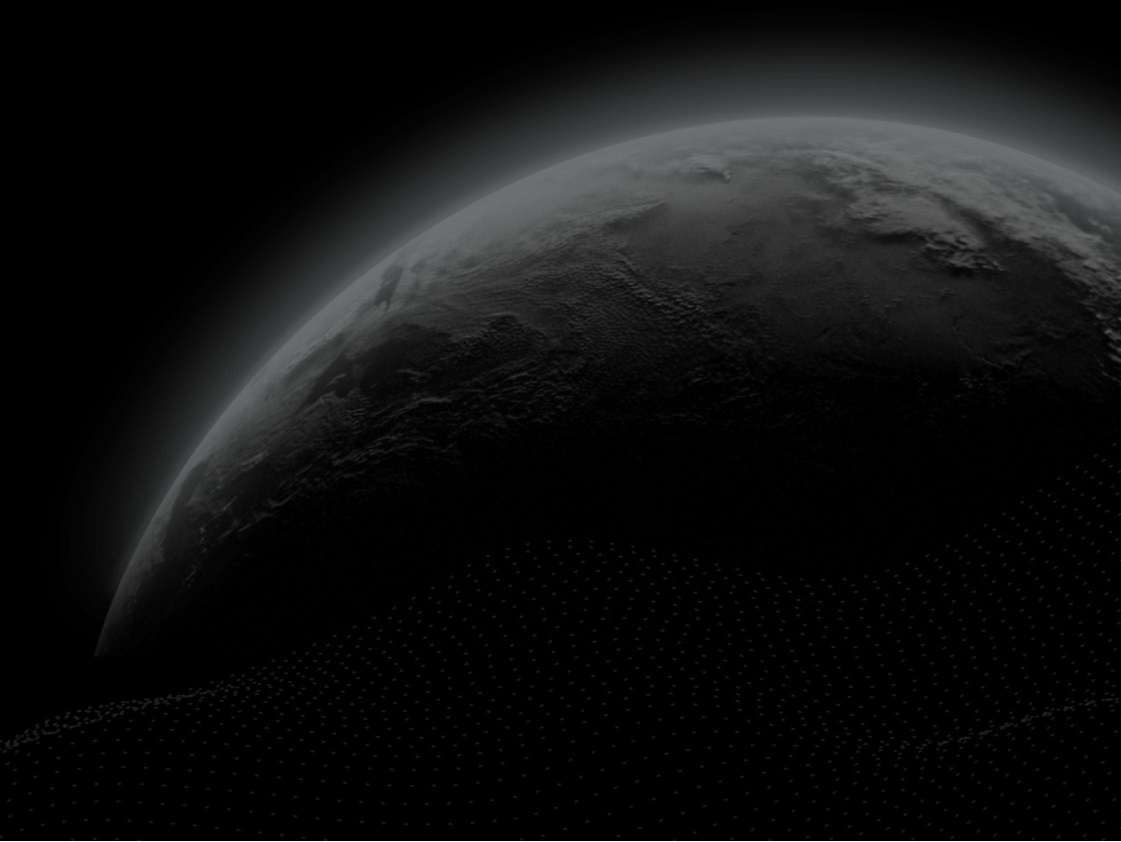
CERTIK

Security Assessment

XRGB-Smart contract audit

CertiK Assessed on Mar 11th, 2024



CertiK Assessed on Mar 11th, 2024

XRGB-Smart contract audit

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Binance Smart Chain (BSC) | Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 03/11/2024	N/A

CODEBASE

https://github.com/XRGB/xrgb-contracts-v1

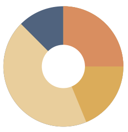
View All in Codebase Page

COMMITS

- 7e1bb55b13f482103e194bd3bc915c5b3cd69f1b
- e978ce7e804b2476d840e003784502bae77bcc82

View All in Codebase Page

Vulnerability Summary



16	13	0	0	3	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

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.
4	Major	1 Resolved, 3 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.
3	Medium	3 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
7	Minor	7 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.
2	Informational	2 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 | XRGB-SMART CONTRACT AUDIT

CODEBASE | XRGB-SMART CONTRACT AUDIT

Repository

https://github.com/XRGB/xrgb-contracts-v1

Commit

- 7e1bb55b13f482103e194bd3bc915c5b3cd69f1b
- e978ce7e804b2476d840e003784502bae77bcc82

AUDIT SCOPE | XRGB-SMART CONTRACT AUDIT

17 files audited ● 6 files with Acknowledged findings ○ 11 files without findings

ID	Repo	File	SHA256 Checksum
● BRC	XRGB/xrgb-contracts-v1	BRC404.sol	3f4da95d82d395ba10c3f45fda588499467418265a2214b4b96224b3a25cd6d7
● BRF	XRGB/xrgb-contracts-v1	BRC404Factory.sol	7eed87c4f6f44fca7e9bb8b4b270b116dbd19293146940f2391924e6f7a538f9
● ERC	XRGB/xrgb-contracts-v1	ERC404.sol	b6f3d0d9f9d18926a859dd20407dc2c6ff5a8b1633dca2f638a0db86e66450a8
● ERX	XRGB/xrgb-contracts-v1	ERC404.sol	351f02bc0d2ff8bf2aebccd9a04438a929ba41b9852fa8271acfac82ad77e031
● BRX	XRGB/xrgb-contracts-v1	BRC404Factory.sol	12157da9d64e4aacd3131354dbdb6b044134cfb453eb9411b1710259186c177b
● BRR	XRGB/xrgb-contracts-v1	BRC404.sol	70cddae7bd4dee9ed9b94c2e6f48798d65dfa78d3456a0ba99d4bc59eed59fc7
○ DTX	XRGB/xrgb-contracts-v1	libraries/DataTypes.sol	2695bb2b976060a5094411ed7834a231b14382bcfccafe877e8dfa8ea6620acb
○ EXR	XRGB/xrgb-contracts-v1	libraries/Errors.sol	aa905c134da1c97a1ab8ae413700c644872da154ce24fa9035c8db91f9539c3b
○ EXG	XRGB/xrgb-contracts-v1	libraries/Events.sol	a233d56552af43cddf3f4b6dea512f38333fa1794b0d487e213fe4d974d870cd
○ BRS	XRGB/xrgb-contracts-v1	storage/BRC404FactoryStorage.sol	da7902143ed2fd19cfae65c5d1c6b7052c7f27f03feae0b32cdc0daba1b2c9e0
○ DTR	XRGB/xrgb-contracts-v1	libraries/DataTypes.sol	2695bb2b976060a5094411ed7834a231b14382bcfccafe877e8dfa8ea6620acb
○ DEQ	XRGB/xrgb-contracts-v1	libraries/DoubleEndedQueue.sol	40d699fdc5e8c08d75d572ca5582c63a927ec048de5d1d1756bb331ce0cc4114
○ ERE	XRGB/xrgb-contracts-v1	libraries/ERC20Events.sol	d47d872424f308f7cf41f9eac2e258f2ff47d18b5167d4732d4f980a4adc1ff8

