



SMART CONTRACT AUDIT REPORT

for

XRGB (ERC404)



Prepared By: Xiaomi Huang

PeckShield
March 11, 2024

Document Properties

Client	XRGB
Title	Smart Contract Audit Report
Target	XRGB
Version	1.0
Author	Xuxian Jiang
Auditors	Jason Shen, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0-rc	March 9, 2024	Xuxian Jiang	Release Candidate #1

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1	Introduction	4
1.1	About XRGB	4
1.2	About PeckShield	5
1.3	Methodology	5
1.4	Disclaimer	7
2	Findings	8
2.1	Summary	8
2.2	Key Findings	9
3	ERC20/ERC721 Compliance	10
3.1	ERC20 Compliance	10
3.2	ERC721 Compliance	12
4	Detailed Results	14
4.1	Adherence of Checks-Effects-Interactions in NFT Redemption	14
4.2	Improved totalSupply Accounting in ERC404::_transferERC20()	16
4.3	Improved Token Transfer Logic in ERC404	17
4.4	Lack of _bNoPermission Initialization in X404Hub	19
4.5	Trust Issue of Admin Keys	20
5	Conclusion	22
	References	23

1 | Introduction

Given the opportunity to review the design document and related source code of the XRGB token contract, we outline in the report our systematic method to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistency between smart contract code and the documentation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of the smart contract can be further improved due to the presence of certain issues related to ERC20/ERC721-compliance, security, or performance. This document outlines our audit results.

1.1 About XRGB

X-404 is an innovative solution designed to revolutionize the NFT marketplace by enhancing liquidity and fostering interoperability across different blockchain networks. It draws inspiration from the ERC404 standard, aiming to create a seamless, decentralized ecosystem where NFTs can be effortlessly exchanged, staked, or traded across various chains. The audited XRGB is a reference implementation of X-404. The basic information of the audited contracts is as follows:

Table 1.1: Basic Information of XRGB

Item	Description
Name	XRGB
Type	ERC20/ERC721 Token Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	March 11, 2024

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- <https://github.com/XRGB/xrgb-x404.git> (2c037e6)

And this is the commit ID after all fixes for the issues found in the audit have been checked in:

- <https://github.com/XRGB/xrgb-x404.git> (8f5a4a9)

1.2 About PeckShield

PeckShield Inc. [10] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystem by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (<https://t.me/peckshield>), Twitter (<http://twitter.com/peckshield>), or Email (contact@peckshield.com).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [9]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk;

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

Table 1.2: Vulnerability Severity Classification

Impact	High	Medium	Low
	Critical	High	Medium
	High	Medium	Low
	Medium	Low	Low
Likelihood			

We perform the audit according to the following procedures:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- ERC20 Compliance Checks: We then manually check whether the implementation logic of the audited smart contract(s) follows the standard ERC20 specification and other best practices.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
	Approve / TransferFrom Race Condition
ERC20 Compliance Checks	Compliance Checks (Section 3)
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool does not identify any issue, the contract is considered safe

regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table [1.3](#).

1.4 Disclaimer



Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.



2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the XRGB token contract. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place ERC20/ERC721-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	1	
Low	4	
Informational	0	
Total	5	

Moreover, we explicitly evaluate whether the given contracts follow the standard ERC20/ERC721 specification and other known best practices, and validate its compatibility with other similar ERC20/ERC721 tokens and current DeFi protocols. The detailed ERC20/ERC721 compliance checks are reported in Section 3. After that, we examine a few identified issues of varying severities that need to be brought up and paid more attention to. (The findings are categorized in the above table.) Additional information can be found in the next subsection, and the detailed discussions are in Section 4.

2.2 Key Findings

Overall, there is no ERC20/ERC721 compliance issue and our detailed checklist can be found in Section 3. While there is no critical or high severity issue, the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 medium-severity vulnerability and 4 low-severity vulnerabilities.

