SALUS SECURITY

MAR 2024

CODE SECURITY ASSESSMENT

XRGB

Overview

Project Summary

- Name: XRGB - X404
- Platform: EVM-compatible Chains
- Language: Solidity
- Repository: https://github.com/XRGB/xrgb-x404
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	XRGB - X404
Version	v2
Type	Solidity
Date	Mar 14 2024
Logs	Mar 12 2024; Mar 14 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	1
Total informational issues	4
Total	7

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 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 with 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.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	User can swap owned NFTs with those in the queue	Medium	Business Logic	Acknowledged
2	Integration issue when used as ERC20 tokens	Medium	Compatibility	Acknowledged
3	Centralization risk	Low	Centralization	Acknowledged
4	Inappropriate X404 token receiver in onERC721Received()	Informational	Business Logic	Resolved
5	Redundant codes	Informational	Redundancy	Resolved
6	Could use Ownable2StepUpgradeable	Informational	Code Quality	Acknowledged
7	Implementation contract should not be left uninitialized	Informational	Configuration	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. User can swap owned NFTs with those in the queue

Severity: Medium	Category: Business Logic
Target: - contracts/ERC404.sol	

Description

contracts/ERC404.sol:L14-L15

```
/// @dev The queue of ERC-721 tokens stored in the contract.
DoubleEndedQueue.Uint256Deque private _storedERC721Ids;
```

When an NFT is burned, its token ID goes into the _storedERC721Ids queue. A new NFT, when minted, uses the token ID taken from this queue in a first-in-first-out (FIFO) manner within the ERC404 contract.

This setup creates an attack vector where users can swap their NFTs with those in the queue.

For example, if Alice has tokenId 1, and tokenId 2 is at the end of the queue (i.e. will be the first to pop out). Then Alice can use setSelfERC721TransferExempt(true) to burn tokenId 1 and push it into the queue. Then, by calling setSelfERC721TransferExempt(false), tokenId 2 gets minted to Alice. This way, Alice exchanges the NFT she owns (tokenId 1) with the one in the queue (tokenId 2).

If different NFT IDs are associated with different rarity and value, this attack vector poses a risk to the project as the attacker can monitor the queue using getERC721TokensInQueue(), when a high-value NFT is at the end of the queue, the attacker can exploit the setSelfERC721TransferExempt toggle attack to swap their low-value NFT with the high-value one.

Recommendation

It is recommended to avoid incorporating NFT rarity or similar features into the X404 NFTs, thereby reducing the impact of tokenId on the X404 NFT's value.

If NFT rarity is a must-have feature for the X404 NFT, one way to mitigate this attack vector is to use the queue in a last-in-first-out (LIFO) manner in the ERC404 contract. By doing this, it can hinder the attacker from swiftly swapping owned NFTs with those in the queue by calling setSelfERC721TransferExempt(true) and setSelfERC721TransferExempt(false) consecutively. However, the attacker may still exploit the attacker vector by calling setSelfERC721TransferExempt(true) and waiting for a valuable NFT to be burned before calling setSelfERC721TransferExempt(false) to mint the target tokenId in the queue.

Status

This issue has been acknowledged by the team.

2. Integration issue when used as ERC20 tokens

Severity: Medium	Category: Compatibility
Target: - contracts/ERC404.sol	

Description

1. Integration issue with ERC404's transferFrom() function

contracts/ERC404.sol:L192-L205

```
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 transferFrom() function in the ERC404 contract has a conditional behavior:

- if the valueOrId_ value passed is a valid ERC721 token ID, an ERC721 transfer is triggered
- otherwise, an ERC20 transfer is triggered

Therefore, the caller of ERC404's transferFrom() might unintentionally trigger an ERC721 transfer when its intention is to transfer ERC20 units.

Since the router of uniswap v2/v3 is set as ERC721TransferExempt in the X404 contract, we assume the XRGB X404 token is intended to be swapped in uniswap or other DEXs. However, uniswap's router contract uses the transferFrom() function for ERC20 transfers, which might unintentionally trigger the ERC721 transfer for X404. This could potentially cause unexpected outcomes.

2. Integration issue with ERC404's approve() function

contracts/ERC404.sol:L192-L205

```
function approve(
    address spender_,
    uint256 valueOrId_
) public virtual returns (bool) {
    if (_isValidTokenId(valueOrId_)) {
        erc721Approve(spender_, valueOrId_);
    } else {
        return erc20Approve(spender_, valueOrId_);
    }

