SALUS SECURITY

JAN 2024

# CODE SECURITY ASSESSMENT

MULTIBIT

# Overview

## Project Summary

- Name:  Multibit - NFT Bridge
- Platform: EVM-compatible Chains
- Language: Solidity
- Audit Scope: See Appendix - 1

# Project Dashboard

## Application Summary

| | |
|---|---|
| **Name** | Multibit - NFT Bridge |
| **Version** | v2 |
| **Type** | Solidity |
| **Dates** | Jan 02 2024 |
| **Logs** | Dec 30 2023; Jan 02 2024 |

## Vulnerability Summary

| | |
|---|---|
| **Total High-Severity issues** | 0 |
| **Total Medium-Severity issues** | 2 |
| **Total Low-Severity issues** | 2 |
| **Total informational issues** | 2 |
| **Total** | 6 |

## 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 | Nonces not used in signed data | Medium | Business Logic | Acknowledged |
| 2 | Centralisation risks | Medium | Centralization | Acknowledged |
| 3 | Missing checks for zero-length ids | Low | Data Validation | Resolved |
| 4 | Direct usage of ecrecover allows signature malleability | Low | Configuration | Acknowledged |
| 5 | Redundant code | Informational | Redundancy | Acknowledged |
| 6 | Same TYPEHASH is used for both mint() and withdraw() | Informational | Business Logic | Acknowledged |

## 2.2 Notable Findings

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

### 1. Nonces not used in signed data

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- MultibitNFTBridge.sol | |

### Description

Digests that have already been used in mint() and withdraw() are recorded in the `used` mapping in order to prevent signature replay attacks.

MultibitNFTBridge.sol:L270-L328

```
function withdraw(
    address token,
    address recipient,
    uint256[] calldata ids,
    bytes calldata data,
    uint8[] calldata v,
    bytes32[] calldata r,
    bytes32[] calldata s
) external payable lock whenNotPaused {
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(
                    WITHDRAW_ERC721_TYPEHASH,
                    token,
                    recipient,
                    keccak256(abi.encodePacked(ids)),
                    keccak256(data)
                ))
            )
        );
        if (used[digest]) {
            revert MultiSignatureReuse(digest);
        }
        used[digest] = true;

}
```

However, nonces are not used in the digest. This means that different transactions would have the same digest if the token, ids, recipients, and data parameters are the same. Since a used digest cannot be reused, the business flow will fail when encountering a previously handled digest. For instance, if an NFT is bridged from BTC to ETH and then back to BTC before being bridged to ETH again, the digest may be the same as the initial bridging digest, causing the bridging process to fail.

## Recommendation

It is recommended to add a nonce state variable and including it to the digest, for example:

```
bytes32 digest = keccak256(
    abi.encodePacked(
        '\x19\x01',
        DOMAIN_SEPARATOR,
        keccak256(abi.encode(
            WITHDRAW_ERC721_TYPEHASH,
            token,
            recipient,
            keccak256(abi.encodePacked(ids)),
            keccak256(data),
            nonce++
        ))
    )
);
```

## Status

This issue has been acknowledged by the team.

## 2. Centralisation risks

| Severity: Medium | Category: Centralization |
| --- | --- |
| Target:<br>- MultibitERC721.sol<br>- MultibitNFTBridge.sol | |

## Description

a) In the MultibitERC721 contract, there is a privileged owner role.

The owner role has the ability to:

- mint new NFTs.
- upgrade the contract’s implementation

If the owner's private key is compromised, the attacker can exploit the owner's role to mint new NFTs or upgrade the contract to a malicious implementation. In this case, the project’s reputation could be damaged.

b) In the MultibitNFTBridge contract, there are two privileged roles: owner and signer.

The owner role can:

- upgrade the bridge contract
- pause the contract
- add or remove signers
- set fee

If the owner's private key is compromised, the attacker can exploit the owner's role to undermine the project.

The signer role is responsible to sign transactions that are used in mint() or withdraw(). If there is only one signer and its private key is compromised, the attacker can exploit the signer’s role to withdraw the NFTs locked in the contract,  causing loss for the users..

## Recommendation

We recommend transferring the owner roles to multisig accounts 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.

We also recommend configuring enough signers and ensuring their safety.

## Status

This issue has been acknowledged by the team.

## 3. Missing checks for zero-length ids

| Severity: Low | Category: Data Validation |
| --- | --- |
| Target:<br>- MultibitNFTBridge.sol | |

## Description

MultibitNFTBridge.sol:L250-L268

```
function deposit(
    address token,
    uint256[] calldata ids,
    bytes calldata data
) external payable lock whenNotPaused {
    uint256 payment = fee * ids.length;
    if (payment > msg.value) {
        revert InsufficientPayment(msg.value);
    }
    if (data.length == 0) {
        revert InvalidRecipientData(data);
    }
    for (uint256 i = 0; i < ids.length; i++) {
        IERC721(token).safeTransferFrom(msg.sender, address(this), ids[i]);
    }
    totalFees += payment;

    emit Deposit(token, msg.sender, ids, payment, data);
}
```

In the deposit() function an empty array of ids is allowed, which will skip all execution steps and submit the “Deposit” event. This means the user can use deposit() to emit a Deposit event with arbitrary `data` by inputting an empty ids array. If the off-chain server processes the `data` parameter from the Deposit event without sufficient validation, this could pose a significant problem, as the user can trigger a Deposit event with user-defined data.

