CERTIK

Security Assessment

BORA

CertiK Verified on Mar 23rd, 2023

CertiK Verified on Mar 23rd, 2023

BORA

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Others	Ethereum	Formal Verification, Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 03/23/2023	N/A

CODEBASE

Private Codebase

https://github.com/boraecosystem/bora-token-v2

...View All

COMMITS

0e68a7c4ee88155392b15b13e977de00039aa2cf

...View All

Vulnerability Summary

8	7	0	0	1	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
2	Major	1 Resolved, 1 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
3	Minor	3 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
1	Informational	1 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | BORA

CODEBASE | BORA

Repository

Private Codebase

https://github.com/boraecosystem/bora-token-v2

Commit

0e68a7c4ee88155392b15b13e977de00039aa2cf

AUDIT SCOPE | BORA

2 files audited ● 1 file with Acknowledged findings ● 1 file with Resolved findings

ID	File	SHA256 Checksum
● BBO	contracts/Bora20v2.sol	40dfff7ee82c01a4dd2eb7cf7fbbcf17f09930a2948a3a0ae277f64a32990899
● BBR	contracts/Bora721v2.sol	83cdc279fbbf6f043e6c5db8852326ca62cdf289f6214bd0f4f6f9f3180a7f2d

APPROACH & METHODS | BORA

This report has been prepared for BORA to discover issues and vulnerabilities in the source code of the BORA project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis 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.

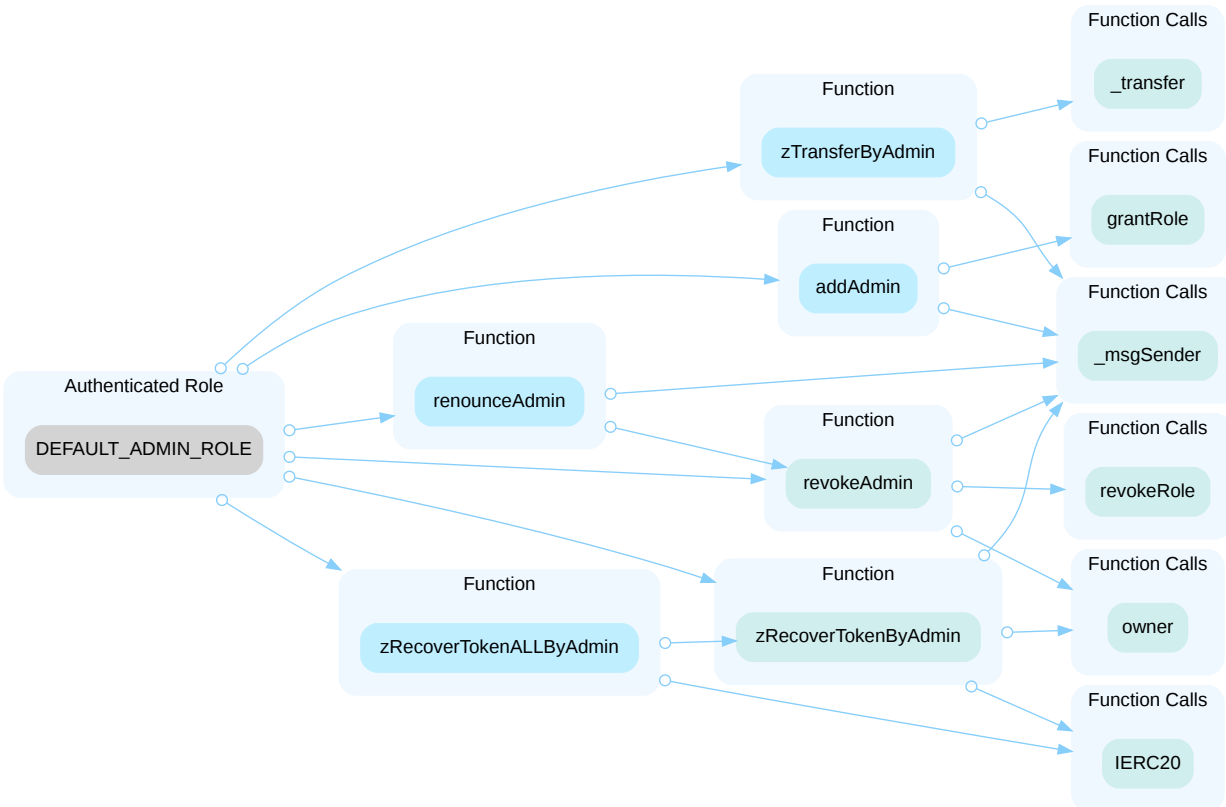
The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

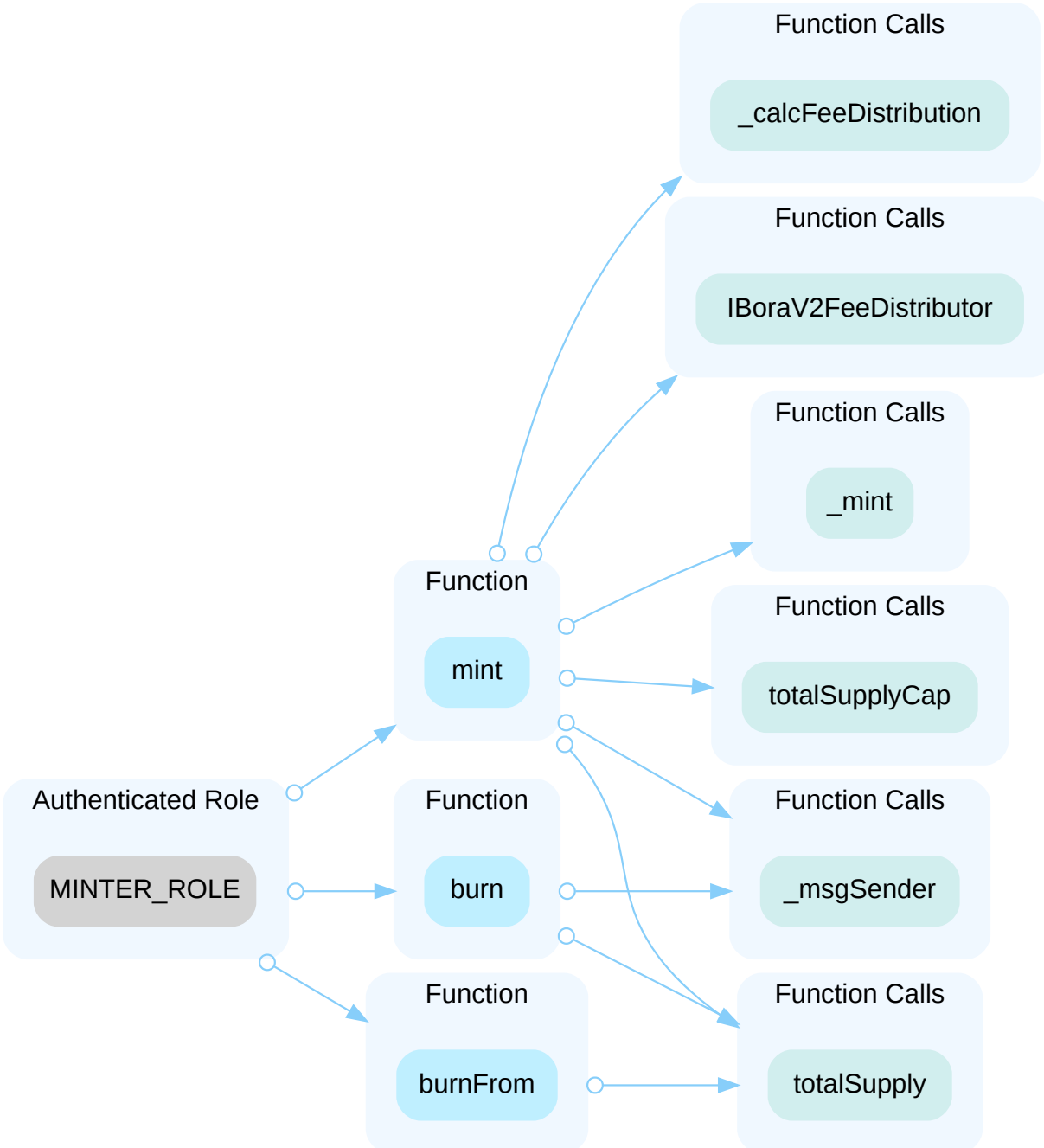
DECENTRALIZATION EFFORTS | BORA

Description

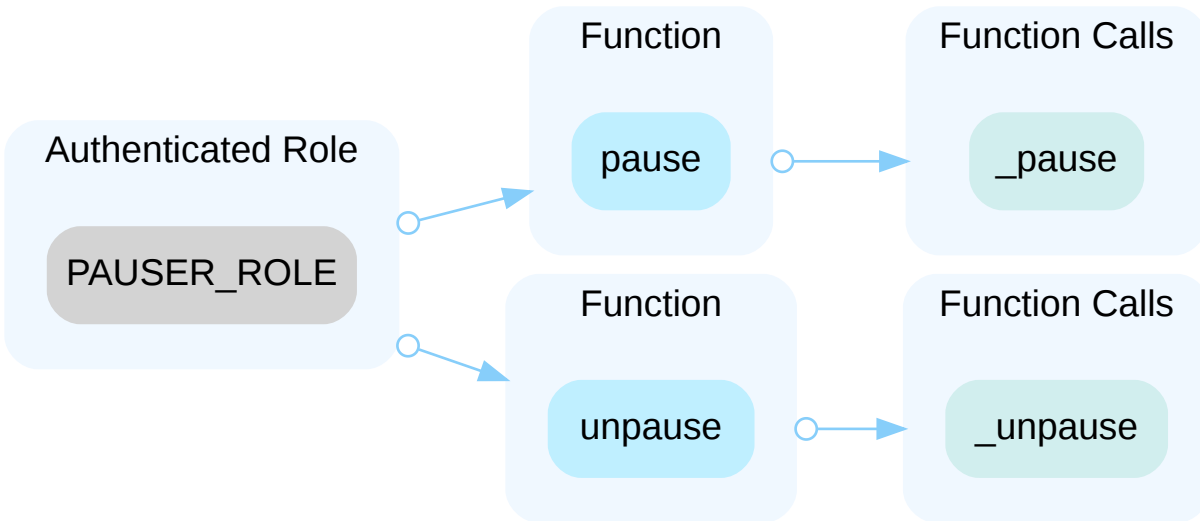
In the contract `Bora20v2` the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below.



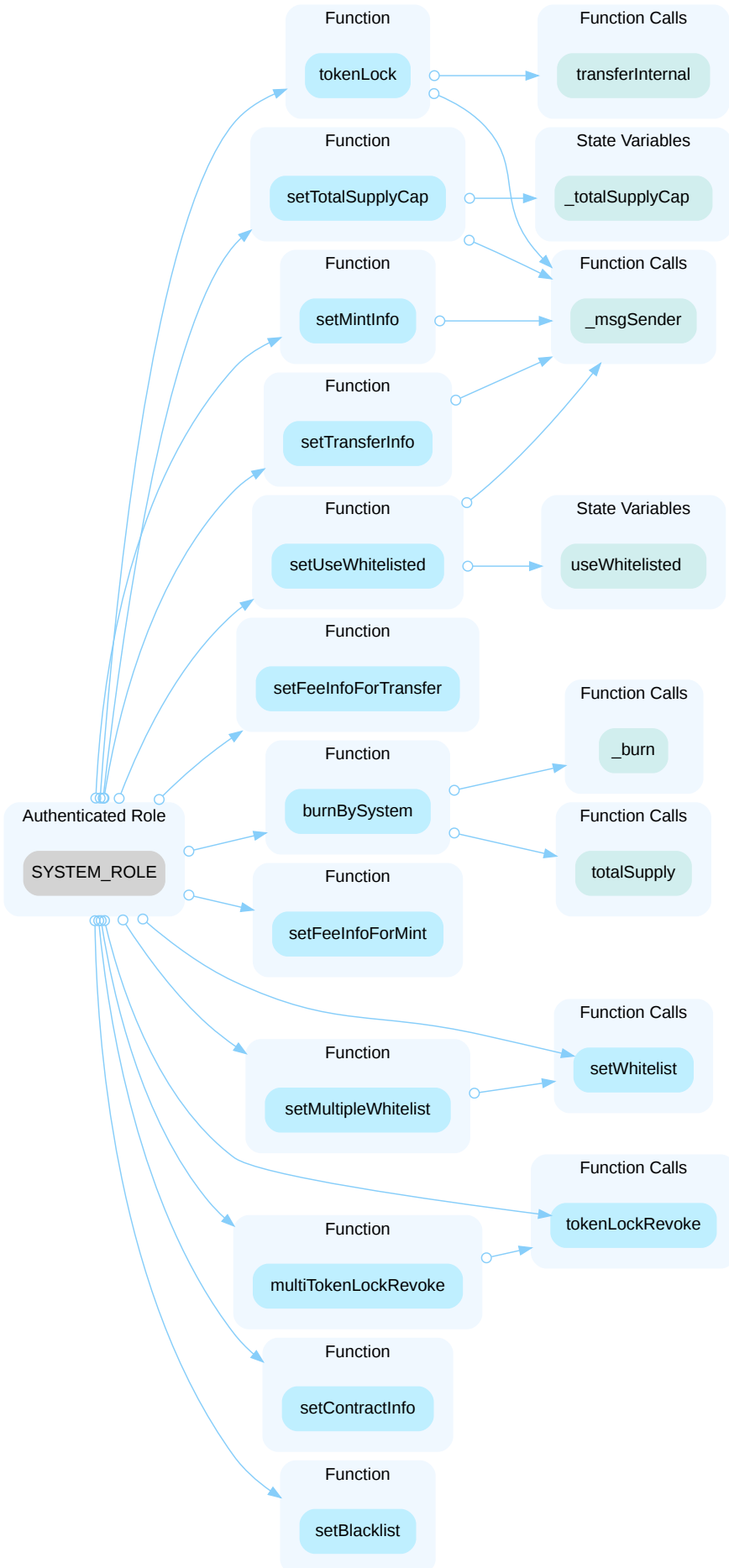
In the contract `Bora20v2` the role `MINTER_ROLE` has authority over the functions shown in the diagram below.



