerable.sol

```

1  pragma solidity ^0.5.13;
2
3  import "../GSN/Context.sol";
4  import "../IERC721Enumerable.sol";
5  import "../ERC721.sol";
6  import "../ERC165.sol";
7
8  /**
9   * @title ERC-721 Non-Fungible Token with optional enumeration extension logic
10   * @dev See https://eips.ethereum.org/EIPS/eip-721
11   */
12  contract ERC721Enumerable is Context, ERC165, ERC721, IERC721Enumerable {
13      // Mapping from owner to list of owned token IDs
14      mapping(address => uint256[]) private _ownedTokens;
15
16      // Mapping from token ID to index of the owner tokens list
17      mapping(uint256 => uint256) private _ownedTokensIndex;
18
19      // Array with all token ids, used for enumeration
20      uint256[] private _allTokens;
21
22      // Mapping from token id to position in the allTokens array
23      mapping(uint256 => uint256) private _allTokensIndex;
24
25      /*
26       * bytes4(keccak256('totalSupply()')) == 0x18160ddd
27       * bytes4(keccak256('tokenOfOwnerByIndex(address,uint256)')) == 0x2f745c59
28       * bytes4(keccak256('tokenByIndex(uint256)')) == 0x4f6ccce7
29       *
30       * => 0x18160ddd ^ 0x2f745c59 ^ 0x4f6ccce7 == 0x780e9d63
31       */

```



```

32 bytes4 private constant _INTERFACE_ID_ERC721_ENUMERABLE = 0x780e9d63;
33
34 /**
35  * @dev Constructor function.
36  */
37  //@CTK NO_ASF
38  //@CTK NO_OVERFLOW
39  //@CTK NO_BUF_OVERFLOW
40  constructor () public {
41      // register the supported interface to conform to ERC721Enumerable via ERC165
42      _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
43  }
44
45  /**
46  * @dev Gets the token ID at a given index of the tokens list of the requested owner.
47  * @param owner address owning the tokens list to be accessed
48  * @param index uint256 representing the index to be accessed of the requested tokens
49  *         list
50  * @return uint256 token ID at the given index of the tokens list owned by the requested
51  *         address
52  */
53  function tokenOfOwnerByIndex(address owner, uint256 index) public view returns (uint256)
54  {
55      require(index < balanceOf(owner), "ERC721Enumerable: owner index out of bounds");
56      return _ownedTokens[owner][index];
57  }
58
59  /**
60  * @dev Gets the total amount of tokens stored by the contract.
61  * @return uint256 representing the total amount of tokens
62  */
63  //@CTK NO_ASF
64  //@CTK NO_OVERFLOW
65  //@CTK NO_BUF_OVERFLOW
66  function totalSupply() public view returns (uint256) {
67      return _allTokens.length;
68  }
69
70  /**
71  * @dev Gets the token ID at a given index of all the tokens in this contract
72  * Reverts if the index is greater or equal to the total number of tokens.
73  * @param index uint256 representing the index to be accessed of the tokens list
74  * @return uint256 token ID at the given index of the tokens list
75  */
76  //@CTK NO_ASF
77  //@CTK NO_OVERFLOW
78  //@CTK NO_BUF_OVERFLOW
79  function tokenByIndex(uint256 index) public view returns (uint256) {
80      require(index < totalSupply(), "ERC721Enumerable: global index out of bounds");
81      return _allTokens[index];
82  }
83
84  /**
85  * @dev Internal function to transfer ownership of a given token ID to another address.
86  * As opposed to transferFrom, this imposes no restrictions on msg.sender.
87  * @param from current owner of the token
88  * @param to address to receive the ownership of the given token ID

```

```

87  * @param tokenId uint256 ID of the token to be transferred
88  */
89  function _transferFrom(address from, address to, uint256 tokenId) internal {
90      super._transferFrom(from, to, tokenId);
91
92      _removeTokenFromOwnerEnumeration(from, tokenId);
93
94      _addTokenToOwnerEnumeration(to, tokenId);
95  }
96
97  /**
98  * @dev Internal function to mint a new token.
99  * Reverts if the given token ID already exists.
100  * @param to address the beneficiary that will own the minted token
101  * @param tokenId uint256 ID of the token to be minted
102  */
103  //@CTK NO_ASF
104  //@CTK NO_BUF_OVERFLOW
105  function _mint(address to, uint256 tokenId) internal {
106      super._mint(to, tokenId);
107
108      _addTokenToOwnerEnumeration(to, tokenId);
109
110      _addTokenToAllTokensEnumeration(tokenId);
111  }
112
113  /**
114  * @dev Internal function to burn a specific token.
115  * Reverts if the token does not exist.
116  * Deprecated, use {ERC721-burn} instead.
117  * @param owner owner of the token to burn
118  * @param tokenId uint256 ID of the token being burned
119  */
120
121  function _burn(address owner, uint256 tokenId) internal {
122      super._burn(owner, tokenId);
123
124      _removeTokenFromOwnerEnumeration(owner, tokenId);
125      // Since tokenId will be deleted, we can clear its slot in _ownedTokensIndex to
126      // trigger a gas refund
127      _ownedTokensIndex[tokenId] = 0;
128
129      _removeTokenFromAllTokensEnumeration(tokenId);
130  }
131
132  /**
133  * @dev Gets the list of token IDs of the requested owner.
134  * @param owner address owning the tokens
135  * @return uint256[] List of token IDs owned by the requested address
136  */
137  //@CTK NO_ASF
138  //@CTK NO_OVERFLOW
139  //@CTK NO_BUF_OVERFLOW
140  function _tokensOfOwner(address owner) internal view returns (uint256[] storage) {
141      return _ownedTokens[owner];
142  }
143  /**

```

```

144  * @dev Private function to add a token to this extension's ownership-tracking data
      structures.
145  * @param to address representing the new owner of the given token ID
146  * @param tokenId uint256 ID of the token to be added to the tokens list of the given
      address
147  */
148  //@CTK NO_ASF
149  //@CTK NO_BUF_OVERFLOW
150  function _addTokenToOwnerEnumeration(address to, uint256 tokenId) private {
151      _ownedTokensIndex[tokenId] = _ownedTokens[to].length;
152      _ownedTokens[to].push(tokenId);
153  }
154
155  /**
156  * @dev Private function to add a token to this extension's token tracking data
      structures.
157  * @param tokenId uint256 ID of the token to be added to the tokens list
158  */
159  //@CTK NO_ASF
160  //@CTK NO_BUF_OVERFLOW
161  function _addTokenToAllTokensEnumeration(uint256 tokenId) private {
162      _allTokensIndex[tokenId] = _allTokens.length;
163      _allTokens.push(tokenId);
164  }
165
166  /**
167  * @dev Private function to remove a token from this extension's ownership-tracking data
      structures. Note that
168  * while the token is not assigned a new owner, the `_ownedTokensIndex` mapping is `_not_`
      updated: this allows for
169  * gas optimizations e.g. when performing a transfer operation (avoiding double writes).
170  * This has O(1) time complexity, but alters the order of the `_ownedTokens` array.
171  * @param from address representing the previous owner of the given token ID
172  * @param tokenId uint256 ID of the token to be removed from the tokens list of the
      given address
173  */
174
175  function _removeTokenFromOwnerEnumeration(address from, uint256 tokenId) private {
176      // To prevent a gap in from's tokens array, we store the last token in the index of
      the token to delete, and
177      // then delete the last slot (swap and pop).
178
179      uint256 lastTokenIndex = _ownedTokens[from].length.sub(1);
180      uint256 tokenIndex = _ownedTokensIndex[tokenId];
181
182      // When the token to delete is the last token, the swap operation is unnecessary
183      if (tokenIndex != lastTokenIndex) {
184          uint256 lastTokenId = _ownedTokens[from][lastTokenIndex];
185
186          _ownedTokens[from][tokenIndex] = lastTokenId; // Move the last token to the slot
      of the to-delete token
187          _ownedTokensIndex[lastTokenId] = tokenIndex; // Update the moved token's index
188      }
189
190      // This also deletes the contents at the last position of the array
191      _ownedTokens[from].length--;
192
193      // Note that _ownedTokensIndex[tokenId] hasn't been cleared: it still points to the