## Recommendation

It is recommended to include a zero-length check for ids in the deposit() function. Additionally, it is advisable to include such zero-length checks for ids in mint() and withdraw().

## Status

This issue has been resolved by the team.

## 4. Direct usage of ecrecover allows signature malleability

| Severity: Low | Category: Configuration |
| --- | --- |
| Target:<br>- MultibitNFTBridge.sol | |

### Description

The withdraw() and mint() functions call the Solidity ecrecover function to verify the given signature. However, the ecrecover EVM opcode allows for malleable (non-unique) signatures and thus is susceptible to attacks.

### Recommendation

It is recommended to use the recover function from OpenZeppelin's ECDSA library for signature verification.

### Status

This issue has been acknowledged by the team.

# 2.3 Informational Findings

## 5. Redundant code

| Severity: Informational | Category: Redundancy |
| --- | --- |
| Target:<br>- MultibitNFTBridge.sol | |

### Description

MultibitNFTBridge.sol:L127-L128

```
bytes32 public constant WITHDRAW_ERC1155_TYPEHASH =
    keccak256(abi.encodePacked("WithdrawERC1155(address token,address to,uint256[]
ids,uint256[] values,bytes data)"));
```

a) The WITHDRAW_ERC1155_TYPEHASH is defined but not used in the contract; thus, this variable can be removed.

MultibitNFTBridge.sol:L65-L111

```
contract TokenCallbackHandler is IERC721Receiver, IERC1155Receiver {
    function tokensReceived(
        address,
        address,
        address,
        uint256,
        bytes calldata,
        bytes calldata
    ) external pure {
    }

    function onERC721Received(
        address,
        address,
        uint256,
        bytes calldata
    ) external pure override returns (bytes4) {
        return IERC721Receiver.onERC721Received.selector;
    }

    function onERC1155Received(
        address,
        address,
        uint256,
        uint256,
        bytes calldata
    ) external pure override returns (bytes4) {
        return IERC1155Receiver.onERC1155Received.selector;
    }

    function onERC1155BatchReceived(
        address,
        address,
        uint256[] calldata,
        uint256[] calldata,
        bytes calldata
```

```
    ) external pure override returns (bytes4) {
        return IERC1155Receiver.onERC1155BatchReceived.selector;
    }

    function supportsInterface(bytes4 interfaceId) external view virtual override
returns (bool) {
        return
            interfaceId == type(IERC721Receiver).interfaceId ||
            interfaceId == type(IERC1155Receiver).interfaceId ||
            interfaceId == type(IERC165).interfaceId;
    }
}
```

b) The MultibitNFTBridge contract inherits the TokenCallbackHandler contract, allowing it to receive ERC1155 and ERC777 tokens. However, there is no function to transfer these tokens out of the contract, causing them to become locked. If the business logic does not involve ERC1155 and ERC777 tokens, the unnecessary callback handler functions can be removed.

## Recommendation

It is recommended to remove the redundant code.

## Status

This issue has been acknowledged by the team.

## 6. Same TYPEHASH is used for both mint() and withdraw()

| Severity: Informational | Category: Business Logic |
| --- | --- |
| Target:<br>- MultibitNFTBridge.sol | |

## Description

In MultibitNFTBridge, *WITHDRAW_ERC721_TYPEHASH* is used for both the mint() and withdraw() functions.

MultibitNFTBridge.sol:L359-L420

```
function mint(
    address token,
    address recipient,
    uint256[] calldata ids,
    bytes calldata data,
    uint8[] memory v,
    bytes32[] memory r,
    bytes32[] memory s
) external payable lock whenNotPaused {
    bytes32 digest = keccak256(
        abi.encodePacked(
            '\x19\x01',
            DOMAIN_SEPARATOR,
            keccak256(abi.encode(
                WITHDRAW_ERC721_TYPEHASH,
                token,
                recipient,
                keccak256(abi.encodePacked(ids)),
                keccak256(data)
            ))
        )
    );
    ...
}
```

This may confuse users who are unsure whether to use the signed data in mint() or withdraw().

## Recommendation

It is recommended to create a distinct TYPEHASH for the mint() function instead of reusing the TYPEHASH used for withdraw().

## Status

This issue has been acknowledged by the team.

# Appendix

## Appendix 1 - Files in Scope

This audit covered the following files provided by the client:

| File | SHA-1 hash |
|---|---|
| MultibitERC721.sol | 1e7c630b83d38ff78d61ac4eefe95a4e970c3f7f |
| MultibitNFTBridge.sol | 86089de425d520613efc35d18cdb4b2fd1c7b101 |