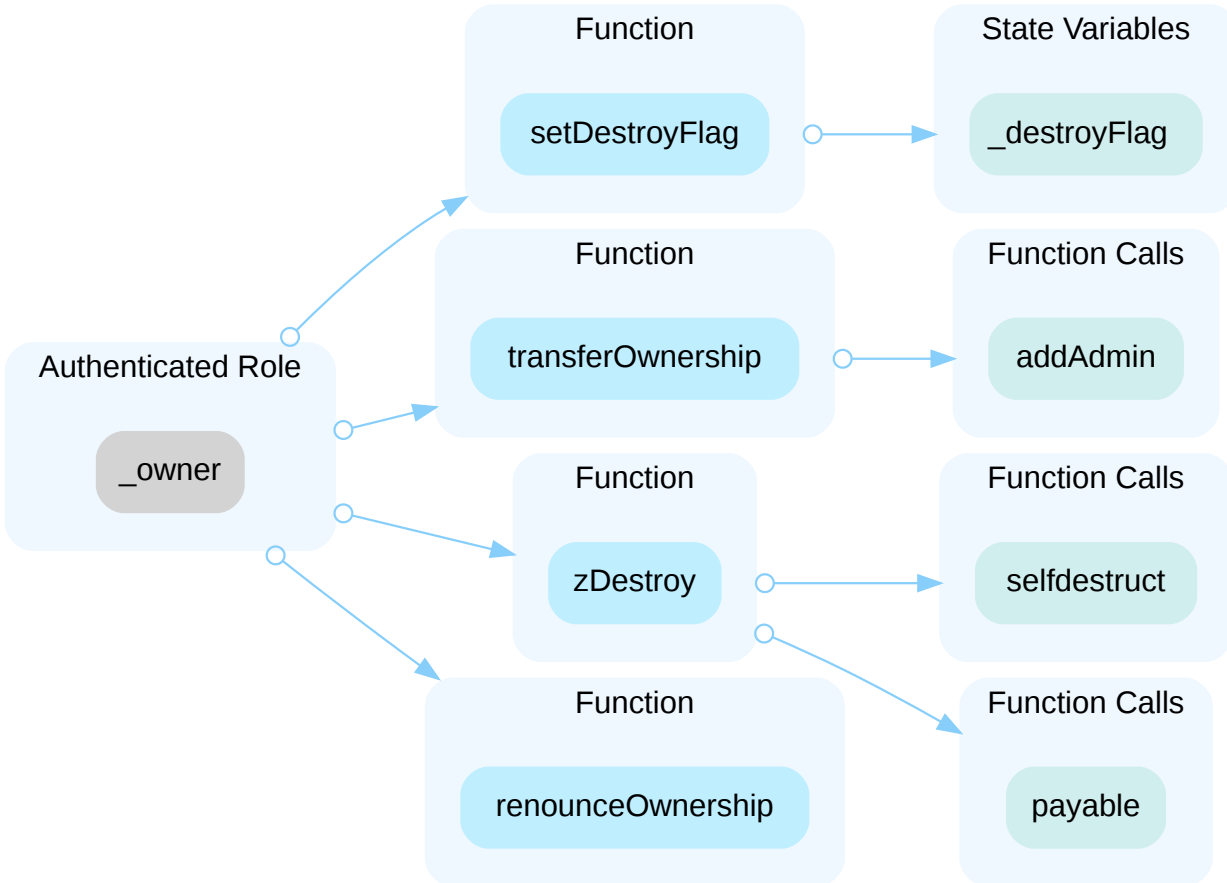
In the contract `Bora20v2` the role `PAUSER_ROLE` has authority over the functions shown in the diagram below.



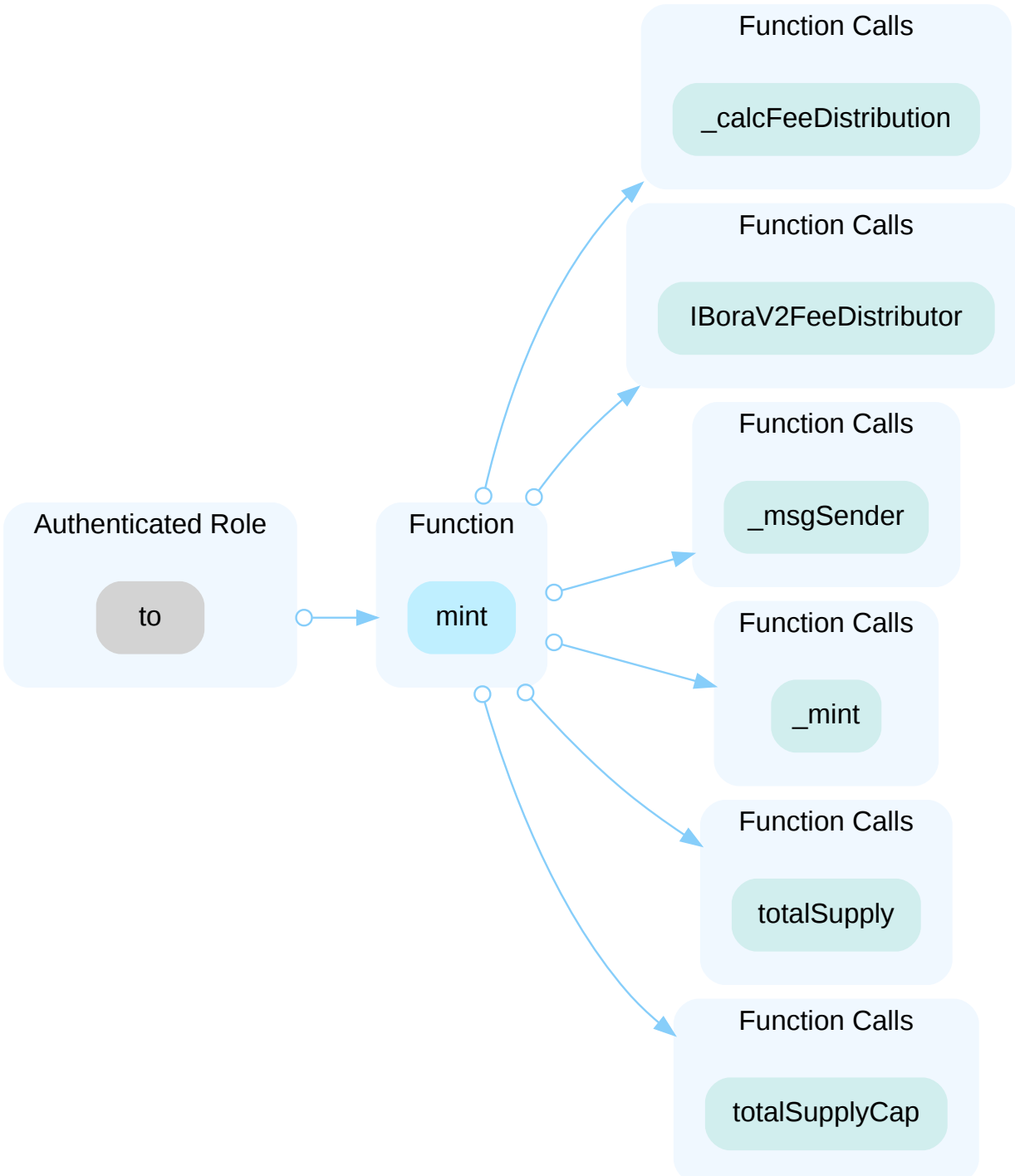
In the contract `Bora20v2` the role `SYSTEM_ROLE` has authority over the functions shown in the diagram below.


Function
tokenLock
Function Calls
transferInternal
Function
setTotalSupplyCap
State Variables
_totalSupplyCap
Function
setMintInfo
Function Calls
_msgSender
Function
setTransferInfo
Function
setUseWhitelisted
State Variables
useWhitelisted
Function
setFeeInfoForTransfer
Function
burnBySystem
Function Calls
_burn
Function Calls
totalSupply
Authenticated Role
SYSTEM_ROLE
Function
setFeeInfoForMint
Function Calls
setWhitelist
Function
setMultipleWhitelist
Function Calls
tokenLockRevoke
Function
multiTokenLockRevoke
Function
setContractInfo
Function
setBlacklist

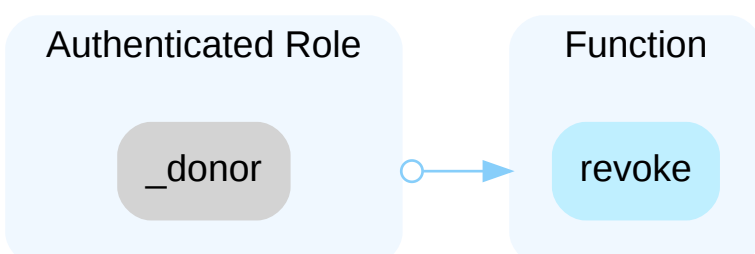
In the contract `Bora20v2` the role `_owner` has authority over the functions shown in the diagram below.



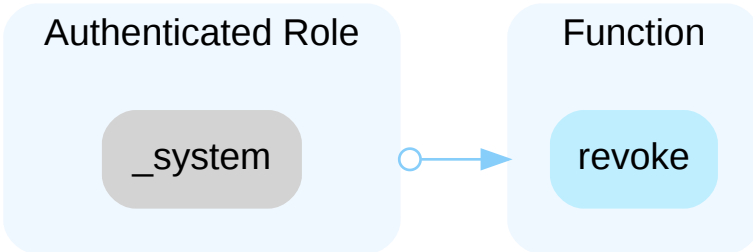
In the contract `Bora20v2` the role `to` has authority over the functions shown in the diagram below.



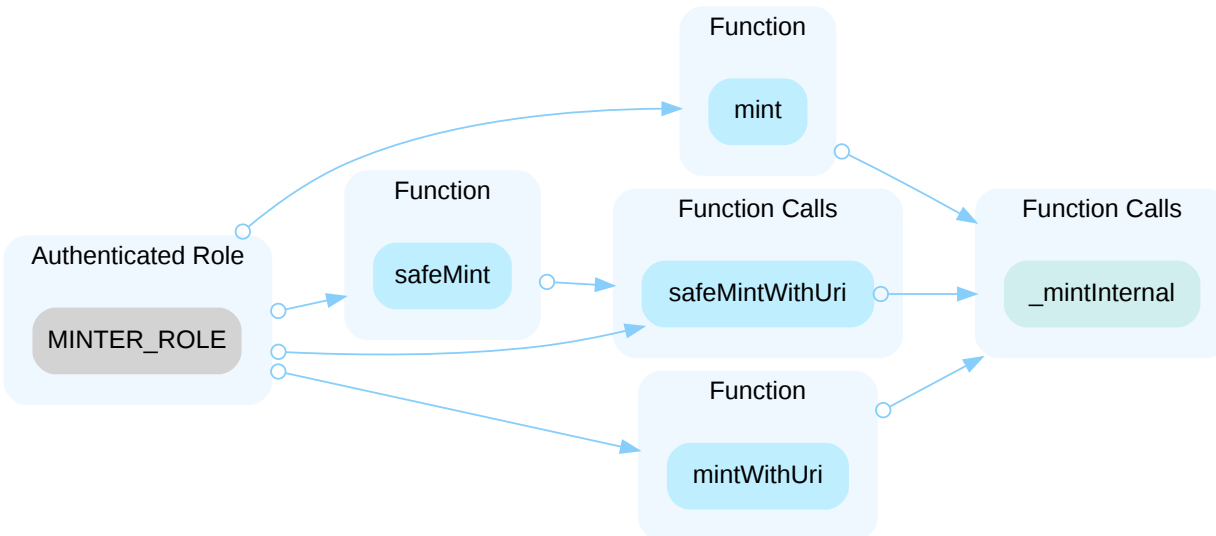
In the contract `LockedToken` the role `_donor` has authority over the functions shown in the diagram below.