ID	Repo	File	SHA256 Checksum
● ERR	XRGB/xrgb-contracts-v1	libraries/ERC721Events.sol	a1bb098902f7bf0e798acddcdb70aee0d26184c0c2645ed6a7853a12f46910ca
● EXB	XRGB/xrgb-contracts-v1	libraries/Errors.sol	aa905c134da1c97a1ab8ae413700c644872da154ce24fa9035c8db91f9539c3b
● ERG	XRGB/xrgb-contracts-v1	libraries/Events.sol	a233d56552af43cddf3f4b6dea512f38333fa1794b0d487e213fe4d974d870cd
● BRG	XRGB/xrgb-contracts-v1	storage/BRC404FactoryStorage.sol	da7902143ed2fd19cfae65c5d1c6b7052c7f27f03feae0b32cdc0daba1b2c9e0

APPROACH & METHODS | XRGB-SMART CONTRACT AUDIT

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

The auditing process pays special attention to the following considerations:

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

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.

REVIEW NOTES | XRGB-SMART CONTRACT AUDIT

Overview

XRGB has implemented a tokenomics that combines the fungibility of `ERC20` with the uniqueness of `ERC721` NFTs, the `ERC404` token.

When the `ERC404` contract is created, it will initialize the `units` parameter, which represents the exchange ratio between `ERC20` tokens and `ERC721` NFTs. When a user participates in `ERC20` token transactions, the contract calculates the amount of `ERC721` tokens that the user will receive or burn based on the `units` parameter. Similarly, when a user participates in `ERC721` token transactions, their `ERC20` token balance will increase or decrease by 1 `units` .

Additionally, the factory contract provides token cross-chain functionality, allowing users to provide the target chain and receiving address while burning tokens. On the target chain, the owner of the factory contract will mint an equivalent amount of tokens to the receiving address provided by the user. Please note that the target chain must be a supported chain by the contract, and users need to pay 0.02 native tokens as a fee. The fee and supported chains can be changed at any time by the owner of the factory contract.

External Dependencies

The contract implementations rely on the following libraries to fulfill the needs of their business logics. The scope of the audit treats third-party entities as black boxes and assume their functional correctness.

- `@openzeppelin/contracts/utils/Strings.sol`
- `@openzeppelin/contracts/access/Ownable.sol`
- `@openzeppelin/contracts/utils/ReentrancyGuard.sol`

However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Privileged Functions

In the **XRGB** project, several roles are adopted to operate with some of the key functionalities of the implemented contract. Such privileged roles and methods are described in the findings:

- **XRG-01: Centralized Balance Manipulation**
- **XRG-02: Centralization Related Risks**

The advantage of this privileged role in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth of note the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private key of the privileged account is compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

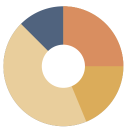
Addendum Audit

The team redesigned the ERC404 contract in the commit e978ce7e804b2476d840e003784502bae77bcc82. The XRGB will apply the latest design introduced in this commit and thus some previous issues will be resolved due to the design change. CertiK performed another round of reviews based on the new design.

Possible Failure On Transferring Small Amount Of ERC404 Token

The team has stated that the inability to transfer small amounts of ERC20 tokens is an intended design feature. However, this limitation is not explicitly coded into the smart contract. As a result, attempts to transfer small quantities of ERC20 tokens may implicitly fail. Any third-party protocols that rely on the token's ERC20 functionality should be aware of the potential failure when transferring small token amounts. They should incorporate appropriate logic to handle situations involving small token transfers.

FINDINGS | XRGB-SMART CONTRACT AUDIT



16	0	4	3	7	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for XRGB-Smart contract audit. Through this audit, we have uncovered 16 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
ERC-05	Inconsistent Enforcement Of Whitelist In ERC404 Token Transfers	Inconsistency, Logical Issue	Major	● Resolved
XRB-02	Possible Failure On Transferring Small Amount Of ERC404 Token	Logical Issue	Major	● Acknowledged
XRB-04	**Centralization Related Risks**	**Centralization**	**Major**	● **Acknowledged**
XRG-01	**Centralized Balance Manipulation**	**Centralization**	**Major**	● **Acknowledged**
BRF-03	`ERC20` And `ERC721` Balance May Not Be Matched Caused By Whitelist Mechanism	Design Issue	Medium	● Resolved
ERC-02	Lack Of Return Value In `transferFrom()`	Design Issue	Medium	● Resolved
XGB-01	Incorrect Burn Logic	Logical Issue	Medium	● Resolved
BRC-02	`tokenURI()` Does Not Comply With ERC721 Metadata Specification	Logical Issue	Minor	● Resolved
BRF-02	The Surplus Native Tokens Are Not Returned	Logical Issue	Minor	● Resolved
ERC-03	Potential Misuse Of `safeTransferFrom()`	Logical Issue	Minor	● Resolved

ID	Title	Category	Severity	Status
ERC-04	Ambiguity In Approval Events	Logical Issue	Minor	● Resolved
ERX-01	Lack Of Validation For Max Supply	Logical Issue, Inconsistency	Minor	● Resolved
ERX-02	Potential Inaccurate ERC721 Total Supply	Logical Issue	Minor	● Resolved
XRB-03	Missing Zero Address Validation	Volatile Code	Minor	● Resolved
ERC-07	Unused Code	Code Optimization	Informational	● Resolved
ERX-03	Potential Incompatible With Proxy Pattern	Logical Issue	Informational	● Resolved

ERC-05 | INCONSISTENT ENFORCEMENT OF WHITELIST IN ERC404 TOKEN TRANSFERS

Category	Severity	Location	Status
Inconsistency, Logical Issue	● Major	ERC404.sol (02/28-7e1bb55): 168	● Resolved

Description

The ERC404 contract is a sophisticated blend of ERC20 and ERC721 standards, introducing a unique approach to handle tokens with mixed characteristics. However, a review of the contract reveals an oversight in the implementation of the `transferFrom()` function, particularly concerning whitelist verification during ERC721 transfers.

In the `transferFrom()` function, the contract distinguishes between ERC20 and ERC721 transfers based on the `amountOrId` parameter. For `amountOrId` values that are less than or equal to the `minted` variable, the contract performs an ERC721 transfer. Otherwise, it handles an ERC20 transfer. The issue occurs within the ERC721 transfer mechanism, as the contract fails to verify whether the recipient is whitelisted prior to executing the transfer. In contrast, the ERC20 transfer mechanism includes this verification.