Table 2.1: Key XRGB Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Medium	Adherence of Checks-Effects-Interactions in NFT Redemption	Time And State	Resolved
PVE-002	Low	Improved totalSupply Accounting in ERC404::_transferERC20()	Business Logic	Resolved
PVE-003	Low	Improved Token Transfer Logic in ERC404	Business Logic	Resolved
PVE-004	Low	Lack of _bNoPermission Initialization in X404Hub	Coding Practices	Resolved
PVE-005	Low	Trust Issue Of Admin Keys	Security Features	Mitigated

Besides recommending specific countermeasures to mitigate the above issue(s), we also emphasize that it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for our detailed compliance checks and Section 4 for elaboration of reported issues.

3 | ERC20/ERC721 Compliance

The ERC20/ERC721 specifications define a list of API functions (and relevant events) that each token contract is expected to implement (and emit). The failure to meet these requirements means the token contract cannot be considered to be ERC20/ERC721-compliant. Naturally, as the first step of our audit, we examine the list of API functions defined by the ERC20 specification and validate whether there exist any inconsistency or incompatibility in the implementation or the inherent business logic of the audited contract(s).

3.1 ERC20 Compliance

Table 3.1: Basic [View-Only](#) Functions Defined in The ERC20 Specification

Item	Description	Status
name()	Is declared as a public view function	✓
	Returns a string, for example "Tether USD"	✓
symbol()	Is declared as a public view function	✓
	Returns the symbol by which the token contract should be known, for example "USDT". It is usually 3 or 4 characters in length	✓
decimals()	Is declared as a public view function	✓
	Returns decimals, which refers to how divisible a token can be, from 0 (not at all divisible) to 18 (pretty much continuous) and even higher if required	✓
totalSupply()	Is declared as a public view function	✓
	Returns the number of total supplied tokens, including the total minted tokens (minus the total burned tokens) ever since the deployment	✓
balanceOf()	Is declared as a public view function	✓
	Anyone can query any address' balance, as all data on the blockchain is public	✓
allowance()	Is declared as a public view function	✓
	Returns the amount which the spender is still allowed to withdraw from the owner	✓

Our analysis shows that there is no ERC20 inconsistency or incompatibility issue in the audited XRGB token contract. In the surrounding two tables, we outline the respective list of basic [view](#)-only functions (Table 3.1) and key [state-changing](#) functions (Table 3.2) according to the widely-adopted ERC20 specification.

Table 3.2: Key State-Changing Functions Defined in The ERC20 Specification

Item	Description	Status
transfer()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the caller does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring to zero address	✓
transferFrom()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the spender does not have enough token allowances to spend	✓
	Updates the spender's token allowances when tokens are transferred successfully	✓
	Reverts if the from address does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring from zero address	✓
	Reverts while transferring to zero address	✓
approve()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token approval status	✓
	Emits Approval() event when tokens are approved successfully	✓
	Reverts while approving to zero address	✓
Transfer() event	Is emitted when tokens are transferred, including zero value transfers	✓
	Is emitted with the from address set to <i>address(0x0)</i> when new tokens are generated	✓
Approval() event	Is emitted on any successful call to approve()	✓

In addition, we perform a further examination on certain features that are permitted by the ERC20 specification or even further extended in follow-up refinements and enhancements, but not required for implementation. These features are generally helpful, but may also impact or bring certain incompatibility with current DeFi protocols. Therefore, we consider it is important to highlight them as well. This list is shown in Table 3.3.

Table 3.3: Additional `Opt-in` ERC20 Features Examined in Our Audit

Feature	Description	Opt-in
Deflationary	Part of the tokens are burned or transferred as fee while on <code>transfer()/transferFrom()</code> calls	—
Rebasing	The <code>balanceOf()</code> function returns a re-based balance instead of the actual stored amount of tokens owned by the specific address	—
Pausable	The token contract allows the owner or privileged users to pause the token transfers and other operations	—
Whitelistable	The token contract allows the owner or privileged users to whitelist a specific address such that only token transfers and other operations related to that address are allowed	—
Mintable	The token contract allows the owner or privileged users to mint tokens to a specific address	—
Burnable	The token contract allows the owner or privileged users to burn tokens of a specific address	—

3.2 ERC721 Compliance

The ERC721 standard for non-fungible tokens, also known as deeds. Inspired by the ERC20 token standard, the ERC721 specification defines a list of API functions (and relevant events) that each token contract is expected to implement (and emit). The failure to meet these requirements means the token contract cannot be considered to be ERC721-compliant. Naturally, we examine the list of necessary API functions defined by the ERC721 specification and validate whether there exist any inconsistency or incompatibility in the implementation or the inherent business logic of the audited contract(s).