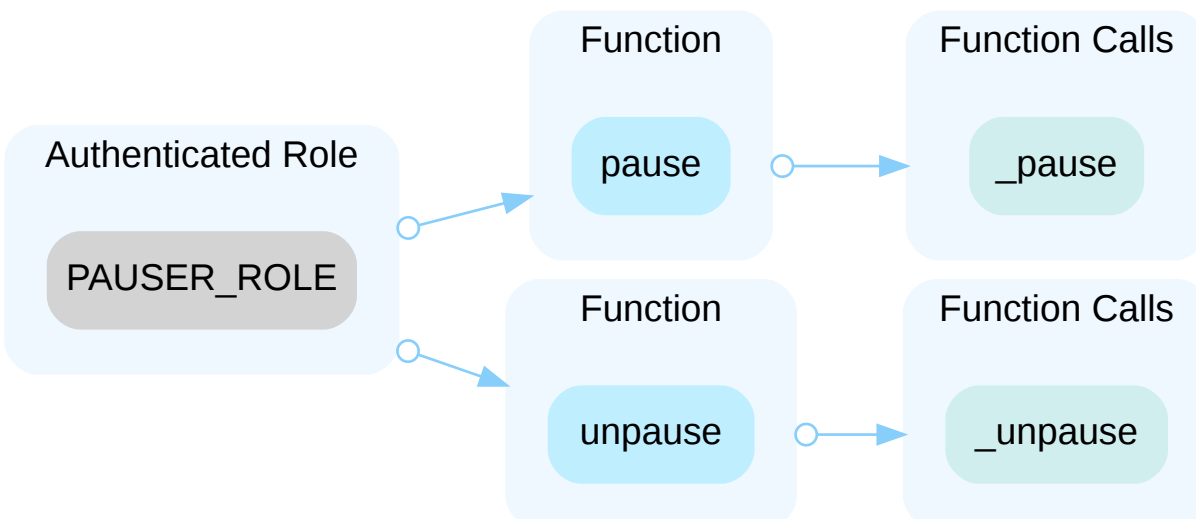
In the contract `LockedToken` the role `_system` has authority over the functions shown in the diagram below.



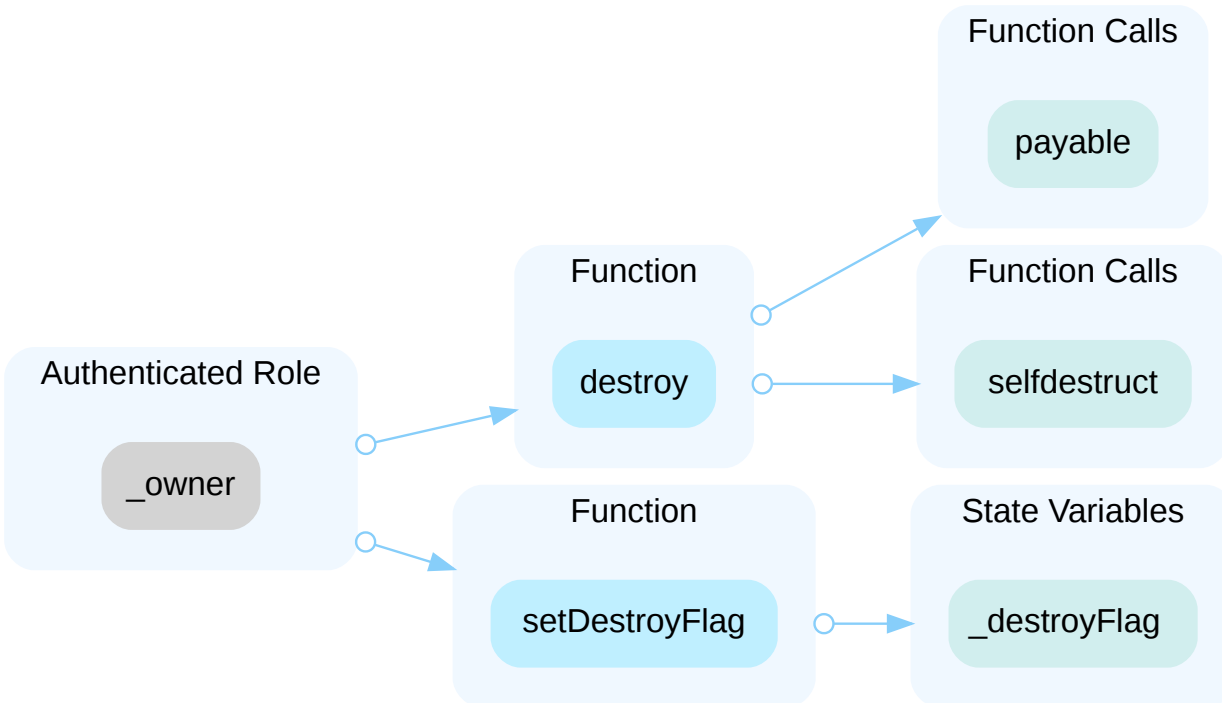
In the contract `Bora721v2` the role `MINTER_ROLE` has authority over the functions shown in the diagram below.



In the contract `Bora721v2` the role `PAUSER_ROLE` has authority over the functions shown in the diagram below.



In the contract `Bora721v2` the role `_owner` has authority over the functions shown in the diagram below.



In the contract `Bora721v2`, `SYSTEM_ROLE`, `DEFAULT_ADMIN_ROLE` and `owner` in `onlySystem` modifier have authority over the functions shown below.

- `setContractURI()`
- `setBaseURI()`
- `setTokenURI(uint256 tokenId, string memory _tokenURI)`
- `setTokenURI(uint256 tokenId, string memory _tokenURI, uint256 logNo)`
- `removeTokenURI()`
- `setTokenURISuffix()`
- `setRevealed()`
- `setNotRevealURI()`
- `setNotRevealedTokenIdScope()`
- `setBlacklist()`
- `setMultipleBlacklist()`
- `setUseWhitelisted()`
- `setWhitelist()`
- `setMultipleWhitelist()`
- `setPublicMintEnabled()`
- `setPublicMintConfig()`
- `setMaxCountTokensOfOwner()`
- `changeTokenLockedState()`
- `withdraw()`

- `withdrawToken()`

In the contract `Bora721v2`, `DEFAULT_ADMIN_ROLE` and `owner` in `onlyOwnerOrAdmin` modifier have authority over the functions shown below.

- `addAdmin()`
- `renounceAdmin()`
- `revokeAdmin()`
- `removeAdmin()`

Any compromise to the privileged account may allow the hacker to take advantage of this authority.

Recommendations

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term: Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term: Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent: Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.

  OR

- Remove the risky functionality.

THIRD-PARTY DEPENDENCY | BORA

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

```
270  IBoraV2FeeDistributor(_feeDist.feeReceiver).distributeFees();
```

- The function `Bora20v2.mint` interacts with third party contract with `IBoraV2FeeDistributor` interface.

```
236  IBoraV2FeeDistributor(_feeDist.feeReceiver).distributeFees();
```

- The function `transferInternal` interacts with third party contract with `IBoraV2FeeDistributor` interface.

```
323  (feeDist.amount, feeDist.fee, feeDist.feeReceiver, feeDist.isDistributor) =
IBoraV2FeeCalculate(feeInfo.feeCalcAddress).calcAmount(amount, from, to,
feeInfo.feeCalcCallBytes);
```

- The function `_calcFeeDistribution` interacts with third party contract with `IBoraV2FeeCalculate` interface.

Recommendations

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

FINDINGS | BORA

8	0	2	2	3	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for BORA. Through this audit, we have uncovered 8 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
BBO-01	Anyone Can Call `claim()` To Withdraw All Tokens To `_beneficiary` Address	Control Flow	Major	Resolved
BBO-02	**Initial Token Distribution**	**Centralization / Privilege**	**Major**	**Acknowledged**
BBR-01	Function `isApprovedForAll` Of Contract `Bora721v2` Returns Incorrect Approvals	Logical Issue	Medium	Resolved
BBR-02	Bot Interval Validation Doe Not Work	Logical Issue	Medium	Resolved
BBO-04	Unused Value	Inconsistency	Minor	Resolved
BBO-10	Unused Return Value	Volatile Code	Minor	Resolved
BBO-11	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
BBO-05	Redundant Statements	Volatile Code	Informational	Resolved

BBO-01 ANYONE CAN CALL `claim()` TO WITHDRAW ALL TOKENS TO `_beneficiary` ADDRESS

Category	Severity	Location	Status
Control Flow	Major	contracts/Bora20v2.sol: 92, 447, 452	Resolved

Description

```
function claim() public returns (bool) {
    require(block.timestamp >= _releaseTime, "LockedToken: current time is before release time");

    uint256 amount = _token.balanceOf(address(this));
    require(amount > 0, "LockedToken: no tokens to claim");

    _token.safeTransfer(_beneficiary, amount);
    emit Claim(_beneficiary, amount, _releaseTime);
    return true;
}
```

```
function tokenLockClaim(LockedToken _lockToken) public returns (bool) {
    _lockToken.claim();
    return true;
}
```

```
function multiTokenLockClaim(LockedToken[] memory _lockToken) external returns (bool) {
    for (uint256 i = 0; i < _lockToken.length; i++) {
        tokenLockClaim(_lockToken[i]);
    }
    return true;
}
```

In the functions `multiTokenLockClaim()`, `tokenLockClaim()`, and `claim()`, any caller can call these functions to transfer all the tokens will be transferred to `_beneficiary` address. As the logic of `claim()` is similar to `revoke(), a specific user validation should be added to` claim()`

Recommendation

We recommend the team adding a validation to make sure only a certain group of user can call the `claim()` function

Alleviation

`[BORA]` : This is intended to be called by anyone after the release time, because the beneficiary is already claimed and cannot be changed.

BBO-02 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/Bora20v2.sol: 177**	**Acknowledged**

Description

All **Bora20v2** tokens are sent to the contract deployer when deploying the contract. This is a potential centralization risk as the deployer can distribute **Bora20v2** tokens without the consensus of the community.

Recommendation

We recommend transparency through providing a breakdown of the intended initial token distribution in a public location. We also recommend the team make an effort to restrict the access of the corresponding private key.

Alleviation

`[BORA]` : The initial supply is zero, and all minted token's will be published on a public page explaining their purpose and process. Minter uses a multisig wallet, and the private keys are strictly kept in an isolated location, accessible only to those with certain privileges.

BBR-01 FUNCTION `isApprovedForAll` OF CONTRACT `Bora721v2` RETURNS INCORRECT APPROVALS

Category	Severity	Location	Status
Logical Issue	Medium	contracts/Bora721v2.sol: 360~366	Resolved

Description

It is expected that calls of the form `isApprovedForAll(owner, operator)` return whether a non-zero address `operator` is approved for tokens of a non-zero address `owner`, or return false. However, the implementation of `isApprovedForAll` in contract `Bora721v2` returns incorrect approvals in some cases.

Recommendation

Ensure that the Boolean values returned for calls of `isApprovedForAll` are in accordance with the entries in the contract's approve mapping.

Alleviation

`[BORA]` : The system account is granted the approve permission to respond to cases where NFTs are unfairly taken due to issues such as hacking or fraud.

BBR-02 | BOT INTERVAL VALIDATION DOE NOT WORK

Category	Severity	Location	Status
Logical Issue	Medium	contracts/Bora721v2.sol: 273~275, 480, 484	Resolved

Description

In the function `publicMint()`, `publicMintWithUri()`, and `_mintInternal()`, the `checkBotInterval` and `publicMintInfo._lastCallBlockNumber[msgSender]` may fail to prevent the bot from minting tokens within 1 block.

```
480 function publicMint(uint256 tokenId, uint256 kind) public payable checkWhitelisted(msg.sender) checkPublicMintCondition(msg.sender, kind, msg.value) returns (bool) {
481     return publicMintWithUri(tokenId, kind, "");
482 }
```

```
484 function publicMintWithUri(uint256 tokenId, uint256 kind, string memory uri) public payable checkWhitelisted(msg.sender) checkPublicMintCondition(msg.sender, kind, msg.value) returns (bool) {
485     return _mintInternal(msg.sender, tokenId, msg.sender, true, uri);
486 }
```

```
263 function _mintInternal(address to, uint256 tokenId, address msgSender, bool checkBotInterval, string memory uri) internal checkWhitelisted(to) returns (bool) {
264     if (bytes(uri).length == 0) {
265         _safeMint(to, tokenId);
266     } else {
267         _safeMint(to, tokenId);
268         _setTokenURI(tokenId, uri);
269     }
270
271     incrementMintCount();
272
273     if (checkBotInterval) {
274         publicMintInfo._lastCallBlockNumber[msgSender] = block.number;
275     }
276     emit Mint(to, tokenId, msgSender);
277     return true;
278 }
```

Scenario

1. Bob deployed a contract to call `publicMint()` or `publicMintWithUri()` to mint a ERC721 token with `msg.value`
2. `_safeMint()` in `_mintInternal()` will be invoked to mint this token
3. `_checkOnERC721Received()` in `_safeMint()` will be called once Bob's contract received the minted token. Implementation can be found in https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol#L247
4. `onERC721Received()` hook in `_checkOnERC721Received()` will be invoked and create a reentrancy chance to call any function in Bob's contract. The hook implementation can be found in https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol#L406
5. Bob's contract can then call the `publicMint()` or `publicMintWithUri()` to mint the other ERC721 token with a new `msg.value`
6. At this time, the `publicMintInfo._lastCallBlockNumber[msgSender]` in `_mintInternal` haven't been updated yet because the invocation of `_safeMint()` happens before the `publicMintInfo._lastCallBlockNumber` update in step 2.
7. So the `checkPublicMintCondition` modifier failed to prevent a bot contract to mint multiple tokens in 1 transaction.
8. Moreover, `totalMintCount` will be inaccurate because `incrementMintCount()` will not be correctly invoked in `_mintInternal()` neither.