This lack of whitelist verification can lead to unintended consequences, particularly in scenarios where the contract logic assumes certain addresses (like those of smart contracts designed for liquidity pools or trading platforms) should be prevented from directly receiving ERC721 tokens due to potential compatibility issues.

```
if (amountOrId <= minted) {
    if (from != _ownerOf[amountOrId]) {
        revert InvalidSender();
    }

    if (to == address(0)) {
        revert InvalidRecipient();
    }

    // Authorization checks are performed, but no whitelist verification
    if (
        msg.sender != from &&
        !isApprovedForAll[from][msg.sender] &&
        msg.sender != getApproved[amountOrId]
    ) {
        revert Unauthorized();
    }

    balanceOf[from] -= units;

    unchecked {
        balanceOf[to] += units;
    }

    _ownerOf[amountOrId] = to;
    delete getApproved[amountOrId];

    // Transfer logic for ERC721 tokens does not include whitelist checks
    ...
}
```

Proof of Concept

This proof of concept demonstrates a situation using Foundry that `ERC721` token can be transferred to white list account through `transferFrom()` which is inconsistent with the function `transfer()` .

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import {Test, console2} from "forge-std/Test.sol";
import "../src/contracts/BRC404Factory.sol";
import "../src/contracts/interfaces/IERC404.sol";

contract XRGBTest is Test {

    address private constant Bob = address(0x123);
    address private Alice = address(0x456);
    address private constant Guard = address(0x789);
    address private owner = address(0x678);

    BRC404Factory factory;
    address token;

    function setUp() public {
        factory = new BRC404Factory(owner);
    }

    function testTransferFromToWhitelistAccount() public{
        vm.startPrank(owner);
        token = factory.createBRC404("ERC404 One", "EO", 6, 10000 * 1e6, 10 * 1e6);
        factory.mintBRC404("ERC404 One", Bob, 9000 * 1e6, "0");
        factory.setWhitelist("ERC404 One", Alice, true);
        vm.stopPrank();

        vm.startPrank(Bob);
        IERC404(token).transfer(Alice, 100 * 1e6);
        assertEq(IERC404(token).erc721BalanceOf(Alice), 0);
        IERC404(token).approve(Alice, 200);
        vm.stopPrank();

        vm.startPrank(Alice);
        IERC404(token).transferFrom(Bob, Alice, 200);
        assertNotEq(IERC404(token).erc721BalanceOf(Alice), 0);
        vm.stopPrank();

        //remove Alice from white list, Alice can get NFT through transfer()

        vm.startPrank(owner);
        factory.setWhitelist("ERC404 One", Alice, false);
        vm.stopPrank();

        vm.startPrank(Bob);
        IERC404(token).transfer(Alice, 100 * 1e6);
        assertNotEq(IERC404(token).erc721BalanceOf(Alice), 0);
        vm.stopPrank();

        console2.log("testTransferFromToWhitelistAccount!");
    }
}
```

Output text:

```
forge test -vv
[⠊] Compiling...
[⠒] Compiling 1 files with 0.8.21
[⠢] Solc 0.8.21 finished in 3.74s
Compiler run successful!

Running 1 test for test/XRGB.t.sol:XRGBTest
[PASS] testTransferFromToWhitelistAccount() (gas: 64060051)
Logs:
  testTransferFromToWhitelistAccount!

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 16.69ms
Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Recommendation

We recommend adding white list mechanism to the `if` branch in the function `transferFrom()`.

Alleviation

[XRGB, 03/05/2024]

Issue acknowledged. Changes have been reflected in the commit hash: e978ce7e804b2476d840e003784502bae77bcc82.

XRB-02 | POSSIBLE FAILURE ON TRANSFERRING SMALL AMOUNT OF ERC404 TOKEN

Category	Severity	Location	Status
Logical Issue	● Major	ERC404.sol (02/28-7e1bb55): 168; ERC404.sol (03/05-e978ce7): 446	● Acknowledged

Description

According to the logic of the function `transferFrom()` , if the value of the passed parameter `amountOrId` is less than the value of the state variable `minted` , the transfer operation will be considered as an `ERC721` transfer. A `ERC721` token and unit `ERC20` tokens will be transferred.

```
    function transferFrom(
        address from,
        address to,
        uint256 amountOrId
     ) public virtual {
        if (amountOrId <= minted) {
            if (from != _ownerOf[amountOrId]) {
                revert InvalidSender();
            }

            if (to == address(0)) {
                revert InvalidRecipient();
            }

            if (
                msg.sender != from &&
                !isApprovedForAll[from][msg.sender] &&
                msg.sender != getApproved[amountOrId]
            ) {
                revert Unauthorized();
            }

            balanceOf[from] -= units;

            unchecked {
                balanceOf[to] += units;
            }

            _ownerOf[amountOrId] = to;
            delete getApproved[amountOrId];

            // update _owned for sender
            uint256 updatedId = _owned[from][_owned[from].length - 1];
            _owned[from][_ownedIndex[amountOrId]] = updatedId;
            // pop
            _owned[from].pop();
            // update index for the moved id
            _ownedIndex[updatedId] = _ownedIndex[amountOrId];
            // push token to to owned
            _owned[to].push(amountOrId);
            // update index for to owned
            _ownedIndex[amountOrId] = _owned[to].length - 1;

            emit Transfer(from, to, amountOrId);
            emit ERC20Transfer(from, to, units);
        } else {
            uint256 allowed = allowance[from][msg.sender];
            if (allowed != type(uint256).max)
                allowance[from][msg.sender] = allowed - amountOrId;

            _transfer(from, to, amountOrId);
        }
    }
```

However, if a transfer operation intends to send a small quantity of `ERC20` tokens, the amount will be treated as an `NFT ID`, which is unexpected.

[CertiK, 03/06/2024]

The team updated the codebase in the commit e978ce7e804b2476d840e003784502bae77bcc82 with a new design. The function uses `_isValidTokenId` to check if the transfer is an ERC721 transfer or an ERC20 transfer. For example, in the `trasnferFrom()` function

```
    function transferFrom(
        address from_,
        address to_,
        uint256 valueOrId_
    ) public virtual returns (bool) {
        if (_isValidTokenId(valueOrId_)) {
            erc721TransferFrom(from_, to_, valueOrId_);
        } else {
            // Intention is to transfer as ERC-20 token (value).
            return erc20TransferFrom(from_, to_, valueOrId_);
        }

        return true;
    }
```

The `_isValidTokenId` checks if the id is smaller than `minted`, which is the biggest number of the token id.

```
   function _isValidTokenId(uint256 id_) internal view returns (bool) {
        return id_ <= minted && id_ != type(uint256).max;
    }
```

Therefore, the amount that is smaller than the value of `minted` will be treated as an ERC721 transfer, using `erc721TransferFrom()` to transfer. However, since the sender may not hold the NFT with such an ID, the token transfers will fail.