Table 3.4: Basic `View-Only` Functions Defined in The ERC721 Specification

Item	Description	Status
balanceOf()	Is declared as a public view function	✓
	Anyone can query any address' balance, as all data on the blockchain is public	—
ownerOf()	Is declared as a public view function	✓
	Returns the address of the owner of the NFT	✓
getApproved()	Is declared as a public view function	✓
	Reverts while <code>'_tokenId'</code> does not exist	✓
	Returns the approved address for this NFT	✓
isApprovedForAll()	Is declared as a public view function	✓
	Returns a boolean value which check <code>'_operator'</code> is an approved operator	✓

Our analysis shows that the `balanceOf()` function is defined to be ERC20-compliant. Thus, this

specific function does not count all NFTs assigned to an owner. And there is no other ERC721 inconsistency or incompatibility issue found in the audited Pandora token contract. In the surrounding two tables, we outline the respective list of basic [view-only](#) functions (Table 3.4) and key state-changing functions (Table 3.5) according to the widely-adopted ERC721 specification.

Table 3.5: Key State-Changing Functions Defined in The ERC721 Specification

Item	Description	Status
safeTransferFrom()	Is declared as a public function	✓
	Reverts while 'to' refers to a smart contract and not implement IERC721Receiver-onERC721Received	✓
	Reverts unless 'msg.sender' is the current owner, an authorized operator, or the approved address for this NFT	✓
	Reverts while 'tokenId' is not a valid NFT	✓
	Reverts while 'from' is not the current owner	✓
	Reverts while transferring to zero address	✓
	Emits Transfer() event when tokens are transferred successfully	✓
transferFrom()	Is declared as a public function	✓
	Reverts unless 'msg.sender' is the current owner, an authorized operator, or the approved address for this NFT	✓
	Reverts while 'tokenId' is not a valid NFT	✓
	Reverts while 'from' is not the current owner	✓
	Reverts while transferring to zero address	✓
	Emits Transfer() event when tokens are transferred successfully	✓
approve()	Is declared as a public function	✓
	Reverts unless 'msg.sender' is the current owner, an authorized operator, or the approved address for this NFT	✓
	Emits Approval() event when tokens are approved successfully	✓
setApprovalForAll()	Is declared as a public function	✓
	Reverts while not approving to caller	✓
	Emits ApprovalForAll() event when tokens are approved successfully	✓
Transfer() event	Is emitted when tokens are transferred	✓
Approval() event	Is emitted on any successful call to approve()	✓
ApprovalForAll() event	Is emitted on any successful call to setApprovalForAll()	✓

4 | Detailed Results

4.1 Adherence of Checks-Effects-Interactions in NFT Redemption

- ID: PVE-001
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: ERC404
- Category: Time and State [8]
- CWE subcategory: CWE-663 [3]

Description

A common coding best practice in Solidity is the adherence of checks-effects-interactions principle. This principle is effective in mitigating a serious attack vector known as re-entrancy. Via this particular attack vector, a malicious contract can be reentering a vulnerable contract in a nested manner. Specifically, it first calls a function in the vulnerable contract, but before the first instance of the function call is finished, second call can be arranged to re-enter the vulnerable contract by invoking functions that should only be executed once. This attack was part of several most prominent hacks in Ethereum history, including the DAO [12] exploit, and the Uniswap/Lendf.Me hack [11].