```

```

194         old slot (now occupied by
195         // lastTokenId, or just over the end of the array if the token was the last one).
196     }
197     /**
198     * @dev Private function to remove a token from this extension's token tracking data
199     *       structures.
200     * This has O(1) time complexity, but alters the order of the _allTokens array.
201     * @param tokenId uint256 ID of the token to be removed from the tokens list
202     */
203     function _removeTokenFromAllTokensEnumeration(uint256 tokenId) private {
204         // To prevent a gap in the tokens array, we store the last token in the index of the
205         // token to delete, and
206         // then delete the last slot (swap and pop).
207
208         uint256 lastTokenIndex = _allTokens.length.sub(1);
209         uint256 tokenIndex = _allTokensIndex[tokenId];
210
211         // When the token to delete is the last token, the swap operation is unnecessary.
212         // However, since this occurs so
213         // rarely (when the last minted token is burnt) that we still do the swap here to
214         // avoid the gas cost of adding
215         // an 'if' statement (like in _removeTokenFromOwnerEnumeration)
216         uint256 lastTokenId = _allTokens[lastTokenIndex];
217
218         _allTokens[tokenIndex] = lastTokenId; // Move the last token to the slot of the to-
219         // delete token
220         _allTokensIndex[lastTokenId] = tokenIndex; // Update the moved token's index
221
222         // This also deletes the contents at the last position of the array
223         _allTokens.length--;
224         _allTokensIndex[tokenId] = 0;
225     }
226 }

```

File token/ERC/ERC721Metadata.sol

```

1  pragma solidity ^0.5.13;
2
3  import "../GSN/Context.sol";
4  import "../ERC721.sol";
5  import "../IERC721Metadata.sol";
6  import "../ERC165.sol";
7
8  contract ERC721Metadata is Context, ERC165, ERC721, IERC721Metadata {
9      // Token name
10     string private _name;
11
12     // Token symbol
13     string private _symbol;
14
15     // Base URI
16     string private _baseURI;
17
18     // Optional mapping for token URIs
19     mapping(uint256 => string) private _tokenURIs;
20
21     /*

```

```

22 * bytes4(keccak256('name()')) == 0x06fdde03
23 * bytes4(keccak256('symbol()')) == 0x95d89b41
24 * bytes4(keccak256('tokenURI(uint256)')) == 0xc87b56dd
25 *
26 * => 0x06fdde03 ^ 0x95d89b41 ^ 0xc87b56dd == 0x5b5e139f
27 */
28 bytes4 private constant _INTERFACE_ID_ERC721_METADATA = 0x5b5e139f;
29
30 /**
31 * @dev Constructor function
32 */
33 //@CTK NO_ASF
34 //@CTK NO_OVERFLOW
35 //@CTK NO_BUF_OVERFLOW
36 constructor (string memory name, string memory symbol) public {
37     _name = name;
38     _symbol = symbol;
39
40     // register the supported interfaces to conform to ERC721 via ERC165
41     _registerInterface(_INTERFACE_ID_ERC721_METADATA);
42 }
43
44 /**
45 * @dev Gets the token name.
46 * @return string representing the token name
47 */
48 //@CTK NO_ASF
49 //@CTK NO_OVERFLOW
50 //@CTK NO_BUF_OVERFLOW
51 function name() external view returns (string memory) {
52     return _name;
53 }
54
55 /**
56 * @dev Gets the token symbol.
57 * @return string representing the token symbol
58 */
59 //@CTK NO_ASF
60 //@CTK NO_OVERFLOW
61 //@CTK NO_BUF_OVERFLOW
62 function symbol() external view returns (string memory) {
63     return _symbol;
64 }
65
66 /**
67 * @dev Returns the URI for a given token ID. May return an empty string.
68 *
69 * If the token's URI is non-empty and a base URI was set (via
70 * {_setBaseURI}), it will be added to the token ID's URI as a prefix.
71 *
72 * Reverts if the token ID does not exist.
73 */
74 //@CTK NO_ASF
75 //@CTK NO_OVERFLOW
76 //@CTK NO_BUF_OVERFLOW
77 function tokenURI(uint256 tokenId) external view returns (string memory) {
78     require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");
79

```

```

80     string memory _tokenURI = _tokenURIs[tokenId];
81
82     // Even if there is a base URI, it is only appended to non-empty token-specific URIs
83     if (bytes(_tokenURI).length == 0) {
84         return "";
85     } else {
86         // abi.encodePacked is being used to concatenate strings
87         return string(abi.encodePacked(_baseURI, _tokenURI));
88     }
89 }
90
91 /**
92  * @dev Returns the base URI set via {_setBaseURI}. This will be
93  * automatically added as a prefix in {tokenURI} to each token's URI, when
94  * they are non-empty.
95  */
96 //@CTK NO_ASF
97 //@CTK NO_OVERFLOW
98 //@CTK NO_BUF_OVERFLOW
99 function baseURI() external view returns (string memory) {
100     return _baseURI;
101 }
102
103 /**
104  * @dev Internal function to set the token URI for a given token.
105  *
106  * Reverts if the token ID does not exist.
107  *
108  * TIP: if all token IDs share a prefix (e.g. if your URIs look like
109  * `http://api.myproject.com/token/<id>`), use {_setBaseURI} to store
110  * it and save gas.
111  */
112 //@CTK NO_ASF
113 //@CTK NO_OVERFLOW
114 //@CTK NO_BUF_OVERFLOW
115 function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal {
116     require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
117     _tokenURIs[tokenId] = _tokenURI;
118 }
119
120 /**
121  * @dev Internal function to set the base URI for all token IDs. It is
122  * automatically added as a prefix to the value returned in {tokenURI}.
123  *
124  * _Available since v2.5.0._
125  */
126 //@CTK NO_ASF
127 //@CTK NO_OVERFLOW
128 //@CTK NO_BUF_OVERFLOW
129 function _setBaseURI(string memory uri) internal {
130     _baseURI = uri;
131 }
132
133 /**
134  * @dev Internal function to burn a specific token.
135  * Reverts if the token does not exist.
136  * Deprecated, use _burn(uint256) instead.
137  * @param owner owner of the token to burn

```

```

138     * @param tokenId uint256 ID of the token being burned by the msg.sender
139     */
140     //@CTK NO_ASF
141     //@CTK NO_OVERFLOW
142     //@CTK NO_BUF_OVERFLOW
143     function _burn(address owner, uint256 tokenId) internal {
144         super._burn(owner, tokenId);
145
146         // Clear metadata (if any)
147         if (bytes(_tokenURIs[tokenId]).length != 0) {
148             delete _tokenURIs[tokenId];
149         }
150     }
151 }

