Mirco3

Audit Report

Mon Jan 29 2024

contact@scalebit.xyz

https://twitter.com/scalebit_

ScaleBit

Mirco3 Audit Report

1 Executive Summary

1.1 Project Information

Description	Micro3 is a decentralized NFT platform
Type	NFT
Auditors	ScaleBit
Timeline	Fri Dec 22 2023 - Mon Jan 29 2024
Languages	Solidity
Platform	BSC
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/scalebit/Audit_Final_Phase1_With_Bridge
Commits	f1b35bf2043eb1d5fd4ed346c0f40792e7373de3 14eaded4e0f1e729052af97d424404342f212018 a7718170adb988061082ac32cdaf5d58f84de223 0e7cc16aad332e2c9ed3696da9aa62bd2c722e60 f30b55afe4cdb1e79ffb729feb117a3645b809f4

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
ERC7	token/ERC721.sol	1f0a17caf33cde44c47133e85e926587b7646acd
IMNFT	interfaces/IMicroNFT.sol	2e1b22e55a90c1b97ab7d0440c30baca9c020482
IMM	interfaces/IMicroManager.sol	21701764288a34089909104d130791ae863f5a6c
IPO	interfaces/IPriceOracle.sol	8cd2522bfe7b604423fd21e5a6293631b473eafb
MUT	MicroUtility.sol	9b58253b740176b8906f8bd58f68fdf5446e9a99
MBR	MicroBridge.sol	30717e658a97792d378ea106ecf35f0dcb3f7074
MNFT	token/MicroNFT.sol	d29033ca00fbd13cee3f9526eb2aa4f2166f106c
MBCCIP	MicroBridgeCCIP.sol	b54137b82d1a98a46441fbf2abba8b068e56e310

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	22	22	0
Informational	0	0	0
Minor	15	15	0
Medium	5	5	0
Major	2	2	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Mirco3 to identify any potential issues and vulnerabilities in the source code of the Mirco3 smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 22 issues of varying severity, listed below.

ID	Title	Severity	Status
MBC-1	Initializers Could be Front-run	Medium	Fixed
MBC-2	`_ccipReceive()` Lacks Sender Verification.	Medium	Fixed
MBC-3	Set the Maximum Gas Limit	Medium	Fixed
MBC-4	To Accommodate Upgrades, It is Recommended to Make `extraArgs` Mutable.	Medium	Fixed
MBR-1	Potential Transaction Failure due to Gas Limit Check in `_creditTill()` Function	Major	Fixed
MBR-2	The Require Check is Incomplete	Medium	Fixed
MBR-3	Using Bools for Storage Incurs Overhead	Minor	Fixed
MBR-4	Cache Array Length Outside of Loop	Minor	Fixed
MBR-5	For Operations That will not Overflow, You could Use	Minor	Fixed

	Unchecked		
MBR-6	Use Custom Errors	Minor	Fixed
MBR-7	Don't Initialize Variables with Default Value	Minor	Fixed
MBR-8	Long Revert Strings	Minor	Fixed
MBR-9	Functions Guaranteed To Revert When Called By Normal Users Can be Marked Payable	Minor	Fixed
MNF-1	The `totalMintsByAddress` Was Not updated	Major	Fixed
MNF-2	State Variables Should Be Cached in Stack Variables Rather Than Re-reading Them from Storage	Minor	Fixed
MNF-3	`require()` / `revert()` Statements Should Have Descriptive Reason Strings	Minor	Fixed
MBR-10	`++i` Costs Less Gas Than `i++`, Especially When It's Used in For-loops (`--i/i--` Too)	Minor	Fixed
MBR-11	Using Private rather than Public for Constants, Saves Gas	Minor	Fixed
MBR-12	Use `!= 0` instead of `> 0` for Unsigned Integer Comparison	Minor	Fixed
MBR-13	Event is Missing Indexed Fields	Minor	Fixed
MBR-14	Functions Not Used Internally Could be Marked External	Minor	Fixed

MBR-15	Empty Function Body - Consider commenting why	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Mirco3 Smart Contract:

Admin

- Admin can mint NFTs directly for a recipient or multiple recipients through `adminMint()` and `adminMintAirdrop()` .
- Admin can set or update sale details, including timings and profit sharing, using `setSaleDetail()` .
- Admin can change the funds recipient address with `setFundsRecipient()` .
- Admin can finalize the open edition by setting the edition size and sale end time using `finalizeOpenEdition()` .
- Admin can mark an ongoing NFT sale as cancelable through `cancelSaleEdition()` .
- Admin can execute an emergency withdrawal of all ETH from the contract with `emergencyWithdraw()` .
- Admin can withdraw any ERC20 token from the contract using `withdrawToken()` .
- Admin can whitelist a blockchain network for interactions using `whitelistChain` ().
- Admin can denylist a blockchain network from interacting with the contract through `denylistChain()` .