Proof of Concept

This proof of concept demonstrates a situation using Foundry that the function `transferFrom()` cannot distinguish between `ERC20` and `NFT` transfers if the amount is less than the maximum NFT ID(`minted`).

```
   // SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import {Test, console2} from "forge-std/Test.sol";
import "../src/contracts/BRC404Factory.sol";
import "../src/contracts/interfaces/IERC404.sol";

contract XRGBTest is Test {

    address private constant Bob = address(0x123);
    address private Alice = address(0x456);
    address private constant Guard = address(0x789);
    address private owner = address(0x678);

    BRC404Factory factory;
    address token;

    function setUp() public {
        factory = new BRC404Factory(owner);
    }

    function testTransferFrom() public {
        vm.startPrank(owner);
        token = factory.createBRC404("ERC404 One", "EO", 6, 10000 * 1e6, 10 * 1e6);
        factory.mintBRC404("ERC404 One", Bob, 9000 * 1e6, "0");
        vm.stopPrank();

        vm.startPrank(Bob);
        IERC404(token).approve(Alice, 800);
        assertEq(IERC404(token).allowance(Bob, Alice), 0);
        vm.stopPrank();

        vm.startPrank(Alice);
        IERC404(token).transferFrom(Bob, Alice, 800);
        assertNotEq(IERC404(token).balanceOf(Alice), 800);
        assertEq(IERC404(token).erc721BalanceOf(Alice), 1);
        vm.stopPrank();
    }
}
```

Output text:

```
   forge test -vv
[⠊] Compiling...
[⠒] Compiling 1 files with 0.8.21
[⠢] Solc 0.8.21 finished in 3.23s
Compiler run successful!

Running 1 test for test/XRGB.t.sol:XRGBTest
[PASS] testTransferFrom() (gas: 64424128)

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 16.29ms
Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Recommendation

We recommend distinguishing the range of values for `NFT IDs` from the range of values for `ERC20` tokens. i.e. setting max supply less than the start nft id.

As an example, Pandora applies a prefix (`ID_ENCODING_PREFIX`) as the start of NFT id, which value equals `2**255` . Therefore, the id of the NFTs will be larger than `2**255` .

```
uint256 public constant ID_ENCODING_PREFIX = 1 << 255;
```

Therefore, the `_isValidTokenId` should be modified accordingly:

```
  function _isValidTokenId(uint256 id_) internal pure returns (bool) {
    return id_ > ID_ENCODING_PREFIX && id_ != type(uint256).max;
  }
```

Note: other parts of the contracts related should also be modified accordingly, such as ERC721/ERC20 minting functions

Alleviation

[XRGB, 03/05/2024]

We don't want the NFT ID (nftid) to start from 2^255, so we intend to restrict ERC-20 token transfers, Forbit the erc20 amounts transfer which less than the 'minted' variable. The maximum value for 'minted' will not exceed 10,000 cause use _storedERC721Ids.

[CertiK, 03/05/2024]

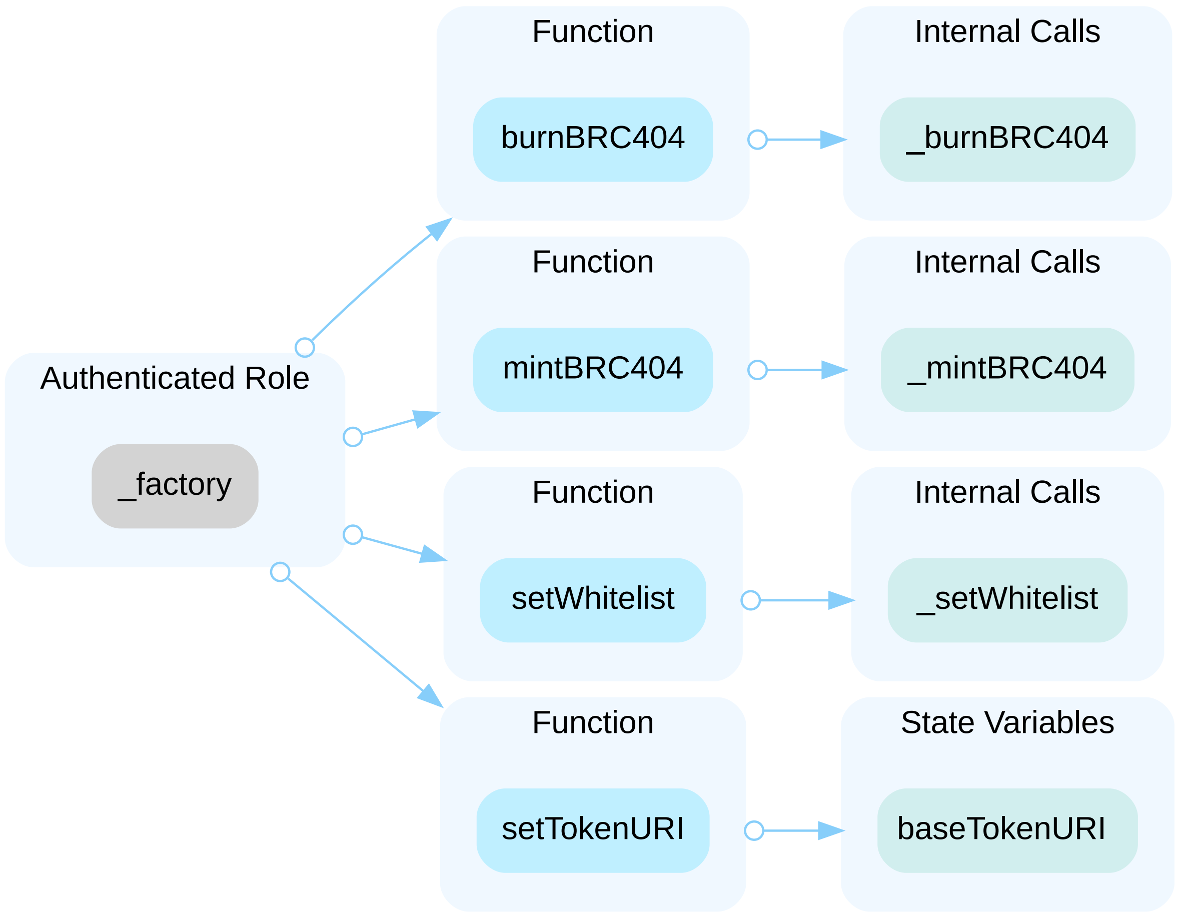
The team has stated that the inability to transfer small amounts of ERC20 tokens is an intended design feature. However, this limitation is not explicitly coded into the smart contract. As a result, attempts to transfer small quantities of ERC20 tokens may implicitly fail. Any third-party protocols that rely on the token's ERC20 functionality should be aware of the potential failure when transferring small token amounts. They should incorporate appropriate logic to handle situations involving small token transfers.

XRB-04 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	● Major	BRC404.sol (02/28-7e1bb55): 31, 39, 43, 47; BRC404Factory.sol (02/28-7e1bb55): 25, 60, 107, 117, 122, 126, 131; BRC404.sol (03/05-e978ce7): 36, 52; BRC404Factory.sol (03/05-e978ce7): 117, 141	● Acknowledged

Description

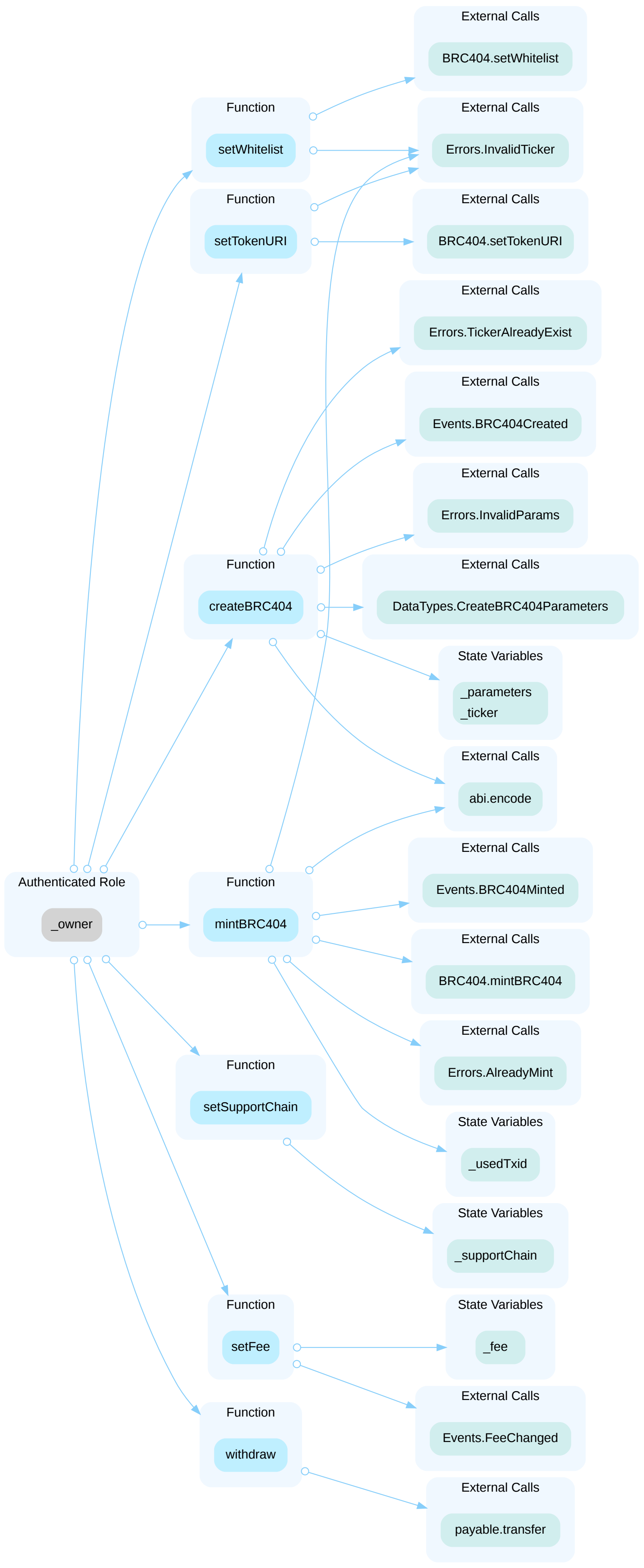
In the contract `BRC404` the role `_factory` has authority over the functions shown in the diagram below.



Any compromise to the **_factory** account may allow a hacker to take advantage of this authority and do the following:

- burn any amount of `BRC404` token from any account.
- mint any amount of `BRC404` token to any account.
- add or remove any account to whitelist.
- set base token uri for `BRC404` token.

In the contract `BRC404Factory` the role `_owner` has authority over the functions shown in the diagram below.



Any compromise to the **_owner** account may allow a hacker to take advantage of this authority and do the following:

- add or remove any account to whitelist.
- set base token uri for `BRC404` token.
- create `BRC404` token.
- mint any amount of `BRC404` token to any account.
- add support chain.
- change the cross-chain fee.
- withdraw native tokens from the `BRC404Factory` contract.

Recommendation

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.

Alleviation

[CertiK, 03/07/2024]

In the new commit e978ce7e804b2476d840e003784502bae77bcc82, a new privileged function `setContractURI()` was added to both contracts `BRC404.sol` and `BRC404Factory.sol` . Besides, old function `setWhitelist()` in contracts `BRC404.sol` and `BRC404Factory.sol` were changed to `setERC721TransferExempt()` . Any compromise to the _**owner**/_**factory** account may allow a hacker to take advantage of this authority and set contract uri, determine whether an account is exempt from `ERC-721` transfer.

[XRGB, 03/08/2024]

Issue acknowledged. We will transfer ownership to a gov smart contract.

[CertiK, 03/08/2024]

As any compromise to the owner account may allow a hacker to take advantage of the authority and mint/burn tokens, set configuration datas. `CertiK` strongly encourages the project team periodically revisit the private key security management of the owner account.

XRG-01 | CENTRALIZED BALANCE MANIPULATION

Category	Severity	Location	Status
Centralization	● **Major**	**BRC404.sol (02/28-7e1bb55): 39, 43; BRC404Factory.sol (02/28-7e1bb55): 60**	● **Acknowledged**

Description

In the contract `BRC404Factory` , the role owner has the authority to update the token balance of an arbitrary account without sanity restriction.

Any compromise to the owner account may allow a hacker to take advantage of this authority and manipulate users' balances. For example, The hacker could also update his/her balance to a large number, sell these tokens, and cause the token price to drop.

Recommendation

We recommend the team makes efforts to restrict access to the private key of the privileged account. A strategy of multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to mint more tokens or engage in similar balance-related operations.
Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently *fully* resolve the risk:

Short Term:

A multi signature (⅔, ⅗) wallet *mitigate* the risk by avoiding a single point of key management failure.

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers' addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A DAO for controlling the operation *mitigate* the risk by applying transparency and decentralization.

- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the multi-signers' addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

The following actions can *fully* resolve the risk:

- Renounce the ownership and never claim back the privileged role.
  OR
- Remove the risky functionality.
  OR
- Add minting logic (such as a vesting schedule) to the contract instead of allowing the owner account to call the sensitive function directly.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[XRGB, 03/08/2024]

Issue acknowledged. We will transfer ownership to a gov smart contract.

[CertiK, 03/08/2024]

As any compromise to the owner account may allow a hacker to take advantage of this authority and manipulate users' balances. `CertiK` strongly encourages the project team periodically revisit the private key security management of the owner account.

BRF-03 ERC20 AND ERC721 BALANCE MAY NOT BE MATCHED CAUSED BY WHITELIST MECHANISM

Category	Severity	Location	Status
Design Issue	● Medium	BRC404Factory.sol (02/28-7e1bb55): 131	● Resolved

Description

Based on the contract design of `ERC404`, if `ERC721` token owners are included in the whitelist, they are allowed to transfer their `ERC20` balance but retain their `ERC721` balance. Consequently, these owners can duplicate the number of `ERC721` tokens they own. To illustrate, let's assume a token owner possesses `t` `ERC721` tokens and `t*units` `ERC20` tokens. Once the owner's account, referred to as `accA`, is whitelisted, they can use the `transfer()` function to transfer their `t*units` `ERC20` tokens to another account, `accB`. It's important to note that `accA` still maintains its `ERC721` balance, which is `t`. Later on, if `accA` is removed from the whitelist, they can transfer the `ERC20` balance from `accB` back to `accA`. As a result of this transfer, `accA` will have `t*units` `ERC20` tokens and a total of `2*t` `ERC721` tokens, including `t` newly minted tokens.