```

File token/ERC/ERC721.sol

```

1  pragma solidity ^0.5.13;
2
3  import "../GSN/Context.sol";
4  import "../IERC721.sol";
5  import "../IERC721Receiver.sol";
6  import "../ERC165.sol";
7  import "../math/SafeMath.sol";
8  import "../utils/Address.sol";
9  import "../drafts/Counters.sol";
10
11  /**
12   * @title ERC721 Non-Fungible Token Standard basic implementation
13   * @dev see https://eips.ethereum.org/EIPS/eip-721
14   */
15  contract ERC721 is Context, ERC165, IERC721 {
16      using SafeMath for uint256;
17      using Address for address;
18      using Counters for Counters.Counter;
19
20      // Equals to `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
21      // which can be also obtained as `IERC721Receiver(0).onERC721Received.selector`
22      bytes4 private constant _ERC721_RECEIVED = 0x150b7a02;
23
24      // Mapping from token ID to owner
25      mapping (uint256 => address) private _tokenOwner;
26
27      // Mapping from token ID to approved address
28      mapping (uint256 => address) private _tokenApprovals;
29
30      // Mapping from owner to number of owned tokens
31      mapping (address => Counters.Counter) private _ownedTokensCount;
32
33      // Mapping from owner to operator approvals
34      mapping (address => mapping (address => bool)) private _operatorApprovals;
35
36      /*
37       *   bytes4(keccak256('balanceOf(address)')) == 0x70a08231
38       *   bytes4(keccak256('ownerOf(uint256)')) == 0x6352211e
39       *   bytes4(keccak256('approve(address,uint256)')) == 0x095ea7b3
40       *   bytes4(keccak256('getApproved(uint256)')) == 0x081812fc
41       *   bytes4(keccak256('setApprovalForAll(address,bool)')) == 0xa22cb465
42       *   bytes4(keccak256('isApprovedForAll(address,address)')) == 0xe985e9c5

```

```

43 * bytes4(keccak256('transferFrom(address,address,uint256)')) == 0x23b872dd
44 * bytes4(keccak256('safeTransferFrom(address,address,uint256)')) == 0x42842e0e
45 * bytes4(keccak256('safeTransferFrom(address,address,uint256,bytes)')) == 0xb88d4fde
46 *
47 * => 0x70a08231 ^ 0x6352211e ^ 0x095ea7b3 ^ 0x081812fc ^
48 * 0xa22cb465 ^ 0xe985e9c ^ 0x23b872dd ^ 0x42842e0e ^ 0xb88d4fde == 0x80ac58cd
49 */
50 bytes4 private constant _INTERFACE_ID_ERC721 = 0x80ac58cd;
51 //@CTK NO_ASF
52 //@CTK NO_OVERFLOW
53 //@CTK NO_BUF_OVERFLOW
54 constructor () public {
55     // register the supported interfaces to conform to ERC721 via ERC165
56     _registerInterface(_INTERFACE_ID_ERC721);
57 }
58
59 /**
60 * @dev Gets the balance of the specified address.
61 * @param owner address to query the balance of
62 * @return uint256 representing the amount owned by the passed address
63 */
64 //@CTK NO_ASF
65 //@CTK NO_OVERFLOW
66 //@CTK NO_BUF_OVERFLOW
67 function balanceOf(address owner) public view returns (uint256) {
68     require(owner != address(0), "ERC721: balance query for the zero address");
69
70     return _ownedTokensCount[owner].current();
71 }
72
73 /**
74 * @dev Gets the owner of the specified token ID.
75 * @param tokenId uint256 ID of the token to query the owner of
76 * @return address currently marked as the owner of the given token ID
77 */
78 //@CTK NO_ASF
79 //@CTK NO_OVERFLOW
80 //@CTK NO_BUF_OVERFLOW
81 function ownerOf(uint256 tokenId) public view returns (address) {
82     address owner = _tokenOwner[tokenId];
83     require(owner != address(0), "ERC721: owner query for nonexistent token");
84
85     return owner;
86 }
87
88 /**
89 * @dev Approves another address to transfer the given token ID
90 * The zero address indicates there is no approved address.
91 * There can only be one approved address per token at a given time.
92 * Can only be called by the token owner or an approved operator.
93 * @param to address to be approved for the given token ID
94 * @param tokenId uint256 ID of the token to be approved
95 */
96 //@CTK NO_ASF
97 //@CTK NO_OVERFLOW
98 //@CTK NO_BUF_OVERFLOW
99 function approve(address to, uint256 tokenId) public {
100     address owner = ownerOf(tokenId);

```



```

101     require(to != owner, "ERC721: approval to current owner");
102
103     require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
104         "ERC721: approve caller is not owner nor approved for all"
105     );
106
107     _tokenApprovals[tokenId] = to;
108     emit Approval(owner, to, tokenId);
109 }
110
111 /**
112  * @dev Gets the approved address for a token ID, or zero if no address set
113  * Reverts if the token ID does not exist.
114  * @param tokenId uint256 ID of the token to query the approval of
115  * @return address currently approved for the given token ID
116  */
117 // @CTK NO_ASF
118 // @CTK NO_OVERFLOW
119 // @CTK NO_BUF_OVERFLOW
120 function getApproved(uint256 tokenId) public view returns (address) {
121     require(_exists(tokenId), "ERC721: approved query for nonexistent token");
122
123     return _tokenApprovals[tokenId];
124 }
125
126 /**
127  * @dev Sets or unsets the approval of a given operator
128  * An operator is allowed to transfer all tokens of the sender on their behalf.
129  * @param to operator address to set the approval
130  * @param approved representing the status of the approval to be set
131  */
132 // @CTK NO_ASF
133 // @CTK NO_OVERFLOW
134 // @CTK NO_BUF_OVERFLOW
135 function setApprovalForAll(address to, bool approved) public {
136     require(to != _msgSender(), "ERC721: approve to caller");
137
138     _operatorApprovals[_msgSender()][to] = approved;
139     emit ApprovalForAll(_msgSender(), to, approved);
140 }
141
142 /**
143  * @dev Tells whether an operator is approved by a given owner.
144  * @param owner owner address which you want to query the approval of
145  * @param operator operator address which you want to query the approval of
146  * @return bool whether the given operator is approved by the given owner
147  */
148 // @CTK NO_ASF
149 // @CTK NO_OVERFLOW
150 // @CTK NO_BUF_OVERFLOW
151 function isApprovedForAll(address owner, address operator) public view returns (bool) {
152     return _operatorApprovals[owner][operator];
153 }
154
155 /**
156  * @dev Transfers the ownership of a given token ID to another address.
157  * Usage of this method is discouraged, use {safeTransferFrom} whenever possible.
158  * Requires the msg.sender to be the owner, approved, or operator.

```

```

159  * @param from current owner of the token
160  * @param to address to receive the ownership of the given token ID
161  * @param tokenId uint256 ID of the token to be transferred
162  */
163
164  function transferFrom(address from, address to, uint256 tokenId) public {
165      //solhint-disable-next-line max-line-length
166      require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not
167          owner nor approved");
168      _transferFrom(from, to, tokenId);
169  }
170
171  /**
172   * @dev Safely transfers the ownership of a given token ID to another address
173   * If the target address is a contract, it must implement {IERC721Receiver-
174       onERC721Received},
175   * which is called upon a safe transfer, and return the magic value
176   * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
177   * the transfer is reverted.
178   * Requires the msg.sender to be the owner, approved, or operator
179   * @param from current owner of the token
180   * @param to address to receive the ownership of the given token ID
181   * @param tokenId uint256 ID of the token to be transferred
182   */
183  function safeTransferFrom(address from, address to, uint256 tokenId) public {
184      safeTransferFrom(from, to, tokenId, "");
185  }
186
187  /**
188   * @dev Safely transfers the ownership of a given token ID to another address
189   * If the target address is a contract, it must implement {IERC721Receiver-
190       onERC721Received},
191   * which is called upon a safe transfer, and return the magic value
192   * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
193   * the transfer is reverted.
194   * Requires the _msgSender() to be the owner, approved, or operator
195   * @param from current owner of the token
196   * @param to address to receive the ownership of the given token ID
197   * @param tokenId uint256 ID of the token to be transferred
198   * @param _data bytes data to send along with a safe transfer check
199   */
200  function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data)
201      public {
202      require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not
203          owner nor approved");
204      _safeTransferFrom(from, to, tokenId, _data);
205  }
206
207  /**
208   * @dev Safely transfers the ownership of a given token ID to another address
209   * If the target address is a contract, it must implement `onERC721Received`,
210   * which is called upon a safe transfer, and return the magic value
211   * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
212   * the transfer is reverted.
213   * Requires the msg.sender to be the owner, approved, or operator