User

- User can buy NFTs during public sales via `purchase()` .
- User can purchase NFTs in presale using `purchasePresale()` .
- User can withdraw their profit share with `withdrawProfitSharing()` .
- Users can transfer single or batch NFTs across different blockchain networks using `sendFrom()` and `sendBatchFrom()` .

4 Findings

MBC-1 Initializers Could be Front-run

Severity: Medium

Status: Fixed

Code Location:

MicroBridgeCCIP.sol#60;

MicroBridge.sol#46;

token/MicroNFT.sol#119

Descriptions:

There is a major security vulnerability involving the Initializers of our smart contract, which potentially allows an attacker to execute these initializers before the contract's normal initialization process, thereby setting their own values or taking ownership of the contract. Additionally, the lack of testing and deployment code in our repository to address this behavior, coupled with the uncertainty of whether contract deployment and initialization occur within the same transaction, further exacerbates the potential risk.

```
function init(bytes memory initPayload) external returns (bool) {
```

Suggestion:

It is recommended to initialize within the deployment code so that deployment and initialization occur in the same transaction.

Resolution:

This issue has been fixed. The client has already added permissions for `init`.

MBC-2 `_ccipReceive()` Lacks Sender Verification.

Severity: Medium

Status: Fixed

Code Location:

MicroBridgeCCIP.sol#224-227

Descriptions:

According to the official documentation's best security practices, `MicroBridgeCCIP._ccipReceive()` needs to validate the sender.

```
function _ccipReceive(Client.Any2EVMMessage memory any2EvmMessage)
    internal
    virtual
    override
{
    // decode and load the toAddress
    (
        bytes memory nftAddressBytes,
        bytes memory toAddressBytes,
        uint256[] memory tokenIds
    )
```

Suggestion:

It is recommended to add an `onlyAllowlisted` modifier.

https://docs.chain.link/ccip/tutorials/programmable-token-transfers#transferring-tokens-and-data-and-pay-in-link

```
/// handle a received message
function _ccipReceive(
    Client.Any2EVMMessage memory any2EvmMessage
)
    internal
    override
    onlyAllowlisted(
        any2EvmMessage.sourceChainSelector,
        abi.decode(any2EvmMessage.sender, (address))
    ) // Make sure source chain and sender are allowlisted
```

Resolution:

This issue has been fixed. The client added a modifier.

MBC-3 Set the Maximum Gas Limit

Severity: Medium

Status: Fixed

Code Location:

MicroBridgeCCIP.sol#367

Descriptions:

According to the documentation, `gasLimit` is the maximum gas that can be consumed on the target chain. However, in the `MicroBridgeCCIP._send()` function, it specifies `minGasToTransferAndStore`

```
 Client.EVM2AnyMessage memory evm2AnyMessage = Client.EVM2AnyMessage({
            receiver: abi.encode(_receiver), // ABI-encoded receiver address
            data: payload, // ABI-encoded string message
            tokenAmounts: new Client.EVMTokenAmount[](0), // Empty array indicating no
tokens are being sent
            extraArgs: Client._argsToBytes(
                Client.EVMExtraArgsV1({
                    gasLimit: minGasToTransferAndStore,
                    strict: false
                })
            ),
```

Suggestion:

It is recommended to set it to the maximum gas limit.

Resolution:

This issue has been fixed. The client set the maximum gas.

MBC-4 To Accommodate Upgrades, It is Recommended to Make `extraArgs` Mutable.