We notice there are occasions where the checks-effects-interactions principle is violated. Using the ERC404 as an example, the `redeemNFT()` function (see the code snippet below) is provided to externally call a token contract to transfer assets. However, the invocation of an external contract requires extra care in avoiding the above re-entrancy. For example, the interaction with the external contract (line 122) start before effecting the update on internal state (lines 128–131), hence violating the principle. In this particular case, if the external contract has certain hidden logic that may be capable of launching re-entrancy via the same entry function.

```
106     function redeemNFT(uint256[] memory tokenIds) external {
107         uint256 len = tokenIds.length;
108         if (len == 0) {
109             revert Errors.InvalidLength();
```

```

110     }
111
112     _transferERC20WithERC721(msg.sender, address(0), units * len);
113
114     for (uint256 i = 0; i < tokenIds.length; ) {
115         address oriOwner = nftDepositInfo[tokenIds[i]].oriOwner;
116         if (
117             oriOwner != msg.sender &&
118             nftDepositInfo[tokenIds[i]].redeemDeadline > block.timestamp
119         ) {
120             revert Errors.NFTCannotRedeem();
121         }
122         IERC721Metadata(blueChipNftAddr).safeTransferFrom(
123             address(this),
124             msg.sender,
125             tokenIds[i]
126         );
127         emit Events.X404RedeemNFT(msg.sender, oriOwner, tokenIds[i]);
128         delete nftDepositInfo[tokenIds[i]];
129         if (!tokenIdSet.remove(tokenIds[i])) {
130             revert Errors.RemoveFailed();
131         }
132         unchecked {
133             i++;
134         }
135     }
136 }

```

Listing 4.1: ERC404::redeemNFT()

We emphasize that the ERC721 feature may naturally have the built-in support for callbacks, which deserve the special attention to guard against possible re-entrancy.

Recommendation Apply necessary reentrancy prevention by following the checks-effects-interactions principle or utilizing the necessary `nonReentrant` modifier to block possible re-entrancy.

Status This issue has been fixed by the following commit: 145dddf.

4.2 Improved totalSupply Accounting in ERC404::_transferERC20()

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: ERC404
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

As mentioned earlier, the ERC404 standard combines the functionalities of ERC20 tokens (fungible tokens) and ERC721 tokens (non-fungible tokens, or NFTs) into a single standard. While examining the ERC20-side functionality, we notice the related `totalSupply` accounting may be improved.

To elaborate, we show below the implementation of the `_transferERC20()` function. As the name indicates, this function is considered as the lowest level ERC-20 transfer function, which should be used for both normal ERC20 transfers as well as minting and burning. However, we notice the `totalSupply` state is not properly updated when it burns the `value_` amount by transferring into `address(0)`, i.e., `to_ == address(0)`.

```

364     function _transferERC20(
365         address from_,
366         address to_,
367         uint256 value_
368     ) internal virtual {
369         // Minting is a special case for which we should not check the balance of
370         // the sender, and we should increase the total supply.
371         if (from_ == address(0)) {
372             totalSupply += value_;
373         } else {
374             // Deduct value from sender's balance.
375             balanceOf[from_] -= value_;
376         }
377
378         // Update the recipient's balance.
379         // Can be unchecked because on mint, adding to totalSupply is checked, and on
380         // transfer balance deduction is checked.
381         unchecked {
382             balanceOf[to_] += value_;
383         }
384         emit ERC20Events.Transfer(from_, to_, value_);
385     }

```

Listing 4.2: ERC404::_transferERC20()

Recommendation Improve the above-mentioned routines to better adjust the `totalSupply` state when there is a burn transfer. An example revision is shown below:

```

364     function _transferERC20(
365         address from_,
366         address to_,
367         uint256 value_
368     ) internal virtual {
369         // Minting is a special case for which we should not check the balance of
370         // the sender, and we should increase the total supply.
371         if (from_ == address(0)) {
372             totalSupply += value_;
373         } else {
374             // Deduct value from sender's balance.
375             balanceOf[from_] -= value_;
376         }
377
378         if (to_ == address(0)) {
379             totalSupply -= value_;
380         } else {
381             // Update the recipient's balance.
382             // Can be unchecked because on mint, adding to totalSupply is checked, and
383             // on transfer balance deduction is checked.
384             unchecked {
385                 balanceOf[to_] += value_;
386             }
387         }
388         emit ERC20Events.Transfer(from_, to_, value_);
389     }

```

Listing 4.3: Revised `ERC404::_transferERC20()`

Status This issue has been fixed by the following commit: 145dddf.

4.3 Improved Token Transfer Logic in ERC404

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: ERC404
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

The combined functionalities of `ERC20` and `ERC721` in `ERC404` make it necessary to revisit each transfer support. In this section, we examine a few helper routines and report possible optimizations.