    return true;
}
```

The approve() function in the ERC404 contract also has a conditional logic:

- if the valueOrId_ value passed is a valid ERC721 token ID, an ERC721 approve is triggered
- otherwise, an ERC20 approve is triggered

Therefore, the caller of ERC404's approve() might unintentionally trigger ERC721 approval when intending to approve ERC20 units.

It is important to note that the users typically call approve(spender, 0) to revoke the ERC20 approval, and some tools also use this call to assist their users in revoking ERC20 approval. However, calling ERC404's approve(spender, 0) will revert. It is because 0 is a valid ERC721 token ID, making approve(spender, 0) calls erc721Approve(spender, 0), which fails since tokenId 0 is not owned by anyone. As a result, ERC404's approve(spender, 0) call can not be used to revoke ERC20 approval, the user has to explicitly use erc20Approve(spender, 0) to revoke ERC20 approvals.

Recommendation

It's recommended to inform the developers and users about the conditional behavior of ERC404's transferFrom() and approve() functions, where it could trigger either ERC721 or ERC20 transfer/approve based on the valueOrId_ parameter.

Additionally, it is recommended to modify the _isValidTokenId() function to make 0 as an invalid token id. This change could allow the users to use the approve(spender, 0) call to revoke ERC20 approval.

Status

This issue has been acknowledged by the team.

3. Centralization risk

Severity: Low	Category: Centralization
Target: - contract/X404Hub.sol	

Description

There is a privileged owner role in the X404Hub contract. The owner can:

1. Set the state of the blueChipNFTs. The corresponding X404 contract cannot be created if the state is false.
2. Set contractURI and tokenURI for created X404 contracts
3. Set redeemMaxDeadline, which is used as a parameter when creating new X404 contracts.
4. Toggle emergencyClose() to prevent createX404() from being used.

If the owner's private key is compromised, an attacker could exploit the owner's privileged functions to disrupt the project, which could damage the team's reputation.

Recommendation

It is recommended to transfer the owner role to a multisig account with a timelock feature for enhanced security. This ensures that no single person has full control over the account and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

4. Inappropriate X404 token receiver in onERC721Received()

Severity: Informational	Category: Business Logic
Target: - contract/X404.sol	

Description

contract/X404.sol:L159

```
function onERC721Received(
    address caller,
    address from,
    uint256 tokenId,
    bytes calldata data
) external returns (bytes4) {
    ...

    _transferERC20WithERC721(address(0), from, units);

    ...
}
```

Instead of depositing blueChipNFTs using depositNFT(), the X404 contract allows users to use safeTransferFrom() to deposit their blueChipNFTs and receive X404 tokens. However, when an operator transfers the blueChipNFT using safeTransferFrom(), X404 tokens go to the blueChipNFT owner, not the operator.

We believe it is better to transfer the X404 tokens to the operator (i.e. caller) in the onERC721Received() function. This provides developers with greater flexibility to build contracts (e.g. staking, yield optimization) that integrate with the X404 contract.

Recommendation

Consider sending the X404 tokens to the operator (i.e. caller) instead of the owner (i.e. from) in the onERC721Received() function.

Status

This issue has been resolved by the team with commit 139d4ab.

5. Redundant codes

Severity: Informational	Category: Redundancy
Target: - contract/X404.sol	

Description

1. The X404 contract inherits the Ownable contract, but it lacks owner-specific logic. Thus the Ownable inheritance is redundant and can be removed.
2. The onlyEOA() modifier in the X404 contract is defined but never be used, therefore it can be removed.
3. The IUniswapV3PoolState interface is imported in the X404 contract but not be used, therefore it can be removed.
4. The _bNoPermission state variable is defined in the X404HubStorage contract and is used in the X404 contract. However, there is no function in the X404 contract that can switch the _bNoPermission variable, making the usage of _bNoPermission redundant.

Recommendation

Consider removing the redundant codes.

Status

This issue has been resolved by the team with commit cf8cf7c, 145dddf, and 139d4ab.

6. Could use Ownable2StepUpgradeable

Severity: Informational	Category: Code Quality
Target: - contracts/X404Hub.sol	

Description

The X404Hub contract uses the OwnableUpgradeable contract for ownership management.

It is recommended to use Ownable2StepUpgradeable for transferring ownership. Ownable2StepUpgradeable ensures that the recipient confirms ownership, preventing the transfer of permissions to an incorrect or non-existent address.

Recommendation

Consider using the Ownable2StepUpgradeable provided by openzeppelin.

Status

This issue has been acknowledged by the team.

7. Implementation contract should not be left uninitialized

Severity: Informational	Category: Configuration
Target: - contract/X404Hub.sol	

Description

The X404Hub contract uses the upgradeable contract pattern. According to OpenZeppelin's documentation, developers shouldn't leave an implementation contract uninitialized. Therefore, it's best to invoke the _disableInitializers function in the constructor of the upgradeable contracts. This prevents the implementation contract from being initialized by malicious actors.

Recommendation

Consider calling _disableInitializers() in the implementation contract's constructor.

Status

This issue has been resolved by the team with commit 139d4ab.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 2c037e6:

File	SHA-1 hash
contracts/ERC404.sol	494b19b3446ca4d410f233473d7119fda94036d0
contracts/X404.sol	e137ea0e191d147d68edc45abc9c75dacd4ee6a8
contracts/X404Hub.sol	db85d284a90a2c60b6e9975ef52448007cf3cdba
contracts/lib/LibCalculatePair.sol	61a9f4e0d079e4c20242c54eeac6ed03bf84e53b