```
    function setWhitelist(
        string memory ticker,
        address target,
        bool state
    ) public onlyOwner {
        if (_ticker[ticker] == address(0x0)) {
            revert Errors.InvalidTicker();
        }
        BRC404(_ticker[ticker]).setWhitelist(target, state);
    }
```

Recommendation

We recommend burning/minting `ERC721` tokens when target account is added/removed to/from whitelist.

Alleviation

[XRGB, 03/05/2024]

Issue acknowledged. Changes have been reflected in the commit hash: e978ce7e804b2476d840e003784502bae77bcc82.

ERC-02 | LACK OF RETURN VALUE IN `transferFrom()`

Category	Severity	Location	Status
Design Issue	● Medium	ERC404.sol (02/28-7e1bb55): 168	● Resolved

Description

The function `transferFrom()` in the contract `ERC404` is designed to implement the experimental `ERC-404` token standard on the blockchain. `ERC-404` was designed to merge the characteristics of fungible (`ERC-20`) and non-fungible tokens (`ERC-721`) into a hybrid form, allowing for both unique and divisible asset representations on the blockchain. However, there is no return value in the function `transferFrom()` .

```
    function transferFrom(
         address from,
         address to,
         uint256 amountOrId
     ) public virtual {
       ...
     }
```

Based on the official `ERC20` standard, it should have a return value in boolean. i.e. `function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)` . Missing return value may result in the inability to integrate with other projects. i.e lending/exchange protocol.

Recommendation

We recommend adding a return value in boolean type to the function `transferFrom()` .

Alleviation

[XRGB, 03/05/2024]

Issue acknowledged. Changes have been reflected in the commit hash: e978ce7e804b2476d840e003784502bae77bcc82.

XGB-01 | INCORRECT BURN LOGIC

Category	Severity	Location	Status
Logical Issue	● Medium	BRC404.sol (03/05-e978ce7): 69; ERC404.sol (03/05-e978ce7): 469	● Resolved

Description

The BRC404 contract, an extension of the ERC404 contract, introduces an issue within its token burn mechanism. This issue stems from the interaction between the `_burnBRC404()` function and its reliance on the `_transferERC20WithERC721()` method for implementing the token burn logic. Specifically, the approach to token burning is misaligned with the conventional expectations of how token burning should affect the total supply of tokens, especially in the context of ERC20 tokens, and mismanages ERC721 token handling.

The `_burnBRC404()` function attempts to burn tokens by internally calling `_transferERC20WithERC721()`, passing the zero address (`address(0)`) as the destination for the tokens being burned. The internal function designed to facilitate the transfer and, by extension, the burning of ERC20 tokens does not decrease the `totalSupply` of the token when tokens are transferred to the zero address. This oversight leads to a discrepancy where burned tokens are not accurately reflected in the total supply, undermining the principle of token burning which intends to reduce the available supply of tokens.

```
function _burnBRC404(address from, uint256 amount) internal {
    _transferERC20WithERC721(from, address(0), amount);
}
```

The `_transferERC20WithERC721()` function, as called above, is expected to handle both ERC20 and ERC721 token transfers. For ERC20 tokens, the function does not decrease the `totalSupply` when transferring to the zero address. This behavior diverges from the standard practice of burning tokens, where the `totalSupply` should be reduced to reflect the removal of tokens from circulation.

```
function _transferERC20(
    address from_,
    address to_,
    uint256 value_
) internal virtual {
    if (from_ == address(0)) {
        totalSupply += value_;
    } else {
        balanceOf[from_] -= value_;
    }
    unchecked {
        balanceOf[to_] += value_;
    }
    emit ERC20Events.Transfer(from_, to_, value_);
}
```

Recommendation

We recommend redesigning the logic of the function `_transferERC20()` to support burning tokens. For example, if the `to_` address is zero address, the `totalSupply` should be decreased accordingly.

Alleviation

[XRGB, 03/08/2024]

Issue acknowledged. Changes have been reflected in the commit hash: 202a78233bedb75a5cff8e33a340b13b7ab0e4b0.

BRC-02 | `tokenURI()` DOES NOT COMPLY WITH ERC721 METADATA SPECIFICATION

Category	Severity	Location	Status
Logical Issue	● Minor	BRC404.sol (02/28-7e1bb55): 35~37	● Resolved

Description

According to the standard, the `tokenURI` method must be reverted if a non-existent `tokenId` is passed. However, in the contract `BRC404` , this point was ignored. What leads to a violation of the EIP721 spec.

Recommendation

It's recommended to check if the token `ID` exists before return any value.

Alleviation

[XRGB, 03/08/2024]

Issue acknowledged. Changes have been reflected in the commit hash: 202a78233bedb75a5cff8e33a340b13b7ab0e4b0.

BRF-02 | THE SURPLUS NATIVE TOKENS ARE NOT RETURNED

Category	Severity	Location	Status
Logical Issue	● Minor	BRC404Factory.sol (02/28-7e1bb55): 79~105	● Resolved

Description

In the `payable` function `burnBRC404()`, there's a validation to ensure `msg.value` meets the minimum required amount of native tokens. However, it's important to note that if `msg.value` exceeds this amount, the function currently lacks a mechanism to refund the excess. This oversight could lead to unintentional loss of funds for the caller, as any surplus in `msg.value` is not returned. Implementing a refund logic for the excess amount is crucial to prevent the potential loss of caller funds.

Recommendation

We recommend adding logic for refunding surplus or modifying the validation process to enforce that the amount of native tokens paid by the caller exactly matches the required amount.

Alleviation

[XRGB, 03/08/2024]

Issue acknowledged. Changes have been reflected in the commit hash: 202a78233bedb75a5cff8e33a340b13b7ab0e4b0.

ERC-03 | POTENTIAL MISUSE OF `safeTransferFrom()`

Category	Severity	Location	Status
Logical Issue	● Minor	ERC404.sol (02/28-7e1bb55): 239, 257	● Resolved

Description

The function `transferFrom()` contains the transfer logic for both `ERC20` and `ERC721` tokens. However, if the transfer of `ERC20` tokens to a contract address is done through the function `safeTransferFrom()` , it will fail due to the validation checks at lines 240 to 242, as standard `ERC20` transfer standard doesn't require the recipient contract implemented the function `onERC721Received()` .