The first helper routine is `_setOwnedIndex()`, which updates the packed representation of `ownerOf` and `owned` indices of a minted NFT. Apparently, the `ownerOf` field occupies the least significant 160 bits while the `owned` index takes the most significant 96 bits. We notice the given `owned` index has been validated against `_BITMASK_OWNED_INDEX >> 160` (line 684). With that, we can simplify current statement of `and(shl(160, index_), _BITMASK_OWNED_INDEX)` (line 691) to `shl(160, index_)`.

```

681     function _setOwnedIndex(uint256 id_, uint256 index_) internal virtual {
682         uint256 data = _ownedData[id_];
683
684         if (index_ > _BITMASK_OWNED_INDEX >> 160) {
685             revert OwnedIndexOverflow();
686         }
687
688         assembly {
689             data := add(
690                 and(data, _BITMASK_ADDRESS),
691                 and(shl(160, index_), _BITMASK_OWNED_INDEX)
692             )
693         }
694
695         _ownedData[id_] = data;
696     }

```

Listing 4.4: ERC404::_setOwnedIndex()

The second function is `safeTransferFrom()`, which supports moving valid NFTs between users. By design, it does not support ERC20 transfers. With that, we can simplify current implementation by replacing `transferFrom(from_, to_, id_)` (line 321) with `erc721TransferFrom(from_, to_, id_)`.

The third function is `erc20TransferFrom()`, which is used to ERC20 transfers only. With the inherent need of adjusting the spender allowance, we can optimize it by additionally checking whether `from` is equal to `msg.sender`. If yes, there is no need to adjust the allowance and simply allow the transfer.

Recommendation Revisit the above-mentioned routines for improved token transfer logic.

Status This issue has been fixed by the following commit: 145dddaf.

4.4 Lack of `_bNoPermission` Initialization in X404Hub

- ID: PVE-004
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: X404Hub
- Category: Coding Practices [6]
- CWE subcategory: CWE-1126 [1]

Description

DeFi protocols typically have a number of system-wide parameters that can be dynamically configured on demand. The XRGB protocol is no exception. Specifically, if we examine the X404Hub contract, it has defined a number of protocol-wide risk parameters, such as `_bNoPermission` and `_bEmergencyClose`. In the following, we show the corresponding routines that allow for their changes.

```

79     function setNewRedeemDeadline(uint256 newDeadline) public onlyOwner {
80         if (newDeadline == 0) {
81             revert Errors.InvalidRedeemMaxDeadline();
82         }
83         _redeemMaxDeadline = newDeadline;}
84
85     function setSwapRouter(
86         DataTypes.SwapRouter[] memory swapRouterAddr) public onlyOwner {
87         delete _swapRouterAddr;
88         for (uint256 i = 0; i < swapRouterAddr.length; ) {
89             _swapRouterAddr.push(swapRouterAddr[i]);
90             unchecked {
91                 i++;
92             }
93         }
94
95     function emergencyClose(bool bClose) public onlyOwner {
96         _bEmergencyClose = bClose;}

```

Listing 4.5: Example Setters in X404Hub()

These parameters define various aspects of the protocol operation and maintenance and need to exercise extra care when configuring or updating them. Our analysis shows the update logic on these parameters can be improved by applying more rigorous sanity checks. Based on the current implementation, we notice the `_bNoPermission` is not initialized but it is actively checked in the `checkPermission()` modifier.

Recommendation Add the setter support for the `_bNoPermission` parameter.

Status This issue has been fixed by the following commit: 145dddf.