```

```

212 * @param from current owner of the token
213 * @param to address to receive the ownership of the given token ID
214 * @param tokenId uint256 ID of the token to be transferred
215 * @param _data bytes data to send along with a safe transfer check
216 */
217
218 function _safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data
    ) internal {
219     _transferFrom(from, to, tokenId);
220     require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721: transfer to non
        ERC721Receiver implementer");
221 }
222
223 /**
224 * @dev Returns whether the specified token exists.
225 * @param tokenId uint256 ID of the token to query the existence of
226 * @return bool whether the token exists
227 */
228 // @CTK NO_ASF
229 // @CTK NO_OVERFLOW
230 // @CTK NO_BUF_OVERFLOW
231 function _exists(uint256 tokenId) internal view returns (bool) {
232     address owner = _tokenOwner[tokenId];
233     return owner != address(0);
234 }
235
236 /**
237 * @dev Returns whether the given spender can transfer a given token ID.
238 * @param spender address of the spender to query
239 * @param tokenId uint256 ID of the token to be transferred
240 * @return bool whether the msg.sender is approved for the given token ID,
241 * is an operator of the owner, or is the owner of the token
242 */
243 // @CTK NO_ASF
244 // @CTK NO_OVERFLOW
245 // @CTK NO_BUF_OVERFLOW
246 function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (
    bool) {
247     require(_exists(tokenId), "ERC721: operator query for nonexistent token");
248     address owner = ownerOf(tokenId);
249     return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(
        owner, spender));
250 }
251
252 /**
253 * @dev Internal function to safely mint a new token.
254 * Reverts if the given token ID already exists.
255 * If the target address is a contract, it must implement `onERC721Received`,
256 * which is called upon a safe transfer, and return the magic value
257 * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
258 * the transfer is reverted.
259 * @param to The address that will own the minted token
260 * @param tokenId uint256 ID of the token to be minted
261 */
262
263 function _safeMint(address to, uint256 tokenId) internal {
264     _safeMint(to, tokenId, "");
265 }

```

```

266
267 /**
268  * @dev Internal function to safely mint a new token.
269  * Reverts if the given token ID already exists.
270  * If the target address is a contract, it must implement `onERC721Received`,
271  * which is called upon a safe transfer, and return the magic value
272  * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
273  * the transfer is reverted.
274  * @param to The address that will own the minted token
275  * @param tokenId uint256 ID of the token to be minted
276  * @param _data bytes data to send along with a safe transfer check
277  */
278
279 function _safeMint(address to, uint256 tokenId, bytes memory _data) internal {
280     _mint(to, tokenId);
281     require(_checkOnERC721Received(address(0), to, tokenId, _data), "ERC721: transfer to
282         non ERC721Receiver implementer");
283 }
284
285 /**
286  * @dev Internal function to mint a new token.
287  * Reverts if the given token ID already exists.
288  * @param to The address that will own the minted token
289  * @param tokenId uint256 ID of the token to be minted
290  */
291 // @CTK NO_ASF
292 // @CTK NO_BUF_OVERFLOW
293 function _mint(address to, uint256 tokenId) internal {
294     require(to != address(0), "ERC721: mint to the zero address");
295     require(!_exists(tokenId), "ERC721: token already minted");
296
297     _tokenOwner[tokenId] = to;
298     _ownedTokensCount[to].increment();
299
300     emit Transfer(address(0), to, tokenId);
301 }
302
303 /**
304  * @dev Internal function to burn a specific token.
305  * Reverts if the token does not exist.
306  * Deprecated, use {_burn} instead.
307  * @param owner owner of the token to burn
308  * @param tokenId uint256 ID of the token being burned
309  */
310
311 function _burn(address owner, uint256 tokenId) internal {
312     require(ownerOf(tokenId) == owner, "ERC721: burn of token that is not own");
313
314     _clearApproval(tokenId);
315
316     _ownedTokensCount[owner].decrement();
317     _tokenOwner[tokenId] = address(0);
318
319     emit Transfer(owner, address(0), tokenId);
320 }
321
322 /**
323  * @dev Internal function to burn a specific token.

```

```

323     * Reverts if the token does not exist.
324     * @param tokenId uint256 ID of the token being burned
325     */
326
327     function _burn(uint256 tokenId) internal {
328         _burn(ownerOf(tokenId), tokenId);
329     }
330
331     /**
332     * @dev Internal function to transfer ownership of a given token ID to another address.
333     * As opposed to {transferFrom}, this imposes no restrictions on msg.sender.
334     * @param from current owner of the token
335     * @param to address to receive the ownership of the given token ID
336     * @param tokenId uint256 ID of the token to be transferred
337     */
338     function _transferFrom(address from, address to, uint256 tokenId) internal {
339         require(ownerOf(tokenId) == from, "ERC721: transfer of token that is not own");
340         require(to != address(0), "ERC721: transfer to the zero address");
341
342         _clearApproval(tokenId);
343
344         _ownedTokensCount[from].decrement();
345         _ownedTokensCount[to].increment();
346
347         _tokenOwner[tokenId] = to;
348
349         emit Transfer(from, to, tokenId);
350     }
351
352     /**
353     * @dev Internal function to invoke {IERC721Receiver-onERC721Received} on a target
354     * address.
355     * The call is not executed if the target address is not a contract.
356     * This is an internal detail of the `ERC721` contract and its use is deprecated.
357     * @param from address representing the previous owner of the given token ID
358     * @param to target address that will receive the tokens
359     * @param tokenId uint256 ID of the token to be transferred
360     * @param _data bytes optional data to send along with the call
361     * @return bool whether the call correctly returned the expected magic value
362     */
363
364     function _checkOnERC721Received(address from, address to, uint256 tokenId, bytes memory
365         _data)
366         internal returns (bool)
367     {
368         if (!to.isContract()) {
369             return true;
370         }
371         bytes4 retval = IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId,
372             _data);
373         return (retval == _ERC721_RECEIVED);
374     }
375
376     /**
377     * @dev Private function to clear current approval of a given token ID.
378     * @param tokenId uint256 ID of the token to be transferred
379     */

```

```

378 //CTK NO_ASF
379 //CTK NO_OVERFLOW
380 //CTK NO_BUF_OVERFLOW
381 function _clearApproval(uint256 tokenId) private {
382     if (_tokenApprovals[tokenId] != address(0)) {
383         _tokenApprovals[tokenId] = address(0);
384     }
385 }
386 }

```

File token/ERC/ERC165.sol

```

1 pragma solidity ^0.5.13;
2
3 import "./IERC165.sol";
4
5 /**
6  * @dev Implementation of the {IERC165} interface.
7  *
8  * Contracts may inherit from this and call {_registerInterface} to declare
9  * their support of an interface.
10 */
11 contract ERC165 is IERC165 {
12     /*
13      * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
14      */
15     bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7;
16
17     /**
18      * @dev Mapping of interface ids to whether or not it's supported.
19      */
20     mapping(bytes4 => bool) private _supportedInterfaces;
21     //CTK NO_ASF
22     //CTK NO_OVERFLOW
23     //CTK NO_BUF_OVERFLOW
24     constructor () internal {
25         // Derived contracts need only register support for their own interfaces,
26         // we register support for ERC165 itself here
27         _registerInterface(_INTERFACE_ID_ERC165);
28     }
29
30     /**
31      * @dev See {IERC165-supportsInterface}.
32      *
33      * Time complexity O(1), guaranteed to always use less than 30 000 gas.
34      */
35     //CTK NO_ASF
36     //CTK NO_OVERFLOW
37     //CTK NO_BUF_OVERFLOW
38     function supportsInterface(bytes4 interfaceId) external view returns (bool) {
39         return _supportedInterfaces[interfaceId];
40     }
41
42     /**
43      * @dev Registers the contract as an implementer of the interface defined by
44      * `interfaceId`. Support of the actual ERC165 interface is automatic and
45      * registering its interface id is not required.
46      *
47      * See {IERC165-supportsInterface}.