Recommendation

We recommend the team consider adding `nonReentrant` modifier to the `_mintInternal()` function. Reference Link

Alleviation

`[BORA]` : Issue acknowledged. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

BBO-04 | UNUSED VALUE

Category	Severity	Location	Status
Inconsistency	Minor	contracts/Bora20v2.sol: 407	Resolved

Description

```
407         (_diffDay, _diffTime) = utilDiffTime(_lockToken.releaseTime());
```

- The variable `_diffTime` is never used after this assignment.

Recommendation

We recommend reviewing those unused values for omission of their usage or considering to remove their definitions.

Alleviation

`[BORA]` : Issue acknowledged. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

BBO-10 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/Bora20v2.sol: 236, 270, 448, 460	Resolved

Description

The return value of an external call is not stored in a local or state variable.

```
236
IBoraV2FeeDistributor(_feeDist.feeReceiver).distributeFees();
```

```
270
IBoraV2FeeDistributor(_feeDist.feeReceiver).distributeFees();
```

```
448         _lockToken.claim();
```

```
460         _lockToken.revoke();
```

Recommendation

We recommend checking or using the return values of all external function calls.

Alleviation

`[BORA]` : The return value is removed from `LockedToken.claim()` , `LockedToken.revoke()` , and its related functions. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

BBO-11 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	contracts/Bora20v2.sol: 546	Resolved

Description

The return value of the transfer()/transferFrom() call is not checked.

```
546         IERC20(_tokenAddress).transfer(owner(), _tokenAmount);
```

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We advise using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Alleviation

`[BORA]` : Issue acknowledged. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

BBO-05 | REDUNDANT STATEMENTS

Category	Severity	Location	Status
Volatile Code	Informational	contracts/Bora20v2.sol: 413, 421, 426, 434	Resolved

Description

The linked statements do not affect the functionality of the codebase and appear to be either remnants of test code or older functionality.

```
413 function multiTransfers(address[] memory recipients, uint256[] memory amount) public returns (bool) {
414     require(recipients.length == amount.length, "BORA: Input arrays must be the same length");
415     for (uint256 i = 0; i < recipients.length; i++) {
416         transfer(recipients[i], amount[i]);
417     }
418     return true;
419 }
420
421 function multiTransfers(address[] memory recipients, uint256[] memory amount, uint256 logNo) external returns (bool) {
422     require(logNo >= 0); // This is meaningless code to get rid of the warning
423     return multiTransfers(recipients, amount);
424 }
```

Similarly, there are two `multiTransferFroms()` functions with different internfaces but same logic.

```
426 function multiTransferFroms(address[] memory senders, address[] memory recipients, uint256[] memory amount) public returns (bool) {
427     require(senders.length == recipients.length && recipients.length == amount.length, "BORA: Input arrays must be the same length");
428     for (uint256 i = 0; i < senders.length; i++) {
429         transferFrom(senders[i], recipients[i], amount[i]);
430     }
431     return true;
432 }
433
434 function multiTransferFroms(address[] memory senders, address[] memory recipients, uint256[] memory amount, uint256 logNo) external returns (bool) {
435     require(logNo >= 0); // This is meaningless code to get rid of the warning
436     return multiTransferFroms(senders, recipients, amount);
437 }
```

Recommendation

We recommend merging the functions that have same logic to save gas

Alleviation

`[BORA]` : Issue acknowledged. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

OPTIMIZATIONS | BORA

ID	Title	Category	Severity	Status
BBO-06	Unnecessary Use Of SafeMath	Gas Optimization	Optimization	Resolved
BBO-07	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
BBO-08	User-Defined Getters	Gas Optimization	Optimization	Resolved
BBO-09	Unnecessary Validation	Gas Optimization	Optimization	Resolved
BBO-12	Tautology Or Contradiction	Gas Optimization	Optimization	Resolved

BBO-06 | UNNECESSARY USE OF SAFEMATH

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/Bora20v2.sol: 333, 334, 340, 345, 388, 389, 390, 395, 396, 397	Resolved

Description

The `SafeMath` library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations will automatically revert in case of integer overflow or underflow.

```
105     using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `Bora20v2` contract.

```
333         uint256 _fee = amount.mul(feeRatio).div(10000);
```

- `SafeMath.mul` is called in `_calcAmount` function of `Bora20v2` contract.

Note: Only a sample of 2 `SafeMath` library usage in this contract (out of 14) are shown above.

Recommendation

We advise removing the usage of `SafeMath` library and using the built-in arithmetic operations provided by the Solidity programming language.

Alleviation

`[BORA]` : Issue acknowledged. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

BBO-07 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/Bora20v2.sol: 28, 29, 30, 31, 32	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

`[BORA]`: Issue acknowledged. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

BBO-08 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/Bora20v2.sol: 68~70	Resolved

Description

The linked functions are equivalent to the compiler-generated getter functions for the respective variables.

Recommendation

We advise that the linked variables are instead declared as `public` as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation

`[BORA]` : Issue acknowledged, `_system` is also declared as public for future use in our explorer. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

BBO-09 | UNNECESSARY VALIDATION

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/Bora20v2.sol: 335	Resolved

Description

In function `_calcAmount()`, the following `require` check is redundant. The previous statement has approved it and no underflow edge case would happen because of the overflow/underflow check in the solidity compiler which version is higher than 0.8.0

```
uint256 _amount = amount.sub(_fee);
require(amount == _amount + _fee, "BORA: check a _amount, _fee");
```

Recommendation

We recommend the team remove the `require` check to save gas.

```
function _calcAmount(uint256 amount, uint256 feeRatio) internal pure returns (uint256, uint256) {
    uint256 _fee = amount.mul(feeRatio).div(10000);
    uint256 _amount = amount.sub(_fee);
    return (_amount, _fee);
}
```

Alleviation

`[BORA]`: Issue acknowledged. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

BBO-12 | TAUTOLOGY OR CONTRADICTION

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/Bora20v2.sol: 422, 435	Resolved

Description

Comparisons that are always true or always false may be incorrect or unnecessary.

```
422         require(logNo >= 0); // This is meaningless code to get rid of the
warning
```

```
435         require(logNo >= 0); // This is meaningless code to get rid of the
warning
```

Recommendation

We recommend fixing the incorrect comparison by changing the value type or the comparison operator.

Alleviation

`[BORA]` : Issue acknowledged. Changes have been reflected in the commit hash 0e68a7c4ee88155392b15b13e977de00039aa2cf

FORMAL VERIFICATION | BORA

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-normal	`transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	`transfer` Succeeds on Admissible Self Transfers
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address
erc20-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Non-self Transfers

Property Name	Title
erc20-transfer-correct-amount-self	`transfer` Transfers the Correct Amount in Self Transfers
erc20-transferfrom-succeed-self	`transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-correct-amount-self	`transferFrom` Performs Self Transfers Correctly
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-succeed-normal	`approve` Succeeds for Admissible Inputs

Property Name	Title
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-change-state	`approve` Has No Unexpected State Changes
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-transfer-change-state	`transfer` Has No Unexpected State Changes
erc20-transferfrom-change-state	`transferFrom` Has No Unexpected State Changes

Verification of Compliance with Pausable ERC-721

We verified the properties of the public interface of those token contracts that implement the pausable ERC-721 interface.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc721pausable-transferfrom-revert-pause	`transferFrom` Fails when Paused
erc721pausable-balanceof-succeed-normal	`balanceOf` Succeeds on Admissible Inputs
erc721pausable-supportsinterface-correct-erc721	`supportsInterface` Signals Support for `ERC721`
erc721pausable-balanceof-correct-count	`balanceOf` Returns the Correct Value
erc721pausable-balanceof-revert	`balanceOf` Fails on the Zero Address
erc721pausable-balanceof-no-change-state	`balanceOf` Does Not Change the Contract's State
erc721pausable-ownerof-succeed-normal	`ownerOf` Succeeds For Valid Tokens
erc721pausable-ownerof-correct-owner	`ownerOf` Returns the Correct Owner
erc721pausable-ownerof-revert	`ownerOf` Fails On Invalid Tokens
erc721pausable-ownerof-no-change-state	`ownerOf` Does Not Change the Contract's State
erc721pausable-getapproved-succeed-normal	`getApproved` Succeeds For Valid Tokens
erc721pausable-getapproved-correct-value	`getApproved` Returns Correct Approved Address
erc721pausable-getapproved-revert-zero	`getApproved` Fails on Invalid Tokens

Property Name	Title
erc721pausable-isapprovedforall-succeed-normal	`isApprovedForAll` Always Succeeds
erc721pausable-getapproved-change-state	`getApproved` Does Not Change the Contract's State
erc721pausable-isapprovedforall-change-state	`isApprovedForAll` Does Not Change the Contract's State
erc721pausable-isapprovedforall-correct	`isApprovedForAll` Returns Correct Approvals
erc721pausable-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Inputs
erc721pausable-approve-succeed-normal	`approve` Returns for Admissible Inputs
erc721pausable-approve-set-correct	`approve` Sets Approval
erc721pausable-approve-revert-invalid-token	`approve` Fails For Calls with Invalid Tokens
erc721pausable-approve-change-state	`approve` Has No Unexpected State Changes
erc721pausable-approve-revert-not-allowed	`approve` Prevents Unpermitted Approvals
erc721pausable-setapprovalforall-succeed-normal	`setApprovalForAll` Returns for Admissible Inputs
erc721pausable-setapprovalforall-set-correct	`setApprovalForAll` Approves Operator
erc721pausable-setapprovalforall-multiple	`setApprovalForAll` Can Set Multiple Operators
erc721pausable-setapprovalforall-change-state	`setApprovalForAll` Has No Unexpected State Changes
erc721pausable-transferfrom-correct-one-token-self	`transferFrom` Performs Self Transfers Correctly
erc721pausable-transferfrom-correct-approval	`transferFrom` Updates the Approval Correctly
erc721pausable-transferfrom-correct-increase	`transferFrom` Transfers the Complete Token in Non-self Transfers
erc721pausable-transferfrom-correct-owner-from	`transferFrom` Removes Token Ownership of From
erc721pausable-transferfrom-correct-owner-to	`transferFrom` Transfers Ownership
erc721pausable-transferfrom-correct-state-balance	`transferFrom` Keeps Balances Constant Except for From and To
erc721pausable-transferfrom-revert-invalid	`transferFrom` Fails for Invalid Tokens
erc721pausable-transferfrom-correct-state-owner	`transferFrom` Has Expected Ownership Changes

Property Name	Title
erc721pausable-transferfrom-correct-state-approval	`transferFrom` Has Expected Approval Changes
erc721pausable-transferfrom-revert-not-owned	`transferFrom` Fails if `From` Is Not Token Owner
erc721pausable-transferfrom-revert-exceed-approval	`transferFrom` Fails for Token Transfers without Approval
erc721pausable-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address
erc721pausable-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc721pausable-supportsinterface-metadata	`supportsInterface` Signals that ERC721Metadata is Implemented
erc721pausable-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc721pausable-supportsinterface-enumerable	`supportsInterface` Signals that ERC721Enumerable is Implemented
erc721pausable-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc721pausable-tokenofownerbyindex-revert	`tokenOfOwnerByIndex` Correctly Fails on Token Owner Indices Greater as the Owner Balance
erc721pausable-supportsinterface-succeed-always	`supportsInterface` Always Succeeds
erc721pausable-supportsinterface-correct-erc165	`supportsInterface` Signals Support for ERC165
erc721pausable-supportsinterface-correct-false	`supportsInterface` Returns `False` for Id 0xffffffff
erc721pausable-supportsinterface-no-change-state	`supportsInterface` Does Not Change the Contract's State
erc721pausable-transferfrom-correct-balance	`transferFrom` Sum of Balances is Constant

Verification Results

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
  - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
  - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this

report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.

- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
  - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
  - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Detailed Results For Contract Bora20v2 (projects/BORA/contracts/Bora20v2.sol)

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-succeed-normal	False	
erc20-transfer-succeed-self	False	
erc20-transfer-correct-amount	False	
erc20-transfer-exceed-balance	Inapplicable	Context not considered
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	
erc20-transfer-correct-amount-self	False	
erc20-transfer-recipient-overflow	Inapplicable	Intended behavior
erc20-transfer-change-state	False	
erc20-transfer-revert-zero	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-succeed-normal	False	
erc20-transferfrom-succeed-self	False	
erc20-transferfrom-correct-amount	False	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-correct-amount-self	False	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-fail-exceed-balance	Inapplicable	Intended behavior
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-fail-recipient-overflow	Inapplicable	Intended behavior
erc20-transferfrom-change-state	False	
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-false	True	
erc20-approve-change-state	True	
erc20-approve-never-return-false	True	
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-false	True	
erc20-approve-change-state	True	
erc20-approve-never-return-false	True	

Detailed Results For Contract Bora721v2 (projects/BORA/contracts/Bora721v2.sol)

Verification of Compliance with Pausable ERC-721

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc721pausable-transferfrom-revert-pause	Inconclusive	
erc721pausable-transferfrom-succeed-normal	False	
erc721pausable-transferfrom-correct-one-token-self	True	
erc721pausable-transferfrom-correct-approval	True	
erc721pausable-transferfrom-correct-increase	True	
erc721pausable-transferfrom-correct-owner-from	True	
erc721pausable-transferfrom-correct-owner-to	True	
erc721pausable-transferfrom-correct-state-balance	True	
erc721pausable-transferfrom-revert-invalid	Inconclusive	
erc721pausable-transferfrom-correct-state-owner	True	
erc721pausable-transferfrom-correct-state-approval	True	
erc721pausable-transferfrom-revert-not-owned	Inconclusive	
erc721pausable-transferfrom-revert-exceed-approval	Inconclusive	
erc721pausable-transferfrom-revert-from-zero	True	
erc721pausable-transferfrom-revert-to-zero	True	
erc721pausable-transferfrom-correct-balance	Inapplicable	Intended behavior

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc721pausable-balanceof-succeed-normal	True	
erc721pausable-balanceof-correct-count	True	
erc721pausable-balanceof-revert	True	
erc721pausable-balanceof-no-change-state	True	

Detailed results for function `supportsInterface`

Property Name	Final Result	Remarks
erc721pausable-supportsinterface-correct-erc721	True	
erc721pausable-supportsinterface-metadata	True	
erc721pausable-supportsinterface-enumerable	True	
erc721pausable-supportsinterface-succeed-always	True	
erc721pausable-supportsinterface-correct-erc165	True	
erc721pausable-supportsinterface-correct-false	True	
erc721pausable-supportsinterface-no-change-state	True	

Detailed results for function `ownerOf`

Property Name	Final Result	Remarks
erc721pausable-ownerof-succeed-normal	True	
erc721pausable-ownerof-correct-owner	True	
erc721pausable-ownerof-revert	True	
erc721pausable-ownerof-no-change-state	True	

Detailed results for function `getApproved`

Property Name	Final Result	Remarks
erc721pausable-getapproved-succeed-normal	True	
erc721pausable-getapproved-correct-value	True	
erc721pausable-getapproved-revert-zero	True	
erc721pausable-getapproved-change-state	True	

Detailed results for function `isApprovedForAll`

Property Name	Final Result	Remarks
erc721pausable-isapprovedforall-succeed-normal	True	
erc721pausable-isapprovedforall-change-state	True	
erc721pausable-isapprovedforall-correct	False	BBR-01 : Function `isApprovedForAll` of Contract `Bora721v2` Returns Incorrect Approvals

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc721pausable-approve-succeed-normal	True	
erc721pausable-approve-set-correct	True	
erc721pausable-approve-revert-invalid-token	True	
erc721pausable-approve-change-state	True	
erc721pausable-approve-revert-not-allowed	Inapplicable	Context not considered

Detailed results for function `setApprovalForAll`

Property Name	Final Result	Remarks
erc721pausable-setapprovalforall-succeed-normal	● True	
erc721pausable-setapprovalforall-set-correct	● True	
erc721pausable-setapprovalforall-multiple	● True	
erc721pausable-setapprovalforall-change-state	● True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc721pausable-totalsupply-succeed-always	● True	
erc721pausable-totalsupply-change-state	● True	

Detailed results for function `tokenOfOwnerByIndex`

Property Name	Final Result	Remarks
erc721pausable-tokenofownerbyindex-revert	● True	

APPENDIX | BORA

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

Technical Description

The model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized

non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and Simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any function. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled using modular arithmetic based on the bit-width of the underlying numeric Solidity type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for Property Specification

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time step. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates as atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of the Analyzed ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf`, and `totalSupply`. In the following, we list those

property specifications.

Properties related to function `transfer`

erc20-transfer-revert-zero

`transfer` Prevents Transfers to the Zero Address. Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address. Specification:

```
[](started(contract.transfer(to, value), to == address(0)) ==>
  <>(reverted(contract.transfer) || finished(contract.transfer(to, value), return
    == false)))
```

erc20-transfer-succeed-normal

`transfer` Succeeds on Admissible Non-self Transfers. All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender`,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transfer(to, value), to != address(0) && to != msg.sender &&
    value >= 0 && value <= _balances[msg.sender] && _balances[to] + value <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true)))
```

erc20-transfer-succeed-self

`transfer` Succeeds on Admissible Self Transfers. All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transfer(to, value), to != address(0) && to == msg.sender &&
    value >= 0 && value <= _balances[msg.sender] && _balances[msg.sender] >= 0 &&
    _balances[msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true)))
```

erc20-transfer-correct-amount

`transfer` Transfers the Correct Amount in Non-self Transfers. All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address. Specification:

```
[](willSucceed(contract.transfer(to, value), to != msg.sender && _balances[to] >= 0
    && value >= 0 && _balances[to] + value <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[msg.sender] >= 0 && _balances[msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true ==>
      _balances[msg.sender] == old(_balances[msg.sender]) - value && _balances[to]
      == old(_balances[to]) + value)))
```

erc20-transfer-correct-amount-self

`transfer` Transfers the Correct Amount in Self Transfers. All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender`. Specification:

```
[](willSucceed(contract.transfer(to, value), to == msg.sender && _balances[to] >= 0
    && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transfer(to, value), return == true ==> _balances[to] ==
      old(_balances[to]))))
```

erc20-transfer-change-state

`transfer` Has No Unexpected State Changes. All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses. Specification:

```
[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to) ==>
  <>(finished(contract.transfer(to, value), return == true ==> (_totalSupply ==
        old(_totalSupply) && _allowances == old(_allowances) && _balances[p1] ==
        old(_balances[p1]) && other_state_variables ==
        old(other_state_variables)))))
```

erc20-transfer-exceed-balance

`transfer` Fails if Requested Amount Exceeds Available Balance. Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail. Specification:

```
[](started(contract.transfer(to, value), value > _balances[msg.sender] &&
    _balances[msg.sender] >= 0 && value <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transfer) || finished(contract.transfer(to, value), return
      == false)))
```

erc20-transfer-recipient-overflow

`transfer` Prevents Overflows in the Recipient's Balance. Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow. Specification:

```
[](started(contract.transfer(to, value), to != msg.sender && _balances[to] + value
    >= 0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000 && value >
    0 && value <= _balances[msg.sender]) ==> <>(reverted(contract.transfer) ||
    finished(contract.transfer(to, value), return == false) ||
    finished(contract.transfer(to, value), _balances[to] > old(_balances[to]) +
      value -
      0x1000000000000000000000000000000000000000000000000000000000000000)))
```

erc20-transfer-false

If `transfer` Returns `false`, the Contract State Is Not Changed. If the `transfer` function in contract `contract` fails by returning `false`, it must undo all state changes it incurred before returning to the caller. Specification:

```
[](willSucceed(contract.transfer(to, value)) ==> <>(finished(contract.transfer(to,
        value), return == false ==> (_balances == old(_balances) && _totalSupply ==
        old(_totalSupply) && _allowances == old(_allowances) &&
        other_state_variables == old(other_state_variables)))))
```

erc20-transfer-never-return-false

`transfer` Never Returns `false`. The transfer function must never return `false` to signal a failure. Specification:

```
[](!(finished(contract.transfer, return == false)))
```

Properties related to function `transferFrom`

erc20-transferfrom-revert-from-zero

`transferFrom` Fails for Transfers From the Zero Address. All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), from == address(0)) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
      false)))
```

erc20-transferfrom-revert-to-zero

`transferFrom` Fails for Transfers To the Zero Address. All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), to == address(0)) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
      false)))
```

erc20-transferfrom-succeed-normal

`transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0) && to !=
    address(0) && from != to && value <= _balances[from] && value <=
    _allowances[from][msg.sender] && _balances[to] + value <
    0x1000000000000000000000000000000000000000000000000000000000000000 && value >=
    0 && _balances[to] >= 0 && _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] >= 0 && _allowances[from][msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true)))
```

erc20-transferfrom-succeed-self

`transferFrom` Succeeds on Admissible Self Transfers. All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0) && from == to
    && value <= _balances[from] && value <= _allowances[from][msg.sender] && value
    >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true)))
```

erc20-transferfrom-correct-amount

`transferFrom` Transfers the Correct Amount in Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0 &&
    _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] + value <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
      _balances[from] == old(_balances[from]) - value && _balances[to] ==
      old(_balances[to] + value))))
```

erc20-transferfrom-correct-amount-self

`transferFrom` Performs Self Transfers Correctly. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`). Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from == to && value >= 0 &&
    value < 0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
      _balances[from] == old(_balances[from]))))
```

erc20-transferfrom-correct-allowance

`transferFrom` Updated the Allowance Correctly. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), value >= 0 && value <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _allowances[from][msg.sender] >= 0 && _allowances[from][msg.sender] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, value), return == true ==>
      ((_allowances[from][msg.sender] == old(_allowances[from][msg.sender]) -
      value) || (_allowances[from][msg.sender] ==
      old(_allowances[from][msg.sender]) && (from == msg.sender ||
        old(_allowances[from][msg.sender]) ==
        0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF))))))
```

erc20-transferfrom-change-state

`transferFrom` Has No Unexpected State Changes. All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest`,
- The balance entry for the address in `from`,
- The allowance for the address in `msg.sender` for the address in `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, amount), p1 != from && p1 != to &&
    (p2 != from || p3 != msg.sender)) ==> <>(finished(contract.transferFrom(from,
      to, amount), return == true ==> (_totalSupply == old(_totalSupply) &&
      _balances[p1] == old(_balances[p1]) && _allowances[p2][p3] ==
      old(_allowances[p2][p3]) && other_state_variables ==
      old(other_state_variables)))))
```

erc20-transferfrom-fail-exceed-balance

`transferFrom` Fails if the Requested Amount Exceeds the Available Balance. Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), value > _balances[from] &&
    _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom, return ==
      false)))
```

erc20-transferfrom-fail-exceed-allowance

`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance. Any call of the form `transferFrom(from,`

`dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), msg.sender != from && value >
    _allowances[from][msg.sender] && _allowances[from][msg.sender] >= 0 && value <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom(from, to,
        value), return == false)))
```

erc20-transferfrom-fail-recipient-overflow

`transferFrom` Prevents Overflows in the Recipient's Balance. Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail. Specification:

```
[](started(contract.transferFrom(from, to, value), from != to && _balances[to] +
    value >= 0x1000000000000000000000000000000000000000000000000000000000000000 &&
    value < 0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(reverted(contract.transferFrom) || finished(contract.transferFrom(from, to,
        value), return == false) || finished(contract.transferFrom(from, to,
        value), _balances[to] > old(_balances[to]) + value -
      0x1000000000000000000000000000000000000000000000000000000000000000)))
```

erc20-transferfrom-false

If `transferFrom` Returns `false`, the Contract's State Is Unchanged. If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller. Specification:

```
[](willSucceed(contract.transferFrom(from, to, value)) ==>
  <>(finished(contract.transferFrom(from, to, value), return == false ==>
    (_balances == old(_balances) && _totalSupply == old(_totalSupply) &&
    _allowances == old(_allowances) && other_state_variables ==
    old(other_state_variables)))))
```

erc20-transferfrom-never-return-false

`transferFrom` Never Returns `false`. The `transferFrom` function must never return `false`. Specification:

```
[](!(finished(contract.transferFrom, return == false)))
```

Properties related to function `totalSupply`

erc20-totalsupply-succeed-always

`totalSupply` Always Succeeds. The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

`totalSupply` Returns the Value of the Corresponding State Variable. The `totalSupply` function must return the value that is held in the corresponding state variable of contract contract. Specification:

```
[](willSucceed(contract.totalSupply) ==> <>(finished(contract.totalSupply, return
      == _totalSupply)))
```

erc20-totalsupply-change-state

`totalSupply` Does Not Change the Contract's State. The `totalSupply` function in contract contract must not change any state variables. Specification:

```
[](willSucceed(contract.totalSupply) ==> <>(finished(contract.totalSupply,
      _totalSupply == old(_totalSupply) && _balances == old(_balances) &&
      _allowances == old(_allowances) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

`balanceOf` Always Succeeds. Function `balanceOf` must always succeed if it does not run out of gas. Specification:

```
[](started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

`balanceOf` Returns the Correct Value. Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
      return == _balances[owner])))
```

erc20-balanceof-change-state

`balanceOf` Does Not Change the Contract's State. Function `balanceOf` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
      _totalSupply == old(_totalSupply) && _balances == old(_balances) &&
      _allowances == old(_allowances) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

`allowance` Always Succeeds. Function `allowance` must always succeed, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