```
   /// @notice Function for native transfers with contract support
    function safeTransferFrom(
        address from,
        address to,
        uint256 id
    ) public virtual {
        transferFrom(from, to, id);

        if (
            to.code.length != 0 &&
            ERC721Receiver(to).onERC721Received(msg.sender, from, id, "") !=
            ERC721Receiver.onERC721Received.selector
        ) {
            revert UnsafeRecipient();
        }
    }

    /// @notice Function for native transfers with contract support and callback
data
    function safeTransferFrom(
        address from,
        address to,
        uint256 id,
        bytes calldata data
    ) public virtual {
        transferFrom(from, to, id);

        if (
            to.code.length != 0 &&
            ERC721Receiver(to).onERC721Received(msg.sender, from, id, data) !=
            ERC721Receiver.onERC721Received.selector
        ) {
            revert UnsafeRecipient();
        }
    }
```

Recommendation

We recommend adding check to avoid transfer `ERC20` balance through the functions `safeTransferFrom()`.

Alleviation

[XRGB, 03/05/2024]

Issue acknowledged. Changes have been reflected in the commit hash: e978ce7e804b2476d840e003784502bae77bcc82.

ERC-04 | AMBIGUITY IN APPROVAL EVENTS

Category	Severity	Location	Status
Logical Issue	● Minor	ERC404.sol (02/28-7e1bb55): 149, 153	● Resolved

Description

The function `approve()` is designed for owners to approve authority to spenders. Authorize both `ERC20` and `ERC721` token types simultaneously. However, both of these two authorities emit the same event which cause ambiguity. The third parameter value represent token id when approving authority for `ERC721` token or token amount when approving authority for `ERC20` token.

```
function approve(
     address spender,
     uint256 amountOrId
 ) public virtual returns (bool) {
     if (amountOrId <= minted && amountOrId > 0) {
         address owner = _ownerOf[amountOrId];

         if (msg.sender != owner && !isApprovedForAll[owner][msg.sender]) {
             revert Unauthorized();
         }

         getApproved[amountOrId] = spender;

         emit Approval(owner, spender, amountOrId);
     } else {
         allowance[msg.sender][spender] = amountOrId;

         emit Approval(msg.sender, spender, amountOrId);
     }

     return true;
 }
```

Recommendation

We recommend defining two different events for these two kinds of authority.

Alleviation

[XRGB, 03/05/2024]

Issue acknowledged. Changes have been reflected in the commit hash: e978ce7e804b2476d840e003784502bae77bcc82.

ERX-01 | LACK OF VALIDATION FOR MAX SUPPLY

Category	Severity	Location	Status
Logical Issue, Inconsistency	● Minor	ERC404.sol (03/05-e978ce7): 655	● Resolved

Description

The function `_mintERC20()` lacks of validation for max supply of `ERC20` tokens.

```
   function _mintERC20(address to_, uint256 value_) internal virtual {
        /// You cannot mint to the zero address (you can't mint and immediately burn
in the same transfer).
        if (to_ == address(0)) {
            revert InvalidRecipient();
        }

        _transferERC20WithERC721(address(0), to_, value_);
    }
```

Recommendation

We recommend adding a check to ensure that the total supply won't exceed the preset max supply or removing the function as it wasn't used in other contracts.

Alleviation

[XRGB, 03/08/2024]

Issue acknowledged. Changes have been reflected in the commit hash: 202a78233bedb75a5cff8e33a340b13b7ab0e4b0.

ERX-02 | POTENTIAL INACCURATE ERC721 TOTAL SUPPLY

Category	Severity	Location	Status
Logical Issue	● Minor	ERC404.sol (03/05-e978ce7): 115	● Resolved

Description

The state variable `minted` represents the total number of `ERC721` tokens that have been minted. Additionally, `ERC721` tokens can be burned, and the corresponding token IDs will be returned to the bank `_storedERC721Ids`, where they can be reused when minting new tokens. However, the function `erc721TotalSupply()` only returns the count of tokens minted thus far and does not account for the tokens that have been burned.

```
function erc721TotalSupply() public view virtual returns (uint256) {
     return minted;
 }
```

Recommendation

We recommend subtracting `_storedERC721Ids.length()` from `minted`, as this represents the accurate count of all existing valid `ERC721` tokens.

Alleviation

[XRGB, 03/08/2024]

Issue acknowledged. Changes have been reflected in the commit hash: 202a78233bedb75a5cff8e33a340b13b7ab0e4b0.

XRB-03 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	BRC404Factory.sol (02/28-7e1bb55): 119; ERC404.sol (02/28-7e1bb55): 112, 137, 160, 268~269; BRC404Factory.sol (03/05-e978ce7): 62	● Resolved

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

BRC404Factory:

```
    function withdraw(address to) external onlyOwner {...}
```

- `to` is not zero-checked before being used.

ERC404:

```
    function _setWhitelist(address target, bool state) internal {...}
```

- `target` is not zero-checked before being used.

```
    function approve(
        address spender,
        uint256 amountOrId
    ) public virtual returns (bool) {...}
```

- `spender` is not zero-checked before being used.

```
    function setApprovalForAll(address operator, bool approved) public virtual {...}
```

- `operator` is not zero-checked before being used.

```
    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal returns (bool) {...}
```

- `from` and `to` are not zero-checked before being used.

```
    function _mintBRC404(address to, uint256 amount) internal {
        if (totalSupply + amount > maxSupply) {
            revert Errors.ExceedMaxSupply();
        }
        _transferERC20WithERC721(address(0), to, amount);
    }
```

- `to` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[CertiK, 03/07/2024]

The team has refactored the design of `ERC404` , the mint function will call the internal function `_transferERC20WithERC721()` to mint `ERC20` and `ERC721` tokens.

```
    function _mintBRC404(address to, uint256 amount) internal {
        if (totalSupply + amount > maxSupply) {
            revert Errors.ExceedMaxSupply();
        }
        _transferERC20WithERC721(address(0), to, amount);
    }
```

[CertiK, 03/08/2024]

Based on `ERC721` standard (https://eips.ethereum.org/EIPS/eip-721), token owners can approve zero address as the spender in order to clear the old allowance. So it is unreasonable to add zero address check for `ERC721` token spender.