```

```

48  *
49  * Requirements:
50  *
51  * - `interfaceId` cannot be the ERC165 invalid interface (`0xffffffff`).
52  */
53  //@CTK NO_ASF
54  //@CTK NO_OVERFLOW
55  //@CTK NO_BUF_OVERFLOW
56  function _registerInterface(bytes4 interfaceId) internal {
57      require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
58      _supportedInterfaces[interfaceId] = true;
59  }
60 }

```

File token/ERC/ERC20Burnable.sol

```

1  pragma solidity ^0.5.13;
2
3  import "../GSN/Context.sol";
4  import "./ERC20.sol";
5
6  /**
7   * @dev Extension of {ERC20} that allows token holders to destroy both their own
8   * tokens and those that they have an allowance for, in a way that can be
9   * recognized off-chain (via event analysis).
10  */
11  contract ERC20Burnable is Context, ERC20 {
12      /**
13       * @dev Destroys `amount` tokens from the caller.
14       *
15       * See {ERC20-_burn}.
16       */
17      //@CTK NO_ASF
18      //@CTK NO_OVERFLOW
19      //@CTK NO_BUF_OVERFLOW
20      function burn(uint256 amount) public {
21          _burn(_msgSender(), amount);
22      }
23
24      /**
25       * @dev See {ERC20-_burnFrom}.
26       */
27      //@CTK NO_ASF
28      //@CTK NO_OVERFLOW
29      //@CTK NO_BUF_OVERFLOW
30      function burnFrom(address account, uint256 amount) public {
31          _burnFrom(account, amount);
32      }
33  }

```

File GSN/Context.sol

```

1  pragma solidity ^0.5.13;
2
3  /**
4   * @dev Provides information about the current execution context, including the
5   * sender of the transaction and its data. While these are generally available
6   * via msg.sender and msg.data, they should not be accessed in such a direct
7   * manner, since when dealing with GSN meta-transactions the account sending and
8   * paying for execution may not be the actual sender (as far as an application

```

```

9  * is concerned).
10 *
11 * This contract is only required for intermediate, library-like contracts.
12 */
13 contract Context {
14     // Empty internal constructor, to prevent people from mistakenly deploying
15     // an instance of this contract, which should be used via inheritance.
16     //@CTK NO_ASF
17     //@CTK NO_OVERFLOW
18     //@CTK NO_BUF_OVERFLOW
19     constructor () internal { }
20     // solhint-disable-previous-line no-empty-blocks
21     //@CTK NO_ASF
22     //@CTK NO_OVERFLOW
23     //@CTK NO_BUF_OVERFLOW
24     function _msgSender() internal view returns (address payable) {
25         return msg.sender;
26     }
27
28     function _msgData() internal view returns (bytes memory) {
29         this; // silence state mutability warning without generating bytecode - see https://
30             github.com/ethereum/solidity/issues/2691
31         return msg.data;
32     }
33 }

```

File drafts/Counters.sol

```

1  pragma solidity ^0.5.13;
2
3  import "../math/SafeMath.sol";
4
5  /**
6   * @title Counters
7   * @author Matt Condon (@shrugs)
8   * @dev Provides counters that can only be incremented or decremented by one. This can be
9   *       used e.g. to track the number
10 *
11 * Include with `using Counters for Counters.Counter;`
12 * Since it is not possible to overflow a 256 bit integer with increments of one, `
13 *       increment` can skip the {SafeMath}
14 * overflow check, thereby saving gas. This does assume however correct usage, in that the
15 *       underlying `_value` is never
16 * directly accessed.
17 */
18 library Counters {
19     using SafeMath for uint256;
20
21     struct Counter {
22         // This variable should never be directly accessed by users of the library:
23         // interactions must be restricted to
24         // the library's function. As of Solidity v0.5.2, this cannot be enforced, though
25         // there is a proposal to add
26         // this feature: see https://github.com/ethereum/solidity/issues/4637
27         uint256 _value; // default: 0
28     }
29
30     //@CTK NO_ASF
31     //@CTK NO_OVERFLOW

```



```

27  // @CTK_NO_BUF_OVERFLOW
28  function current(Counter storage counter) internal view returns (uint256) {
29      return counter._value;
30  }
31  // @CTK_NO_ASF
32  // @CTK_NO_BUF_OVERFLOW
33  function increment(Counter storage counter) internal {
34      // The {SafeMath} overflow check can be skipped here, see the comment at the top
35      counter._value += 1;
36  }
37  function decrement(Counter storage counter) internal {
38      counter._value = counter._value.sub(1);
39  }
40 }

```

File utils/Ownable.sol

```

1  pragma solidity ^0.5.13;
2
3  contract Ownable {
4      address public owner;
5      // @CTK_NO_ASF
6      // @CTK_NO_OVERFLOW
7      // @CTK_NO_BUF_OVERFLOW
8      constructor() public {
9          owner = msg.sender;
10     }
11
12     modifier onlyOwner() {
13         require (msg.sender == owner, "only Owner");
14         _;
15     }
16     // @CTK_NO_ASF
17     // @CTK_NO_OVERFLOW
18     // @CTK_NO_BUF_OVERFLOW
19     function transferOwnership(address newOwner) public onlyOwner {
20         if (newOwner != address(0))
21             owner = newOwner;
22     }
23 }

```

File utils/Address.sol

```

1  pragma solidity ^0.5.13;
2
3  /**
4   * @dev Collection of functions related to the address type
5   */
6  library Address {
7      /**
8       * @dev Returns true if `account` is a contract.
9       *
10      * This test is non-exhaustive, and there may be false-negatives: during the
11      * execution of a contract's constructor, its address will be reported as
12      * not containing a contract.
13      *
14      * IMPORTANT: It is unsafe to assume that an address for which this
15      * function returns false is an externally-owned account (EOA) and not a
16      * contract.
17      */

```

```

18  /*@CTK isContract
19      @post !__reverted -> __return == (account != msg.sender)
20  */
21  //@CTK NO_BUF_OVERFLOW
22  //@CTK NO_OVERFLOW
23  //@CTK NO_ASF
24  function isContract(address account) internal view returns (bool) {
25      return (account != msg.sender);
26      /*
27      // This method relies in extcodesize, which returns 0 for contracts in
28      // construction, since the code is only stored at the end of the
29      // constructor execution.
30
31      // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
32      // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
33      // returned
34      // for accounts without code, i.e. `keccak256('')`
35      bytes32 codehash;
36      bytes32 accountHash = 0
37          xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
38      // solhint-disable-next-line no-inline-assembly
39      assembly { codehash := extcodehash(account) }
40      return (codehash != 0x0 && codehash != accountHash);
41  */
42  }
43 }

```

File utils/BytesLib.sol

```

1  /*
2  * @title Solidity Bytes Arrays Utils
3  * @author Goncalo Sa <goncalo.sa@consensys.net>
4  *
5  * @dev Bytes tightly packed arrays utility library for ethereum contracts written in
6  *      Solidity.
7  *      The library lets you concatenate, slice and type cast bytes arrays both in memory
8  *      and storage.
9  */
10
11  pragma solidity ^0.5.13;
12
13  library BytesLib {
14      //@CTK NO_BUF_OVERFLOW
15      //@CTK NO_OVERFLOW
16      //@CTK NO_ASF
17      function concat(
18          bytes memory _preBytes,
19          bytes memory _postBytes
20      )
21      internal
22      pure
23      returns (bytes memory)
24      {
25          bytes memory tempBytes;
26
27          assembly {
28              // Get a location of some free memory and store it in tempBytes as
29              // Solidity does for memory variables.