`allowance` Returns Correct Value. Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`. Specification:

```
[](willSucceed(contract.allowance(owner, spender)) ==>
  <>(finished(contract.allowance(owner, spender), return ==
    _allowances[owner][spender])))
```

erc20-allowance-change-state

`allowance` Does Not Change the Contract's State. Function `allowance` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.allowance(owner, spender)) ==>
  <>(finished(contract.allowance(owner, spender), _totalSupply == old(_totalSupply)
    && _balances == old(_balances) && _allowances == old(_allowances) &&
    other_state_variables == old(other_state_variables))))
```

Properties related to function `approve`

erc20-approve-revert-zero

`approve` Prevents Approvals For the Zero Address. All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address. Specification:

```
[](started(contract.approve(spender, value), spender == address(0)) ==>
  <>(reverted(contract.approve) || finished(contract.approve(spender, value),
    return == false)))
```

erc20-approve-succeed-normal

`approve` Succeeds for Admissible Inputs. All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas. Specification:

```
[](started(contract.approve(spender, value), spender != address(0)) ==>
  <>(finished(contract.approve(spender, value), return == true)))
```

erc20-approve-correct-amount

`approve` Updates the Approval Mapping Correctly. All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`. Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0) && value >=
    0 && value <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.approve(spender, value), return == true ==>
      _allowances[msg.sender][spender] == value)))
```

erc20-approve-change-state

`approve` Has No Unexpected State Changes. All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes. Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0) && (p1 !=
      msg.sender || p2 != spender)) ==> <>(finished(contract.approve(spender,
        value), return == true ==> _totalSupply == old(_totalSupply) && _balances
      == old(_balances) && _allowances[p1][p2] == old(_allowances[p1][p2]) &&
      other_state_variables == old(other_state_variables))))
```

erc20-approve-false

If `approve` Returns `false`, the Contract's State Is Unchanged. If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller. Specification:

```
[](willSucceed(contract.approve(spender, value)) ==>
  <>(finished(contract.approve(spender, value), return == false ==> (_balances ==
      old(_balances) && _totalSupply == old(_totalSupply) && _allowances ==
      old(_allowances) && other_state_variables == old(other_state_variables)))))
```

erc20-approve-never-return-false

`approve` Never Returns `false`. The function `approve` must never returns `false`. Specification:

```
[](!(finished(contract.approve, return == false)))
```

Description of ERC-721-Pausable Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-721 functions `transferFrom`, `balanceOf`, `ownerOf`, `getApproved`, `isApprovedForAll`, `approve`, `setApprovalForAll` `supportsInterface`, `tokenURI`, `tokenByIndex`, `tokenByIndex`, `decimals` and `totalSupply`. In the following, we list those property specifications.

Properties related to function `transferFrom`

erc721pausable-transferfrom-succeed-normal

`transferFrom` Succeeds on Admissible Inputs. All invocations of `transferFrom(from, to, tokenId)` must succeed if

- address `from` is the owner of token `tokenId`,
- the sender is approved to transfer token `tokenId`,
- the contract is not paused,
- transferring the token to the address `to` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), !_paused && from != address(0)
    && to != address(0) && _owner[tokenId]==from && ((from == msg.sender) ||
      (_approved[tokenId] == msg.sender) || _approvedAll[from][msg.sender]) &&
  _balances[to] >= 0 && _balances[from] >= 1 && _balances[to] <
  0x1000000000000000000000000000000000000000000000000000000000000000 - 1 &&
  _balances[from] <
  0x1000000000000000000000000000000000000000000000000000000000000000) ==> <>
finished(contract.transferFrom(from, to, tokenId)))
```

erc721pausable-transferfrom-revert-pause

`transferFrom` Fails when Paused. Any call of the form `transferFrom(from, to, tokenId)` to a paused contract must fail. Specification:

```
[](started(contract.transferFrom, _paused) ==> <> reverted(contract.transferFrom))
```

erc721pausable-transferfrom-correct-increase

`transferFrom` Transfers the Complete Token in Non-self Transfers. All invocations of `transferFrom(from, to, tokenId)` that succeed must subtract a token from the balance of address `from` and add the token to the balance of address `to`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from != to &&
    _balances[from] > 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 - 1 &&
    _balances[to] >= 0) ==> <>(finished(contract.transferFrom(from, to, tokenId),
      _balances[from] == (old(_balances[from]) - 1) && _balances[to] ==
      (old(_balances[to]) + 1))))
```

erc721pausable-transferfrom-correct-one-token-self

`transferFrom` Performs Self Transfers Correctly. All non-reverting invocations of `transferFrom(from, to, tokenId)` that return `true` and where the address `from` equals the address `to` (i.e. self-transfers) must not change the balance entry of the address `from` (which equals `to`). Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from == to &&
    _owner[tokenId] == from && _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), _balances[from] ==
      old(_balances[from]))))
```

erc721pausable-transferfrom-correct-approval

`transferFrom` Updates the Approval Correctly. All non-reverting invocations of `transferFrom(from, to, tokenId)` that return must remove any approval for token `tokenId`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), p1 != address(0)) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), (_approved[tokenId] !=
        p1))))
```

erc721pausable-transferfrom-correct-owner-from

`transferFrom` Removes Token Ownership of From. All non-reverting and non-self invocations of `transferFrom(from, to, tokenId)` that return, must remove the ownership of token `tokenId` from address `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from != to && from !=
    address(0) && to != address(0) && (msg.sender==from ||
      _approved[tokenId]==msg.sender || _approvedAll[from][msg.sender])) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), (_owner[tokenId] !=
        from))))
```

erc721pausable-transferfrom-correct-owner-to

`transferFrom` Transfers Ownership. All non-reverting invocations of `transferFrom(from, to, tokenId)` must transfer the ownership of token `tokenId` to the address `to`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from != address(0) && to
    != address(0) && _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    (msg.sender==from || _approved[tokenId]==msg.sender ||
    _approvedAll[from][msg.sender])) ==> <>(finished(contract.transferFrom(from,
      to, tokenId), (_owner[tokenId] == to))))
```

erc721pausable-transferfrom-correct-balance

`transferFrom` Sum of Balances is Constant. All non-reverting invocations of `transferFrom(from, to, tokenId)` must keep the sum of token balances constant. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from!=address(0) &&
    _balances[from]>0 && to!=address(0) && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0) ==> <>(finished(contract.transferFrom(from, to, tokenId),
      (old(_balances[from])-_balances[from]) ==
    (_balances[to]-old(_balances[to]))))))
```

erc721pausable-transferfrom-correct-state-balance

`transferFrom` Keeps Balances Constant Except for From and To. All non-reverting invocations of `transferFrom(from, to, tokenId)` must only modify the balance of the addresses `from` and `to`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), p1 != from && p1 != to )
  ==> <>(finished(contract.transferFrom(from, to, tokenId), _balances[p1] ==
      old(_balances[p1]))))
```

erc721pausable-transferfrom-correct-state-owner

`transferFrom` Has Expected Ownership Changes. All non-reverting invocations of `transferFrom(from, to, tokenId)` must only modify the ownership of token `tokenId`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), t1 != tokenId) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), _owner[t1] ==
      old(_owner[t1]) && _owner[t1] == old(_owner[t1]))))
```

erc721pausable-transferfrom-correct-state-approval

`transferFrom` Has Expected Approval Changes. All non-reverting invocations of `transferFrom(from, to, tokenId)` must remove only approvals for token `tokenId` Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), t1 != tokenId) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), _approved[t1] ==
      old(_approved[t1]))))
```

erc721pausable-transferfrom-revert-invalid

`transferFrom` Fails for Invalid Tokens. All calls of the form `transferFrom(from, to, tokenId)` must fail for any invalid token. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), _owner[tokenId] == address(0))
  ==> <>(reverted(contract.transferFrom)))
```

erc721pausable-transferfrom-revert-from-zero

`transferFrom` Fails for Transfers From the Zero Address. All calls of the form `transferFrom(from, to, tokenId)` must fail if the `from` address is zero. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), from == address(0)) ==>
  <>(reverted(contract.transferFrom(from, to, tokenId))))
```

erc721pausable-transferfrom-revert-to-zero

`transferFrom` Fails for Transfers To the Zero Address. All calls of the form `transferFrom(from, to, tokenId)` must fail if the address `to` is the zero address. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), to == address(0)) ==>
  <>(reverted(contract.transferFrom(from, to, tokenId))))
```

erc721pausable-transferfrom-revert-not-owned

`transferFrom` Fails if `From` Is Not Token Owner. Any call of the form `transferFrom(from, to, tokenId)` must fail if address 'from' is not the owner of token `tokenId`. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), _owner[tokenId]!= from) ==>
  <>(reverted(contract.transferFrom)))
```

erc721pausable-transferfrom-revert-exceed-approval

`transferFrom` Fails for Token Transfers without Approval. Any call of the form `transferFrom(from, to, tokenId)` must fail if the sender is neither the token owner nor an operator of the token owner nor approved for token `tokenId`. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), msg.sender!=from &&
    _approved[tokenId]!=msg.sender && !_approvedAll[from][msg.sender]) ==>
  <>(reverted(contract.transferFrom)))
```

Properties related to function `supportsInterface`

erc721pausable-supportsinterface-correct-erc721

`supportsInterface` Signals Support for `ERC721`. Invocations of `supportsInterface(id)` must signal that the interface `ERC721` is implemented. Specification:

```
[](willSucceed(contract.supportsInterface(id), id==0x80ac58cd) ==> <>
  finished(contract.supportsInterface(id), return==true))
```

erc721pausable-supportsinterface-metadata

`supportsInterface` Signals that ERC721Metadata is Implemented. A call of `supportsInterface(interfaceId)` with the interface id of ERC721Metadata must return true. Specification:

```
[](willSucceed(contract.supportsInterface(interfaceId), interfaceId==0x5b5e139f)
  ==> <> finished(contract.supportsInterface(interfaceId), return==true))
```

erc721pausable-supportsinterface-enumerable

`supportsInterface` Signals that ERC721Enumerable is Implemented. Invocations of `supportsInterface(interfaceId)` must signal the support of the interface `ERC721Enumerable` since it is implemented. Specification:

```
[](willSucceed(contract.supportsInterface(interfaceId), interfaceId==0x780e9d63)
  ==> <> finished(contract.supportsInterface(interfaceId), return==true))
```

erc721pausable-supportsinterface-succeed-always

`supportsInterface` Always Succeeds. Function `supportsInterface` must always succeed if it does not run out of gas. Specification:

```
[](started(contract.supportsInterface(id)) ==> <>
  finished(contract.supportsInterface(id)))
```

erc721pausable-supportsinterface-correct-erc165

`supportsInterface` Signals Support for ERC165. Invocations of `supportsInterface(id)` must signal that the interface `ERC165` is implemented. Specification:

```
[](willSucceed(contract.supportsInterface(id), id==0x01ffc9a7) ==> <>
  finished(contract.supportsInterface(id), return==true))
```

erc721pausable-supportsinterface-correct-false

`supportsInterface` Returns `False` for Id 0xffffffff. Invocations of `supportsInterface(id)` with `id` 0xffffffff must return

`false` . Specification:

```
[](willSucceed(contract.supportsInterface(id), id==0xffffffff) ==> <>
  finished(contract.supportsInterface(id), return==false))
```

erc721pausable-supportsinterface-no-change-state

`supportsInterface` Does Not Change the Contract's State. Function `supportsInterface` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.supportsInterface(id)) ==>
  <>(finished(contract.supportsInterface(id), other_state_variables ==
    old(other_state_variables))))
```

Properties related to function `balanceOf`

erc721pausable-balanceof-succeed-normal

`balanceOf` Succeeds on Admissible Inputs. All invocations of `balanceOf(owner)` must succeed if the address `owner` is not zero and it does not run out of gas. Specification:

```
[](started(contract.balanceOf(owner), owner!=address(0)) ==>
  <>(finished(contract.balanceOf)))
```

erc721pausable-balanceof-correct-count

`balanceOf` Returns the Correct Value. Invocations of `balanceOf(owner)` must return the value that is held in the balance mapping for address `owner` . Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
      return == _balances[owner])))
```

erc721pausable-balanceof-revert

`balanceOf` Fails on the Zero Address. Invocations of `balanceOf(owner)` must fail if the address `owner` is the zero address. Specification:

```
[](started(contract.balanceOf(owner), owner==address(0)) ==>
  <>(reverted(contract.balanceOf(owner))))
```

erc721pausable-balanceof-no-change-state

`balanceOf` Does Not Change the Contract's State. Function `balanceOf` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf, _balances ==
      old(_balances) && other_state_variables == old(other_state_variables))))
```

Properties related to function `ownerOf`

erc721pausable-ownerof-succeed-normal

`ownerOf` Succeeds For Valid Tokens. Function `ownerOf(token)` must always succeed for valid tokens if it does not run out of gas. Specification:

```
[](started(contract.ownerOf(token), _owner[token]!=address(0)) ==>
  <>(finished(contract.ownerOf)))
```

erc721pausable-ownerof-correct-owner

`ownerOf` Returns the Correct Owner. Invocations of `ownerOf(token)` must return the owner for a valid token `token` that is held in the contract's owner mapping. Specification:

```
[](willSucceed(contract.ownerOf(token), _owner[token]!=address(0)) ==>
  <>(finished(contract.ownerOf(token), return == _owner[token])))
```

erc721pausable-ownerof-revert

`ownerOf` Fails On Invalid Tokens. Invocations of `ownerOf(token)` must fail for an invalid token. Specification:

```
[](started(contract.ownerOf(token), _owner[token]==address(0)) ==>
  <>(reverted(contract.ownerOf(token))))
```

erc721pausable-ownerof-no-change-state

`ownerOf` Does Not Change the Contract's State. Function `ownerOf` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.ownerOf) ==> <>(finished(contract.ownerOf, _owner ==
      old(_owner) && other_state_variables == old(other_state_variables))))
```

Properties related to function `getApproved`

erc721pausable-getapproved-succeed-normal

`getApproved` Succeeds For Valid Tokens. Function `getApproved` must always succeed for valid tokens, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.getApproved(token), _owner[token]!=address(0)) ==>
  <>(finished(contract.getApproved)))
```

erc721pausable-getapproved-correct-value

`getApproved` Returns Correct Approved Address. Invocations of `getApproved(token)` must return the approved address of a valid `token`. Specification:

```
[](willSucceed(contract.getApproved(token)) ==>
  <>(finished(contract.getApproved(token), return == _approved[token] || return ==
    address(0))))
```

erc721pausable-getapproved-revert-zero

`getApproved` Fails on Invalid Tokens. Invocations of `getApproved(token)` with an invalid token must fail. Specification:

```
[](started(contract.getApproved(token), _owner[token]==address(0)) ==>
  <>(reverted(contract.getApproved)))
```

erc721pausable-getapproved-change-state

`getApproved` Does Not Change the Contract's State. Function `getApproved` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.getApproved) ==> <>(finished(contract.getApproved,
      _approved == old(_approved) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `isApprovedForAll`

erc721pausable-isapprovedforall-succeed-normal

`isApprovedForAll` Always Succeeds. Function `isApprovedForAll` does always succeed, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.isApprovedForAll(owner, operator)) ==>
  <>(finished(contract.isApprovedForAll)))
```

erc721pausable-isapprovedforall-correct

`isApprovedForAll` Returns Correct Approvals. Invocations of `isApprovedForAll(owner, operator)` must return whether a non-zero address `operator` is approved for tokens of a non-zero address `owner`, or return false. Specification:

```
[](willSucceed(contract.isApprovedForAll(owner, operator), owner!=address(0) &&
    operator!=address(0)) ==> <>(finished(contract.isApprovedForAll(owner,
        operator), return == _approvedAll[owner][operator])))
```

erc721pausable-isapprovedforall-change-state

`isApprovedForAll` Does Not Change the Contract's State. Function `isApprovedForAll` does not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.isApprovedForAll) ==>
  <>(finished(contract.isApprovedForAll, _approvedAll == old(_approvedAll) &&
    other_state_variables == old(other_state_variables))))
```

Properties related to function `approve`

erc721pausable-approve-succeed-normal

`approve` Returns for Admissible Inputs. All calls of the form `approve(to, tokenId)` must return if

- the sender is the owner or an authorized operator of the owner
- the token `tokenId` is valid and
- the execution does not run out of gas. Specification:

```
[](started(contract.approve(to, tokenId), (_owner[tokenId]!=address(0)) &&
    (_owner[tokenId]==msg.sender || _approvedAll[_owner[tokenId]][msg.sender]) &&
  (_owner[tokenId]!=to)) ==> <>(finished(contract.approve)))
```

erc721pausable-approve-set-correct

`approve` Sets Approval. Any returning call of the form `approve(to, tokenId)` must approve the address `to` for token `tokenId`. Specification:

```
[](willSucceed(contract.approve(to, tokenId), (_owner[tokenId]!=address(0)) &&
    (_owner[tokenId]==msg.sender || _approvedAll[_owner[tokenId]][msg.sender])) ==>
<>(finished(contract.approve(to, tokenId), _approved[tokenId]==to)))
```

erc721pausable-approve-revert-not-allowed

`approve` Prevents Unpermitted Approvals. All calls of the form `approve(to, tokenId)` must fail if the message sender is not permitted to access token `tokenId`. Specification:

```
[](started(contract.approve(to, tokenId), _owner[tokenId]!=msg.sender &&
    !_approvedAll[_owner[tokenId]][msg.sender]) ==> <>(reverted(contract.approve)))
```

erc721pausable-approve-revert-invalid-token

`approve` Fails For Calls with Invalid Tokens. All calls of the form `approve(to, tokenId)` must fail for an invalid token. Specification:

```
[](started(contract.approve(to, tokenId), _owner[tokenId] == address(0)) ==>
  <>(reverted(contract.approve)))
```

erc721pausable-approve-change-state

`approve` Has No Unexpected State Changes. All calls of the form `approve(to, tokenId)` must only update the allowance mapping according to a valid token `tokenId` and the address `to`, and incur no other state changes. Specification:

```
[](willSucceed(contract.approve(approved, tokenId), t1!=tokenId) ==>
  <>(finished(contract.approve(approved, tokenId),
    _approved[t1]==old(_approved[t1]) && other_state_variables ==
    old(other_state_variables))))
```

Properties related to function `setApprovalForAll`

erc721pausable-setapprovalforall-succeed-normal

`setApprovalForAll` Returns for Admissible Inputs. Calls of the form `setApprovalForAll(operator, approved)` must return if

- the message sender is not the `operator`,
- `operator` is not the zero address and
- the execution does not run out of gas. Specification:

```
[](started(contract.setApprovalForAll(operator, approved), (msg.sender!=operator)
    && (operator!=address(0))) ==> <>(finished(contract.setApprovalForAll)))
```

erc721pausable-setapprovalforall-set-correct

`setApprovalForAll` Approves Operator. All non-reverting calls of the form `setApprovalForAll(operator, approved)` must set the approval of a non-zero address `operator` according to the Boolean value `approved`. Specification:

```
[](willSucceed(contract.setApprovalForAll(operator, approved),
    operator!=address(0)) ==> <>(finished(contract.setApprovalForAll(operator,
        approved), _approvedAll[msg.sender][operator]==approved)))
```

erc721pausable-setapprovalforall-multiple

`setApprovalForAll` Can Set Multiple Operators. Calls of the form `setApprovalForAll(operator, approved)` must be able to set multiple operators for the tokens of the message sender. Specification:

```
[](willSucceed(contract.setApprovalForAll(operator, approved), op1!=address(0) &&
    approved && _approvedAll[msg.sender][op1]  ) ==>
  <>(finished(contract.setApprovalForAll(operator, approved),
    _approvedAll[msg.sender][operator] && _approvedAll[msg.sender][op1])))
```

erc721pausable-setapprovalforall-change-state

`setApprovalForAll` Has No Unexpected State Changes. All calls of the form `setApprovalForAll(operator, approved)` must only update the approval mapping according to the message sender, the address `operator` and the Boolean value `approved` but incur no other state changes. Specification:

```
[](started(contract.setApprovalForAll(op, approved), ow1!=msg.sender || op1!=op)
  ==> <>(finished(contract.setApprovalForAll(op, approved),
      _approvedAll[ow1][op1]==old(_approvedAll[ow1][op1]) &&
      _approvedAll[msg.sender][op]==approved && other_state_variables ==
      old(other_state_variables)) || reverted(contract.setApprovalForAll(op,
        approved))))
```

Properties related to function `totalSupply`

erc721pausable-totalsupply-succeed-always

`totalSupply` Always Succeeds. The function `totalSupply` must always succeed, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc721pausable-totalsupply-change-state

`totalSupply` Does Not Change the Contract's State. The `totalSupply` function in contract contract must not change any state variables. Specification:

```
[](willSucceed(contract.totalSupply) ==> <>(finished(contract.totalSupply, _total
      == old(_total) && _balances == old(_balances) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `tokenOfOwnerByIndex`

erc721pausable-tokenofownerbyindex-revert

`tokenOfOwnerByIndex` Correctly Fails on Token Owner Indices Greater as the Owner Balance. All calls of the form `tokenOfOwnerByIndex(owner, index)` must fail for token owner index `index` that are greater than the owner's balance. Specification:

```
[](started(contract.tokenOfOwnerByIndex(owner, index), _balances[owner]<=index ||
    owner==address(0)) ==> <> reverted(contract.tokenOfOwnerByIndex))
```

Description of ERC-721 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-721 functions `transferFrom`, `balanceOf`, `ownerOf`, `getApproved`, `isApprovedForAll`, `approve`, `setApprovalForAll` `supportsInterface`, `tokenURI`, `tokenByIndex`, `tokenByIndex`, `decimals` and `totalSupply`. In the following, we list those property specifications.

Properties related to function `transferFrom`

erc721-transferfrom-succeed-normal

`transferFrom` Succeeds on Admissible Inputs. All invocations of `transferFrom(from, to, tokenId)` must succeed if

- address `from` is the owner of token `tokenId`,
- the sender is approved to transfer token `tokenId`,
- transferring the token to the address `to` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), from != address(0) && to !=
    address(0) && _owner[tokenId]==from && ((from == msg.sender) ||
      (_approved[tokenId] == msg.sender) || _approvedAll[from][msg.sender]) &&
  _balances[to] >= 0 && _balances[from] >= 1 && _balances[to] <
  0x1000000000000000000000000000000000000000000000000000000000000000 - 1 &&
  _balances[from] <
  0x1000000000000000000000000000000000000000000000000000000000000000) ==> <>
finished(contract.transferFrom(from, to, tokenId)))
```

erc721-transferfrom-correct-increase

`transferFrom` Transfers the Complete Token in Non-self Transfers. All invocations of `transferFrom(from, to, tokenId)` that succeed must subtract a token from the balance of address `from` and add the token to the balance of address `to`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from != to &&
    _balances[from] > 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 - 1 &&
    _balances[to] >= 0) ==> <>(finished(contract.transferFrom(from, to, tokenId),
      _balances[from] == (old(_balances[from]) - 1) && _balances[to] ==
      (old(_balances[to]) + 1))))
```

erc721-transferfrom-correct-one-token-self

`transferFrom` Performs Self Transfers Correctly. All non-reverting invocations of `transferFrom(from, to, tokenId)` that return `true` and where the address `from` equals the address `to` (i.e. self-transfers) must not change the balance entry of the address `from` (which equals `to`). Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from == to &&
    _owner[tokenId] == from && _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), _balances[from] ==
      old(_balances[from]))))
```

erc721-transferfrom-correct-approval

`transferFrom` Updates the Approval Correctly. All non-reverting invocations of `transferFrom(from, to, tokenId)` that return must remove any approval for token `tokenId`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), p1 != address(0)) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), (_approved[tokenId] !=
        p1))))
```

erc721-transferfrom-correct-owner-from

`transferFrom` Removes Token Ownership of From. All non-reverting and non-self invocations of `transferFrom(from, to, tokenId)` that return, must remove the ownership of token `tokenId` from address `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from != to && from !=
    address(0) && to != address(0) && (msg.sender==from ||
      _approved[tokenId]==msg.sender || _approvedAll[from][msg.sender])) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), (_owner[tokenId] !=
        from))))
```

erc721-transferfrom-correct-owner-to

`transferFrom` Transfers Ownership. All non-reverting invocations of `transferFrom(from, to, tokenId)` must transfer the ownership of token `tokenId` to the address `to`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from != address(0) && to
    != address(0) && _balances[from] >= 0 && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0 && _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    (msg.sender==from || _approved[tokenId]==msg.sender ||
    _approvedAll[from][msg.sender])) ==> <>(finished(contract.transferFrom(from,
      to, tokenId), (_owner[tokenId] == to))))
```

erc721-transferfrom-correct-balance

`transferFrom` Sum of Balances is Constant. All non-reverting invocations of `transferFrom(from, to, tokenId)` must keep the sum of token balances constant. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), from!=address(0) &&
    _balances[from]>0 && to!=address(0) && _balances[from] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] <
    0x1000000000000000000000000000000000000000000000000000000000000000 &&
    _balances[to] >= 0) ==> <>(finished(contract.transferFrom(from, to, tokenId),
      (old(_balances[from])-_balances[from]) ==
    (_balances[to]-old(_balances[to]))))
```

erc721-transferfrom-correct-state-balance

`transferFrom` Keeps Balances Constant Except for From and To. All non-reverting invocations of `transferFrom(from, to, tokenId)` must only modify the balance of the addresses `from` and `to`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), p1 != from && p1 != to )
  ==> <>(finished(contract.transferFrom(from, to, tokenId), _balances[p1] ==
      old(_balances[p1]))))
```

erc721-transferfrom-correct-state-owner

`transferFrom` Has Expected Ownership Changes. All non-reverting invocations of `transferFrom(from, to, tokenId)` must only modify the ownership of token `tokenId`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), t1 != tokenId) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), _owner[t1] ==
      old(_owner[t1]) && _owner[t1] == old(_owner[t1]))))
```

erc721-transferfrom-correct-state-approval

`transferFrom` Has Expected Approval Changes. All non-reverting invocations of `transferFrom(from, to, tokenId)` must remove only approvals for token `tokenId` Specification:

```
[](willSucceed(contract.transferFrom(from, to, tokenId), t1 != tokenId) ==>
  <>(finished(contract.transferFrom(from, to, tokenId), _approved[t1] ==
      old(_approved[t1]))))
```

erc721-transferfrom-revert-invalid

`transferFrom` Fails for Invalid Tokens. All calls of the form `transferFrom(from, to, tokenId)` must fail for any invalid token. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), _owner[tokenId] == address(0))
  ==> <>(reverted(contract.transferFrom)))
```

erc721-transferfrom-revert-from-zero

`transferFrom` Fails for Transfers From the Zero Address. All calls of the form `transferFrom(from, to, tokenId)` must fail if the `from` address is zero. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), from == address(0)) ==>
  <>(reverted(contract.transferFrom(from, to, tokenId))))
```

erc721-transferfrom-revert-to-zero

`transferFrom` Fails for Transfers To the Zero Address. All calls of the form `transferFrom(from, to, tokenId)` must fail if the address `to` is the zero address. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), to == address(0)) ==>
  <>(reverted(contract.transferFrom(from, to, tokenId))))
```

erc721-transferfrom-revert-not-owned

`transferFrom` Fails if `From` Is Not Token Owner. Any call of the form `transferFrom(from, to, tokenId)` must fail if address 'from' is not the owner of token `tokenId`. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), _owner[tokenId]!= from) ==>
  <>(reverted(contract.transferFrom)))
```

erc721-transferfrom-revert-exceed-approval

`transferFrom` Fails for Token Transfers without Approval. Any call of the form `transferFrom(from, to, tokenId)` must fail if the sender is neither the token owner nor an operator of the token owner nor approved for token `tokenId`. Specification:

```
[](started(contract.transferFrom(from, to, tokenId), msg.sender!=from &&
    _approved[tokenId]!=msg.sender && !_approvedAll[from][msg.sender]) ==>
  <>(reverted(contract.transferFrom)))
```

Properties related to function `supportsInterface`

erc721-supportsinterface-correct-erc721

`supportsInterface` Signals Support for `ERC721`. Invocations of `supportsInterface(id)` must signal that the interface `ERC721` is implemented. Specification:

```
[](willSucceed(contract.supportsInterface(id), id==0x80ac58cd) ==> <>
  finished(contract.supportsInterface(id), return==true))
```

erc721-supportsinterface-metadata

`supportsInterface` Signals that ERC721Metadata is Implemented. A call of `supportsInterface(interfaceId)` with the interface id of ERC721Metadata must return true. Specification:

```
[](willSucceed(contract.supportsInterface(interfaceId), interfaceId==0x5b5e139f)
  ==> <> finished(contract.supportsInterface(interfaceId), return==true))
```

erc721-supportsinterface-succeed-always

`supportsInterface` Always Succeeds. Function `supportsInterface` must always succeed if it does not run out of gas. Specification:

```
[](started(contract.supportsInterface(id)) ==> <>
  finished(contract.supportsInterface(id)))
```

erc721-supportsinterface-correct-erc165

`supportsInterface` Signals Support for ERC165. Invocations of `supportsInterface(id)` must signal that the interface `ERC165` is implemented. Specification:

```
[](willSucceed(contract.supportsInterface(id), id==0x01ffc9a7) ==> <>
  finished(contract.supportsInterface(id), return==true))
```

erc721-supportsinterface-correct-false

`supportsInterface` Returns `False` for Id 0xffffffff. Invocations of `supportsInterface(id)` with `id` 0xffffffff must return `false` . Specification:

```
[](willSucceed(contract.supportsInterface(id), id==0xffffffff) ==> <>
  finished(contract.supportsInterface(id), return==false))
```

erc721-supportsinterface-no-change-state

`supportsInterface` Does Not Change the Contract's State. Function `supportsInterface` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.supportsInterface(id)) ==>
  <>(finished(contract.supportsInterface(id), other_state_variables ==
    old(other_state_variables))))
```

Properties related to function `balanceOf`

erc721-balanceof-succeed-normal

`balanceOf` Succeeds on Admissible Inputs. All invocations of `balanceOf(owner)` must succeed if the address `owner` is not zero and it does not run out of gas. Specification:

```
[](started(contract.balanceOf(owner), owner!=address(0)) ==>
  <>(finished(contract.balanceOf)))
```

erc721-balanceof-correct-count

`balanceOf` Returns the Correct Value. Invocations of `balanceOf(owner)` must return the value that is held in the balance mapping for address `owner`. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf(owner),
      return == _balances[owner])))
```

erc721-balanceof-revert

`balanceOf` Fails on the Zero Address. Invocations of `balanceOf(owner)` must fail if the address `owner` is the zero address. Specification:

```
[](started(contract.balanceOf(owner), owner==address(0)) ==>
  <>(reverted(contract.balanceOf(owner))))
```

erc721-balanceof-no-change-state

`balanceOf` Does Not Change the Contract's State. Function `balanceOf` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.balanceOf) ==> <>(finished(contract.balanceOf, _balances ==
      old(_balances) && other_state_variables == old(other_state_variables))))
```

Properties related to function `ownerOf`

erc721-ownerof-succeed-normal

`ownerOf` Succeeds For Valid Tokens. Function `ownerOf(token)` must always succeed for valid tokens if it does not run out of gas. Specification:

```
[](started(contract.ownerOf(token), _owner[token]!=address(0)) ==>
  <>(finished(contract.ownerOf)))
```

erc721-ownerof-correct-owner

`ownerOf` Returns the Correct Owner. Invocations of `ownerOf(token)` must return the owner for a valid token `token` that is held in the contract's owner mapping. Specification:

```
[](willSucceed(contract.ownerOf(token), _owner[token]!=address(0)) ==>
  <>(finished(contract.ownerOf(token), return == _owner[token])))
```

erc721-ownerof-revert

`ownerOf` Fails On Invalid Tokens. Invocations of `ownerOf(token)` must fail for an invalid token. Specification:

```
[](started(contract.ownerOf(token), _owner[token]==address(0)) ==>
  <>(reverted(contract.ownerOf(token))))
```

erc721-ownerof-no-change-state

`ownerOf` Does Not Change the Contract's State. Function `ownerOf` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.ownerOf) ==> <>(finished(contract.ownerOf, _owner ==
      old(_owner) && other_state_variables == old(other_state_variables))))
```

Properties related to function `getApproved`

erc721-getapproved-succeed-normal

`getApproved` Succeeds For Valid Tokens. Function `getApproved` must always succeed for valid tokens, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.getApproved(token), _owner[token]!=address(0)) ==>
  <>(finished(contract.getApproved)))
```

erc721-getapproved-correct-value

`getApproved` Returns Correct Approved Address. Invocations of `getApproved(token)` must return the approved address of a valid `token`. Specification:

```
[](willSucceed(contract.getApproved(token)) ==>
  <>(finished(contract.getApproved(token), return == _approved[token] || return ==
    address(0))))
```

erc721-getapproved-revert-zero

`getApproved` Fails on Invalid Tokens. Invocations of `getApproved(token)` with an invalid token must fail. Specification:

```
[](started(contract.getApproved(token), _owner[token]==address(0)) ==>
  <>(reverted(contract.getApproved)))
```

erc721-getapproved-change-state

`getApproved` Does Not Change the Contract's State. Function `getApproved` must not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.getApproved) ==> <>(finished(contract.getApproved,
      _approved == old(_approved) && other_state_variables ==
      old(other_state_variables))))
```

Properties related to function `isApprovedForAll`

erc721-isapprovedforall-succeed-normal

`isApprovedForAll` Always Succeeds. Function `isApprovedForAll` does always succeed, assuming that its execution does not run out of gas. Specification:

```
[](started(contract.isApprovedForAll(owner, operator)) ==>
  <>(finished(contract.isApprovedForAll)))
```

erc721-isapprovedforall-correct

`isApprovedForAll` Returns Correct Approvals. Invocations of `isApprovedForAll(owner, operator)` must return whether a non-zero address `operator` is approved for tokens of a non-zero address `owner`, or return false. Specification:

```
[](willSucceed(contract.isApprovedForAll(owner, operator), owner!=address(0) &&
    operator!=address(0)) ==> <>(finished(contract.isApprovedForAll(owner,
        operator), return == _approvedAll[owner][operator])))
```

erc721-isapprovedforall-change-state

`isApprovedForAll` Does Not Change the Contract's State. Function `isApprovedForAll` does not change any of the contract's state variables. Specification:

```
[](willSucceed(contract.isApprovedForAll) ==>
  <>(finished(contract.isApprovedForAll, _approvedAll == old(_approvedAll) &&
    other_state_variables == old(other_state_variables))))
```

Properties related to function `approve`

erc721-approve-succeed-normal

`approve` Returns for Admissible Inputs. All calls of the form `approve(to, tokenId)` must return if

- the sender is the owner or an authorized operator of the owner
- the token `tokenId` is valid and
- the execution does not run out of gas. Specification:

```
[](started(contract.approve(to, tokenId), (_owner[tokenId]!=address(0)) &&
    (_owner[tokenId]==msg.sender || _approvedAll[_owner[tokenId]][msg.sender]) &&
  (_owner[tokenId]!=to)) ==> <>(finished(contract.approve)))
```

erc721-approve-set-correct

`approve` Sets Approval. Any returning call of the form `approve(to, tokenId)` must approve the address `to` for token `tokenId`. Specification:

```
[](willSucceed(contract.approve(to, tokenId), (_owner[tokenId]!=address(0)) &&
    (_owner[tokenId]==msg.sender || _approvedAll[_owner[tokenId]][msg.sender])) ==>
<>(finished(contract.approve(to, tokenId), _approved[tokenId]==to)))
```

erc721-approve-revert-not-allowed

`approve` Prevents Unpermitted Approvals. All calls of the form `approve(to, tokenId)` must fail if the message sender is not permitted to access token `tokenId`. Specification:

```
[](started(contract.approve(to, tokenId), _owner[tokenId]!=msg.sender &&
    !_approvedAll[_owner[tokenId]][msg.sender]) ==> <>(reverted(contract.approve)))
```

erc721-approve-revert-invalid-token

`approve` Fails For Calls with Invalid Tokens. All calls of the form `approve(to, tokenId)` must fail for an invalid token. Specification:

```
[](started(contract.approve(to, tokenId), _owner[tokenId] == address(0)) ==>
  <>(reverted(contract.approve)))
```

erc721-approve-change-state

`approve` Has No Unexpected State Changes. All calls of the form `approve(to, tokenId)` must only update the allowance mapping according to a valid token `tokenId` and the address `to`, and incur no other state changes. Specification:

```
[](willSucceed(contract.approve(approved, tokenId), t1!=tokenId) ==>
  <>(finished(contract.approve(approved, tokenId),
    _approved[t1]==old(_approved[t1]) && other_state_variables ==
    old(other_state_variables))))
```

Properties related to function `setApprovalForAll`

erc721-setapprovalforall-succeed-normal

`setApprovalForAll` Returns for Admissible Inputs. Calls of the form `setApprovalForAll(operator, approved)` must return if

- the message sender is not the `operator`,
- `operator` is not the zero address and
- the execution does not run out of gas. Specification:

```
[](started(contract.setApprovalForAll(operator, approved), (msg.sender!=operator)
    && (operator!=address(0))) ==> <>(finished(contract.setApprovalForAll)))
```

erc721-setapprovalforall-set-correct

`setApprovalForAll` Approves Operator. All non-reverting calls of the form `setApprovalForAll(operator, approved)` must set the approval of a non-zero address `operator` according to the Boolean value `approved`. Specification:

```
[](willSucceed(contract.setApprovalForAll(operator, approved),
    operator!=address(0)) ==> <>(finished(contract.setApprovalForAll(operator,
        approved), _approvedAll[msg.sender][operator]==approved)))
```

erc721-setapprovalforall-multiple

`setApprovalForAll` Can Set Multiple Operators. Calls of the form `setApprovalForAll(operator, approved)` must be able to set multiple operators for the tokens of the message sender. Specification:

```
[](willSucceed(contract.setApprovalForAll(operator, approved), op1!=address(0) &&
    approved && _approvedAll[msg.sender][op1]  ) ==>
  <>(finished(contract.setApprovalForAll(operator, approved),
    _approvedAll[msg.sender][operator] && _approvedAll[msg.sender][op1])))
```

erc721-setapprovalforall-change-state

`setApprovalForAll` Has No Unexpected State Changes. All calls of the form `setApprovalForAll(operator, approved)` must only update the approval mapping according to the message sender, the address `operator` and the Boolean value `approved` but incur no other state changes. Specification:

```
[](started(contract.setApprovalForAll(op, approved), ow1!=msg.sender || op1!=op)
  ==> <>(finished(contract.setApprovalForAll(op, approved),
      _approvedAll[ow1][op1]==old(_approvedAll[ow1][op1]) &&
      _approvedAll[msg.sender][op]==approved && other_state_variables ==
      old(other_state_variables)) || reverted(contract.setApprovalForAll(op,
        approved))))
```

DISCLAIMER | CERTIK

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 Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with 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, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.