4.5 Trust Issue of Admin Keys

- ID: PVE-005
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: X404Hub
- Category: Security Features [5]
- CWE subcategory: CWE-287 [2]

Description

In the XRGB token contract, there is a privileged admin account `owner` that plays a critical role in regulating the token-wide operations (e.g., configure parameters, set up swap routers, and exercise privileged operations). In the following, we show the representative function potentially affected by this privilege.

```

60     function setContractURI(
61         address nftContract,
62         string calldata newContractUri) public onlyOwner {
63         if (_x404Contract[nftContract] == address(0)) {
64             revert Errors.X404NotCreate();
65         }
66         X404(_x404Contract[nftContract]).setContractURI(newContractUri);}
67
68     function setTokenURI(
69         address nftContract,
70         string calldata newTokenURI) public onlyOwner {
71         if (_x404Contract[nftContract] == address(0)) {
72             revert Errors.X404NotCreate();
73         }
74         X404(_x404Contract[nftContract]).setTokenURI(newTokenURI);}
75
76     function setSupportChain(uint256 chainId, bool bSet) external onlyOwner {
77         _supportChain[chainId] = bSet;}
78
79     function setNewRedeemDeadline(uint256 newDeadline) public onlyOwner {
80         if (newDeadline == 0) {
81             revert Errors.InvalidRedeemMaxDeadline();
82         }
83         _redeemMaxDeadline = newDeadline;}
84
85     function setSwapRouter(
86         DataTypes.SwapRouter[] memory swapRouterAddr) public onlyOwner {
87         delete _swapRouterAddr;
88         for (uint256 i = 0; i < swapRouterAddr.length; ) {
89             _swapRouterAddr.push(swapRouterAddr[i]);
90             unchecked {
91                 i++;
92             }
93         }

```

```
94
95     function emergencyClose(bool bClose) public onlyOwner {
96         _bEmergencyClose = bClose;}
97
98     function setBlueChipNftContract(
99         address[] memory contractAddrs,
100         bool state) public onlyOwner {
101         for (uint256 i = 0; i < contractAddrs.length; ) {
102             if (contractAddrs[i] == address(0x0)) {
103                 revert Errors.CantBeZeroAddress();
104             }
105             _blueChipNftContract[contractAddrs[i]] = state;
106             unchecked {
107                 i++;
108             }
109         }}
```

Listing 4.6: Example Privileged Operations in x404Hub

We emphasize that the privilege assignment may be necessary and consistent with the protocol design. However, if the privileged owner account is a plain EOA account, this may be worrisome and pose counter-party risk to the exchange users. Note that a multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO. In the meantime, a timelock-based mechanism can also be considered as mitigation.

Recommendation Promptly transfer the privileged account to the intended DAO-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status The issue has been mitigated with the use of multi-sig to manage the admin key.

5 | Conclusion

In this security audit, we have examined the XRGB token design and implementation. During our audit, we first checked all respects related to the compatibility of the ERC20/ERC721 specification and other known ERC20/ERC721 pitfalls/vulnerabilities. We then proceeded to examine other areas such as coding practices and business logics. Overall, there are no critical level vulnerabilities discovered and other identified issues are promptly addressed.

=



References

- [1] MITRE. CWE-1126: Declaration of Variable with Unnecessarily Wide Scope. <https://cwe.mitre.org/data/definitions/1126.html>.
- [2] MITRE. CWE-287: Improper Authentication. <https://cwe.mitre.org/data/definitions/287.html>.
- [3] MITRE. CWE-663: Use of a Non-reentrant Function in a Concurrent Context. <https://cwe.mitre.org/data/definitions/663.html>.
- [4] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. <https://cwe.mitre.org/data/definitions/841.html>.
- [5] MITRE. CWE CATEGORY: 7PK - Security Features. <https://cwe.mitre.org/data/definitions/254.html>.
- [6] MITRE. CWE CATEGORY: Bad Coding Practices. <https://cwe.mitre.org/data/definitions/1006.html>.
- [7] MITRE. CWE CATEGORY: Business Logic Errors. <https://cwe.mitre.org/data/definitions/840.html>.
- [8] MITRE. CWE CATEGORY: Concurrency. <https://cwe.mitre.org/data/definitions/557.html>.
- [9] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.
- [10] PeckShield. PeckShield Inc. <https://www.peckshield.com>.

- [11] PeckShield. Uniswap/Lendf.Me Hacks: Root Cause and Loss Analysis. <https://medium.com/@peckshield/uniswap-lendf-me-hacks-root-cause-and-loss-analysis-50f3263dcc09>.
- [12] David Siegel. Understanding The DAO Attack. <https://www.coindesk.com/understanding-dao-hack-journalists>.