```

```

29     tempBytes := mload(0x40)
30
31     // Store the length of the first bytes array at the beginning of
32     // the memory for tempBytes.
33     let length := mload(_preBytes)
34     mstore(tempBytes, length)
35
36     // Maintain a memory counter for the current write location in the
37     // temp bytes array by adding the 32 bytes for the array length to
38     // the starting location.
39     let mc := add(tempBytes, 0x20)
40     // Stop copying when the memory counter reaches the length of the
41     // first bytes array.
42     let end := add(mc, length)
43
44     for {
45         // Initialize a copy counter to the start of the _preBytes data,
46         // 32 bytes into its memory.
47         let cc := add(_preBytes, 0x20)
48     } lt(mc, end) {
49         // Increase both counters by 32 bytes each iteration.
50         mc := add(mc, 0x20)
51         cc := add(cc, 0x20)
52     } {
53         // Write the _preBytes data into the tempBytes memory 32 bytes
54         // at a time.
55         mstore(mc, mload(cc))
56     }
57
58     // Add the length of _postBytes to the current length of tempBytes
59     // and store it as the new length in the first 32 bytes of the
60     // tempBytes memory.
61     length := mload(_postBytes)
62     mstore(tempBytes, add(length, mload(tempBytes)))
63
64     // Move the memory counter back from a multiple of 0x20 to the
65     // actual end of the _preBytes data.
66     mc := end
67     // Stop copying when the memory counter reaches the new combined
68     // length of the arrays.
69     end := add(mc, length)
70
71     for {
72         let cc := add(_postBytes, 0x20)
73     } lt(mc, end) {
74         mc := add(mc, 0x20)
75         cc := add(cc, 0x20)
76     } {
77         mstore(mc, mload(cc))
78     }
79
80     // Update the free-memory pointer by padding our last write location
81     // to 32 bytes: add 31 bytes to the end of tempBytes to move to the
82     // next 32 byte block, then round down to the nearest multiple of
83     // 32. If the sum of the length of the two arrays is zero then add
84     // one before rounding down to leave a blank 32 bytes (the length block with 0).
85     mstore(0x40, and(
86         add(add(end, iszero(add(length, mload(_preBytes)))), 31),

```

```

87         not(31) // Round down to the nearest 32 bytes.
88     ))
89 }
90
91     return tempBytes;
92 }
93 //@CTK NO_BUF_OVERFLOW
94 //@CTK NO_OVERFLOW
95 //@CTK NO_ASF
96 function concatStorage(bytes storage _preBytes, bytes memory _postBytes) internal {
97     assembly {
98         // Read the first 32 bytes of _preBytes storage, which is the length
99         // of the array. (We don't need to use the offset into the slot
100        // because arrays use the entire slot.)
101        let fslot := sload(_preBytes_slot)
102        // Arrays of 31 bytes or less have an even value in their slot,
103        // while longer arrays have an odd value. The actual length is
104        // the slot divided by two for odd values, and the lowest order
105        // byte divided by two for even values.
106        // If the slot is even, bitwise and the slot with 255 and divide by
107        // two to get the length. If the slot is odd, bitwise and the slot
108        // with -1 and divide by two.
109        let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
110        let mlength := mload(_postBytes)
111        let newlength := add(slength, mlength)
112        // slength can contain both the length and contents of the array
113        // if length < 32 bytes so let's prepare for that
114        // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-state-variables-in-storage
115        switch add(lt(slength, 32), lt(newlength, 32))
116        case 2 {
117            // Since the new array still fits in the slot, we just need to
118            // update the contents of the slot.
119            // uint256(bytes_storage) = uint256(bytes_storage) + uint256(bytes_memory) +
120            //   new_length
121            sstore(
122                _preBytes_slot,
123                // all the modifications to the slot are inside this
124                // next block
125                add(
126                    // we can just add to the slot contents because the
127                    // bytes we want to change are the LSBs
128                    fslot,
129                    add(
130                        mul(
131                            div(
132                                // load the bytes from memory
133                                mload(add(_postBytes, 0x20)),
134                                // zero all bytes to the right
135                                exp(0x100, sub(32, mlength))
136                            ),
137                            // and now shift left the number of bytes to
138                            // leave space for the length in the slot
139                            exp(0x100, sub(32, newlength))
140                        ),
141                        // increase length by the double of the memory
142                        // bytes length
143                        mul(mlength, 2)
144                    )
145                )
146            )
147        }
148    }
149 }

```

```

143         )
144     )
145 )
146 }
147 case 1 {
148     // The stored value fits in the slot, but the combined value
149     // will exceed it.
150     // get the keccak hash to get the contents of the array
151     mstore(0x0, _preBytes_slot)
152     let sc := add(keccak256(0x0, 0x20), div(slength, 32))
153
154     // save new length
155     sstore(_preBytes_slot, add(mul(newlength, 2), 1))
156
157     // The contents of the _postBytes array start 32 bytes into
158     // the structure. Our first read should obtain the `submod`
159     // bytes that can fit into the unused space in the last word
160     // of the stored array. To get this, we read 32 bytes starting
161     // from `submod`, so the data we read overlaps with the array
162     // contents by `submod` bytes. Masking the lowest-order
163     // `submod` bytes allows us to add that value directly to the
164     // stored value.
165
166     let submod := sub(32, slength)
167     let mc := add(_postBytes, submod)
168     let end := add(_postBytes, mlength)
169     let mask := sub(exp(0x100, submod), 1)
170
171     sstore(
172         sc,
173         add(
174             and(
175                 fslot,
176                 0xffffffffffffffffffffffffffffffffffffffffffffffff00
177             ),
178             and(mload(mc), mask)
179         )
180     )
181
182     for {
183         mc := add(mc, 0x20)
184         sc := add(sc, 1)
185     } lt(mc, end) {
186         sc := add(sc, 1)
187         mc := add(mc, 0x20)
188     } {
189         sstore(sc, mload(mc))
190     }
191
192     mask := exp(0x100, sub(mc, end))
193
194     sstore(sc, mul(div(mload(mc), mask), mask))
195 }
196 default {
197     // get the keccak hash to get the contents of the array
198     mstore(0x0, _preBytes_slot)
199     // Start copying to the last used word of the stored array.
200     let sc := add(keccak256(0x0, 0x20), div(slength, 32))

```

```

201
202 // save new length
203 sstore(_preBytes_slot, add(mul(newlength, 2), 1))
204
205 // Copy over the first `submod` bytes of the new data as in
206 // case 1 above.
207 let slengthmod := mod(slength, 32)
208 let mlengthmod := mod(mlength, 32)
209 let submod := sub(32, slengthmod)
210 let mc := add(_postBytes, submod)
211 let end := add(_postBytes, mlength)
212 let mask := sub(exp(0x100, submod), 1)
213
214 sstore(sc, add(sload(sc), and(mload(mc), mask)))
215
216 for {
217     sc := add(sc, 1)
218     mc := add(mc, 0x20)
219 } lt(mc, end) {
220     sc := add(sc, 1)
221     mc := add(mc, 0x20)
222 } {
223     sstore(sc, mload(mc))
224 }
225
226 mask := exp(0x100, sub(mc, end))
227
228 sstore(sc, mul(div(mload(mc), mask), mask))
229 }
230 }
231 }
232 //@CTK NO_BUF_OVERFLOW
233 //@CTK NO_ASF
234 function slice(
235     bytes memory _bytes,
236     uint _start,
237     uint _length
238 )
239     internal
240     pure
241     returns (bytes memory)
242 {
243     require(_bytes.length >= (_start + _length), "_bytes.length >= (_start + _length)");
244
245     bytes memory tempBytes;
246
247     assembly {
248         switch iszero(_length)
249         case 0 {
250             // Get a location of some free memory and store it in tempBytes as
251             // Solidity does for memory variables.
252             tempBytes := mload(0x40)
253
254             // The first word of the slice result is potentially a partial
255             // word read from the original array. To read it, we calculate
256             // the length of that partial word and start copying that many
257             // bytes into the array. The first word we copy will start with
258             // data we don't care about, but the last `lengthmod` bytes will