```
    function approve(
         address spender_,
         uint256 valueOrId_
     ) public virtual returns (bool) {
         if (spender_ == address(0)) {
             revert InvalidSpender();
         }
         ...
     }

     function erc721Approve(address spender_, uint256 id_) public virtual {
         if (spender_ == address(0)) {
             revert InvalidSpender();
         }
         ...
     }
```

[XRGB, 03/08/2024]

Issue acknowledged. Changes have been reflected in the commit hash: e500c7b71c14feda9327e05d4953fca555f33451.

ERC-07 | UNUSED CODE

Category	Severity	Location	Status
Code Optimization	● Informational	ERC404.sol (02/28-7e1bb55): 48	● 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.

Recommendation

We recommend that the unused code is removed to better prepare the code for production environments.

Alleviation

[XRGB, 03/05/2024]

Issue acknowledged. Changes have been reflected in the commit hash: e978ce7e804b2476d840e003784502bae77bcc82.

ERX-03 | POTENTIAL INCOMPATIBLE WITH PROXY PATTERN

Category	Severity	Location	Status
Logical Issue	● Informational	ERC404.sol (03/05-e978ce7): 418~424	● Resolved

Description

In the `DOMAIN_SEPARATOR()` function to calculate the domain separator for the EIP712 standard, it will be checked if the current blockchain id is similar to the cached one (i.e., `_INITIAL_DOMAIN_SEPARATOR` calculated in the constructor). This check is to ensure the domain separator should be recalculated after a hard fork and prevent potential replay attacks due to it.

```
/// @notice Returns domain initial domain separator, or recomputes if chain id is not equal to initial chain id

    function DOMAIN_SEPARATOR() public view virtual returns (bytes32) {
        return
            block.chainid == _INITIAL_CHAIN_ID
                ? _INITIAL_DOMAIN_SEPARATOR
                : _computeDomainSeparator();
    }
```

However, it will have a compatibility issue if the contract is used as an implementation contract of a proxy, namely the contract is used in `delegatecall`, where the `address(this)` should be the proxy contract. In this case, the domain separator should be recalculated.

As a reference, Openzeppelin fix this issue in the PR #2852.

Recommendation

If the current contract will be used in a proxy, we recommend checking if cached `address(this)` is equal to current `address(this)`. For example, Openzeppelin EIP712 implementation

Alleviation

[XRGB, 03/08/2024]

Issue acknowledged. Changes have been reflected in the commit hash: 202a78233bedb75a5cff8e33a340b13b7ab0e4b0.

FORMAL VERIFICATION | XRGB-SMART CONTRACT AUDIT

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 formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

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-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transferfrom-fail-recipient-overflow	`transferFrom` 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-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Valid Transfers
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed

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

Verification Results

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

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.

- Inapplicable: The property does not apply to the project.

Detailed Results For Contract BRC404 (contracts/BRC404.sol) In Commit e978ce7e804b2476d840e003784502bae77bcc82

Verification of ERC-20 Compliance

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-false	● Inconclusive	
erc20-transferfrom-fail-recipient-overflow	● Inconclusive	
erc20-transferfrom-fail-exceed-allowance	● Inconclusive	
erc20-transferfrom-fail-exceed-balance	● Inconclusive	
erc20-transferfrom-correct-allowance	● Inconclusive	
erc20-transferfrom-correct-amount	● Inconclusive	
erc20-transferfrom-succeed-normal	● Inconclusive	
erc20-transferfrom-revert-zero-argument	● Inconclusive	
erc20-transferfrom-never-return-false	● Inconclusive	

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-never-return-false	● Inconclusive	
erc20-transfer-false	● Inconclusive	
erc20-transfer-recipient-overflow	● Inconclusive	
erc20-transfer-exceed-balance	● Inconclusive	
erc20-transfer-correct-amount	● Inconclusive	
erc20-transfer-succeed-normal	● Inconclusive	
erc20-transfer-revert-zero	● Inconclusive	

Detailed Results for Function `approve`

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

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	

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	

APPENDIX | XRGB-SMART CONTRACT AUDIT

Finding Categories

Categories	Description
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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. Each such contract was compiled into a mathematical model that 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.

The following assumptions and simplifications apply to our model:

- 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 specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

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

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed ERC-20 Properties

Properties related to function `transferFrom`

erc20-transferfrom-correct-allowance

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:

```
ensures \result ==> allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) - \old(amount)
                  || (allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) && \old(allowance(sender, msg.sender)) == type(uint256).max);
```

erc20-transferfrom-correct-amount

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:

```
requires recipient != sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient) +
amount)
                  && balanceOf(\old(sender)) == \old(balanceOf(sender) - amount);
  also
requires recipient == sender;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient));
```

erc20-transferfrom-fail-exceed-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:

```
requires msg.sender != sender;
requires amount > allowance(sender, msg.sender);
ensures !\result;
```

erc20-transferfrom-fail-exceed-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:

```
requires amount > balanceOf(sender);
ensures !\result;
```

erc20-transferfrom-fail-recipient-overflow

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:

```
requires recipient != sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false` .

Specification:

```
ensures \result;
```

erc20-transferfrom-revert-zero-argument

All calls of the form `transferFrom(from, dest, amount)` must fail for transfers from or to the zero address.

Specification:

```
ensures \old(sender) == address(0) ==> !\result;
also
ensures \old(recipient) == address(0) ==> !\result;
```

erc20-transferfrom-succeed-normal

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:

```
requires recipient != address(0) && sender != address(0) && recipient != sender;
requires amount <= balanceOf(sender);
requires amount <= allowance(sender, msg.sender);
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result;
reverts_only_when false;
```

Properties related to function `transfer`

erc20-transfer-correct-amount

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:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(recipient) == \old(balanceOf(recipient) + amount)
&& balanceOf(msg.sender) == \old(balanceOf(msg.sender) - amount);
  also
requires recipient == msg.sender;
ensures \result ==> balanceOf(msg.sender) == \old(balanceOf(msg.sender));
```

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
requires amount > balanceOf(msg.sender);
ensures !\result;
```

erc20-transfer-false

If the `transfer` function in contract `BRC404` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

Specification:

```
ensures \result;
```

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
ensures \old(recipient) == address(0) ==> !\result;
```

erc20-transfer-succeed-normal

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:

```
requires recipient != address(0) && recipient != msg.sender;
requires amount <= balanceOf(msg.sender);
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result;
reverts_only_when false;
```

Properties related to function `approve`

erc20-approve-correct-amount

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:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-approve-never-return-false

The function `approve` must never returns `false` .

Specification:

```
ensures \result;
```

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

erc20-approve-succeed-normal

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:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract BRC404 must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract BRC404.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-succeed-always

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

Specification:

```
reverts_only_when false;
```

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-allowance-correct-value

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

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

erc20-allowance-succeed-always

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

Specification:

```
reverts_only_when false;
```

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-balanceof-correct-value

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

Specification:

```
ensures \result == balanceOf(\old(account));
```

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

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.