```

```

259      // land at the beginning of the contents of the new array. When
260      // we're done copying, we overwrite the full first word with
261      // the actual length of the slice.
262      let lengthmod := and(_length, 31)
263
264      // The multiplication in the next line is necessary
265      // because when slicing multiples of 32 bytes (lengthmod == 0)
266      // the following copy loop was copying the origin's length
267      // and then ending prematurely not copying everything it should.
268      let mc := add(add(tempBytes, lengthmod), mul(0x20, iszero(lengthmod)))
269      let end := add(mc, _length)
270
271      for {
272          // The multiplication in the next line has the same exact purpose
273          // as the one above.
274          let cc := add(add(add(_bytes, lengthmod), mul(0x20, iszero(lengthmod))),
275              _start)
276      } lt(mc, end) {
277          mc := add(mc, 0x20)
278          cc := add(cc, 0x20)
279      } {
280          mstore(mc, mload(cc))
281      }
282
283      mstore(tempBytes, _length)
284
285      //update free-memory pointer
286      //allocating the array padded to 32 bytes like the compiler does now
287      mstore(0x40, and(add(mc, 31), not(31)))
288  }
289  //if we want a zero-length slice let's just return a zero-length array
290  default {
291      tempBytes := mload(0x40)
292
293      mstore(0x40, add(tempBytes, 0x20))
294  }
295
296  return tempBytes;
297 }
298 //@CTK NO_BUF_OVERFLOW
299 //@CTK NO_ASF
300 function toAddress(bytes memory _bytes, uint _start) internal pure returns (address) {
301     require(_bytes.length >= (_start + 20), "_bytes.length >= (_start + 20)");
302     address tempAddress;
303
304     assembly {
305         tempAddress := div(mload(add(add(_bytes, 0x20), _start)), 0
306             x100000000000000000000000000000000)
307     }
308
309     return tempAddress;
310 }
311 //@CTK NO_BUF_OVERFLOW
312 //@CTK NO_ASF
313 function toUint8(bytes memory _bytes, uint _start) internal pure returns (uint8) {
314     require(_bytes.length >= (_start + 1), "_bytes.length >= (_start + 1)");
315     uint8 tempUint;

```

```

315
316     assembly {
317         tempUint := mload(add(add(_bytes, 0x1), _start))
318     }
319
320     return tempUint;
321 }
322 //@CTK NO_BUF_OVERFLOW
323 //@CTK NO_ASF
324 function toUint16(bytes memory _bytes, uint _start) internal pure returns (uint16) {
325     require(_bytes.length >= (_start + 2), "_bytes.length >= (_start + 2)");
326     uint16 tempUint;
327
328     assembly {
329         tempUint := mload(add(add(_bytes, 0x2), _start))
330     }
331
332     return tempUint;
333 }
334 //@CTK NO_BUF_OVERFLOW
335 //@CTK NO_ASF
336 function toUint32(bytes memory _bytes, uint _start) internal pure returns (uint32) {
337     require(_bytes.length >= (_start + 4), "_bytes.length >= (_start + 4)");
338     uint32 tempUint;
339
340     assembly {
341         tempUint := mload(add(add(_bytes, 0x4), _start))
342     }
343
344     return tempUint;
345 }
346 //@CTK NO_BUF_OVERFLOW
347 //@CTK NO_ASF
348 function toUint64(bytes memory _bytes, uint _start) internal pure returns (uint64) {
349     require(_bytes.length >= (_start + 8), "_bytes.length >= (_start + 8)");
350     uint64 tempUint;
351
352     assembly {
353         tempUint := mload(add(add(_bytes, 0x8), _start))
354     }
355
356     return tempUint;
357 }
358 //@CTK NO_BUF_OVERFLOW
359 //@CTK NO_ASF
360 function toUint96(bytes memory _bytes, uint _start) internal pure returns (uint96) {
361     require(_bytes.length >= (_start + 12), "_bytes.length >= (_start + 12)");
362     uint96 tempUint;
363
364     assembly {
365         tempUint := mload(add(add(_bytes, 0xc), _start))
366     }
367
368     return tempUint;
369 }
370 //@CTK NO_BUF_OVERFLOW
371 //@CTK NO_ASF
372 function toUint128(bytes memory _bytes, uint _start) internal pure returns (uint128) {

```



```

373     require(_bytes.length >= (_start + 16), "_bytes.length >= (_start + 16)");
374     uint128 tempUint;
375
376     assembly {
377         tempUint := mload(add(add(_bytes, 0x10), _start))
378     }
379
380     return tempUint;
381 }
382 //@CTK NO_BUF_OVERFLOW
383 //@CTK NO_ASF
384 function toUint(bytes memory _bytes, uint _start) internal pure returns (uint256) {
385     require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
386     uint256 tempUint;
387
388     assembly {
389         tempUint := mload(add(add(_bytes, 0x20), _start))
390     }
391
392     return tempUint;
393 }
394 //@CTK NO_BUF_OVERFLOW
395 //@CTK NO_ASF
396 function toBytes32(bytes memory _bytes, uint _start) internal pure returns (bytes32) {
397     require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
398     bytes32 tempBytes32;
399
400     assembly {
401         tempBytes32 := mload(add(add(_bytes, 0x20), _start))
402     }
403
404     return tempBytes32;
405 }
406 //@CTK NO_BUF_OVERFLOW
407 //@CTK NO_OVERFLOW
408 //@CTK NO_ASF
409 function equal(bytes memory _preBytes, bytes memory _postBytes) internal pure returns (
410     bool) {
411     bool success = true;
412
413     assembly {
414         let length := mload(_preBytes)
415
416         // if lengths don't match the arrays are not equal
417         switch eq(length, mload(_postBytes))
418         case 1 {
419             // cb is a circuit breaker in the for loop since there's
420             // no said feature for inline assembly loops
421             // cb = 1 - don't breaker
422             // cb = 0 - break
423             let cb := 1
424
425             let mc := add(_preBytes, 0x20)
426             let end := add(mc, length)
427
428             for {
429                 let cc := add(_postBytes, 0x20)
430                 // the next line is the loop condition:

```

```

430         // while(uint(mc < end) + cb == 2)
431     } eq(add(lt(mc, end), cb), 2) {
432         mc := add(mc, 0x20)
433         cc := add(cc, 0x20)
434     } {
435         // if any of these checks fails then arrays are not equal
436         if iszero(eq(mload(mc), mload(cc))) {
437             // unsuccessful:
438             success := 0
439             cb := 0
440         }
441     }
442 }
443 default {
444     // unsuccessful:
445     success := 0
446 }
447 }
448
449 return success;
450 }
451 //@CTK NO_BUF_OVERFLOW
452 //@CTK NO_OVERFLOW
453 //@CTK NO_ASF
454 function equalStorage(
455     bytes storage _preBytes,
456     bytes memory _postBytes
457 )
458     internal
459     view
460     returns (bool)
461 {
462     bool success = true;
463
464     assembly {
465         // we know _preBytes_offset is 0
466         let fslot := sload(_preBytes_slot)
467         // Decode the length of the stored array like in concatStorage().
468         let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
469         let mlength := mload(_postBytes)
470
471         // if lengths don't match the arrays are not equal
472         switch eq(slength, mlength)
473         case 1 {
474             // slength can contain both the length and contents of the array
475             // if length < 32 bytes so let's prepare for that
476             // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-state-variables-in-storage
477             if iszero(iszero(slength)) {
478                 switch lt(slength, 32)
479                 case 1 {
480                     // blank the last byte which is the length
481                     fslot := mul(div(fslot, 0x100), 0x100)
482
483                     if iszero(eq(fslot, mload(add(_postBytes, 0x20)))) {
484                         // unsuccessful:
485                         success := 0
486                     }

```

```

487     }
488     default {
489         // cb is a circuit breaker in the for loop since there's
490         // no said feature for inline assembly loops
491         // cb = 1 - don't breaker
492         // cb = 0 - break
493         let cb := 1
494
495         // get the keccak hash to get the contents of the array
496         mstore(0x0, _preBytes_slot)
497         let sc := keccak256(0x0, 0x20)
498
499         let mc := add(_postBytes, 0x20)
500         let end := add(mc, mlength)
501
502         // the next line is the loop condition:
503         // while(uint(mc < end) + cb == 2)
504         for {} eq(add(lt(mc, end), cb), 2) {
505             sc := add(sc, 1)
506             mc := add(mc, 0x20)
507         } {
508             if iszero(eq(sload(sc), mload(mc))) {
509                 // unsuccessful:
510                 success := 0
511                 cb := 0
512             }
513         }
514     }
515 }
516 }
517 default {
518     // unsuccessful:
519     success := 0
520 }
521 }
522
523 return success;
524 }
525 }

```

File math/SafeMath.sol

```

1  pragma solidity ^0.5.13;
2
3  /**
4   * @dev Wrappers over Solidity's arithmetic operations with added overflow
5   * checks.
6   *
7   * Arithmetic operations in Solidity wrap on overflow. This can easily result
8   * in bugs, because programmers usually assume that an overflow raises an
9   * error, which is the standard behavior in high level programming languages.
10  * `SafeMath` restores this intuition by reverting the transaction when an
11  * operation overflows.
12  *
13  * Using this library instead of the unchecked operations eliminates an entire
14  * class of bugs, so it's recommended to use it always.
15  */
16  library SafeMath {
17      /**

```

```

18  * @dev Returns the addition of two unsigned integers, reverting on
19  * overflow.
20  *
21  * Counterpart to Solidity's `+` operator.
22  *
23  * Requirements:
24  * - Addition cannot overflow.
25  */
26  /*@CTK "SafeMath add"
27  @post (a + b < a || a + b < b) == __reverted
28  @post !__reverted -> __return == a + b
29  @post !__reverted -> !__has_overflow
30  @post !__reverted -> !__has_assertion_failure
31  @post !(__has_buf_overflow)
32  */
33  function add(uint256 a, uint256 b) internal pure returns (uint256) {
34      uint256 c = a + b;
35      require(c >= a, "SafeMath: addition overflow");
36
37      return c;
38  }
39
40  /**
41  * @dev Returns the subtraction of two unsigned integers, reverting on
42  * overflow (when the result is negative).
43  *
44  * Counterpart to Solidity's `-` operator.
45  *
46  * Requirements:
47  * - Subtraction cannot overflow.
48  */
49  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
50      return sub(a, b, "SafeMath: subtraction overflow");
51  }
52
53  /**
54  * @dev Returns the subtraction of two unsigned integers, reverting with custom message
55  * on
56  * overflow (when the result is negative).
57  *
58  * Counterpart to Solidity's `-` operator.
59  *
60  * Requirements:
61  * - Subtraction cannot overflow.
62  * _Available since v2.4.0._
63  */
64  /*@CTK "SafeMath sub"
65  @post (a < b) == __reverted
66  @post !__reverted -> __return == a - b
67  @post !__reverted -> !__has_overflow
68  @post !__reverted -> !__has_assertion_failure
69  @post !(__has_buf_overflow)
70  */
71  function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
72      uint256) {
73      require(b <= a, errorMessage);
74      uint256 c = a - b;

```

```

74
75     return c;
76 }
77
78 /**
79  * @dev Returns the multiplication of two unsigned integers, reverting on
80  * overflow.
81  *
82  * Counterpart to Solidity's `*` operator.
83  *
84  * Requirements:
85  * - Multiplication cannot overflow.
86  */
87 /*@CTK "SafeMath mul"
88  @post (((a) > (0)) && (((a) * (b)) / (a)) != (b))) == (__reverted)
89  @post !__reverted -> __return == a * b
90  @post !__reverted == !__has_overflow
91  @post !__reverted -> !__has_assertion_failure
92  @post !(__has_buf_overflow)
93  */
94 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
95     // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
96     // benefit is lost if 'b' is also tested.
97     // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
98     if (a == 0) {
99         return 0;
100     }
101
102     uint256 c = a * b;
103     require(c / a == b, "SafeMath: multiplication overflow");
104
105     return c;
106 }
107
108 /**
109  * @dev Returns the integer division of two unsigned integers. Reverts on
110  * division by zero. The result is rounded towards zero.
111  *
112  * Counterpart to Solidity's `/` operator. Note: this function uses a
113  * `revert` opcode (which leaves remaining gas untouched) while Solidity
114  * uses an invalid opcode to revert (consuming all remaining gas).
115  *
116  * Requirements:
117  * - The divisor cannot be zero.
118  */
119 function div(uint256 a, uint256 b) internal pure returns (uint256) {
120     return div(a, b, "SafeMath: division by zero");
121 }
122
123 /**
124  * @dev Returns the integer division of two unsigned integers. Reverts with custom
125     message on
126     division by zero. The result is rounded towards zero.
127  *
128  * Counterpart to Solidity's `/` operator. Note: this function uses a
129  * `revert` opcode (which leaves remaining gas untouched) while Solidity
130  * uses an invalid opcode to revert (consuming all remaining gas).
131  */

```

```

131  * Requirements:
132  * - The divisor cannot be zero.
133  *
134  * _Available since v2.4.0._
135  */
136  /*@CTK "SafeMath div"
137  @post (b <= 0) == __reverted
138  @post !__reverted -> __return == a / b
139  @post !__reverted -> !__has_overflow
140  @post !__reverted -> !__has_assertion_failure
141  @post !(__has_buf_overflow)
142  */
143  function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
144      uint256) {
145      // Solidity only automatically asserts when dividing by 0
146      require(b > 0, errorMessage);
147      uint256 c = a / b;
148      // assert(a == b * c + a % b); // There is no case in which this doesn't hold
149
150      return c;
151  }
152
153  /**
154   * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
155   * modulo),
156   * Reverts when dividing by zero.
157   * Counterpart to Solidity's `%` operator. This function uses a `revert`
158   * opcode (which leaves remaining gas untouched) while Solidity uses an
159   * invalid opcode to revert (consuming all remaining gas).
160   *
161   * Requirements:
162   * - The divisor cannot be zero.
163   */
164  function mod(uint256 a, uint256 b) internal pure returns (uint256) {
165      return mod(a, b, "SafeMath: modulo by zero");
166  }
167
168  /**
169   * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
170   * modulo),
171   * Reverts with custom message when dividing by zero.
172   * Counterpart to Solidity's `%` operator. This function uses a `revert`
173   * opcode (which leaves remaining gas untouched) while Solidity uses an
174   * invalid opcode to revert (consuming all remaining gas).
175   *
176   * Requirements:
177   * - The divisor cannot be zero.
178   *
179   * _Available since v2.4.0._
180   */
181  /*@CTK "SafeMath mod"
182  @post (b == 0) == __reverted
183  @post !__reverted -> __return == a % b
184  @post !__reverted -> !__has_overflow
185  @post !__reverted -> !__has_assertion_failure
186  @post !(__has_buf_overflow)

```

```
186  */
187  function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
188      uint256) {
189      require(b != 0, errorMessage);
189      return a % b;
190  }
191 }
```