Severity: Medium

Status: Fixed

Code Location:

MicroBridgeCCIP.sol#360-368

Descriptions:

To accommodate future upgrades, `extraArgs` needs to be mutable, but it is fixed in the `MicroBridgeCCIP._send()` function.

```
    Client.EVM2AnyMessage memory evm2AnyMessage = Client.EVM2AnyMessage({
      receiver: abi.encode(_receiver), // ABI-encoded receiver address
      data: payload, // ABI-encoded string message
      tokenAmounts: new Client.EVMTokenAmount[](0), // Empty array indicating no
tokens are being sent
      extraArgs: Client._argsToBytes(
        Client.EVMExtraArgsV1({
          gasLimit: minGasToTransferAndStore,
          strict: false
        })
      ),
```

In fact, the latest version of the client contract, `EVMExtraArgsV1`, only has one element, while the code still contains two elements.

```
// bytes4(keccak256("CCIP EVMExtraArgsV1"));
bytes4 public constant EVM_EXTRA_ARGS_V1_TAG = 0x97a657c9;
struct EVMExtraArgsV1 {
  uint256 gasLimit;
}
```

Suggestion:

It is recommend to make `extraArgs` mutable.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

MBR-1 Potential Transaction Failure due to Gas Limit Check in `_creditTill()` Function

Severity: Major

Status: Fixed

Code Location:

MicroBridge.sol#325

Descriptions:

In the `MicroBridge._creditTill()` function, there is a check `if (gasleft() < minGasToTransferAndStore) break`. If there is not enough gas to process, the current index is stored for the next loop iteration. Currently, on Ethereum and EVM-compatible chains, calls can consume at most 63/64 of the parent's call gas (See EIP-150). This may lead to a situation of insufficient gas.

```
function _creditTill(
      address _nftAddress,
      uint16 _srcChainId,
      address _toAddress,
      uint256 _startIndex,
      uint256[] memory _tokenIds
  ) internal returns (uint256) {
      uint256 i = _startIndex;
      while (i < _tokenIds.length) {
         // if not enough gas to process, store this index for next loop
         if (gasleft() < minGasToTransferAndStore) break;

         IMicroNFT(_nftAddress).bridgeIn(
            _toAddress,
            uint64(_srcChainId),
            _tokenIds[i]
         );
         i++;
      }

      // indicates the next index to send of tokenIds,
      // if i == tokenIds.length, we are finished
```

```
    return i;
  }
```

Suggestion:

It is recommend to change the check to account for this 63/64 rule:

```
if (gasleft() < minGasToTransferAndStore * 64 / 63) break;
```

Resolution:

This issue has been fixed. The client has implemented our recommendations.

MBR-2 The Require Check is Incomplete

Severity: Medium

Status: Fixed

Code Location:

MicroBridge.sol#171

Descriptions:

In the MicroBridge._send() function, the protocol provides the native fee to lzEndpoint for use. The validation in the code, `require(msg.value >= microProtocolFee, "Need more gas fee");` , is incomplete as it does not consider the native fee.

```
//Start to pay the protocol fee
        uint256 microProtocolFee = getMicroFeeWei(_tokenIds.length);

        require(msg.value >= microProtocolFee, "Need more gas fee");

        _payoutMicroFee(_tokenIds.length);
```

Suggestion:

It is recommended to consider the native fee in the require statement.

MBR-3 Using Bools for Storage Incurs Overhead

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#12

Descriptions:

Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from `false` to `true`, after having been `true` in the past. See source.

```
bool private initialized;
```

Suggestion:

It is recommended to use `uint256(1)` and `uint256(2)` for true/false.

MBR-4 Cache Array Length Outside of Loop

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#176

Descriptions:

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

```
for (uint256 i = 0; i < _tokenIds.length; i++) {
```

Suggestion:

It is recommended to cache the array length before entering the loop.

MBR-5 For Operations That will not Overflow, You could Use Unchecked

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#176

Descriptions:

For Operations that will not overflow, you could use unchecked.

```
for (uint256 i = 0; i < _tokenIds.length; i++) {
```

Suggestion:

It is recommended to use Solidity's `unchecked` block to save the overflow checks.

Resolution:

This issue has been fixed. The client used `unchecked`.

MBR-6 Use Custom Errors

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#47

Descriptions:

Instead of using error strings, to reduce deployment and runtime costs, you should use Custom Errors. This would save both deployment and runtime costs.Source

```
require(!initialized, "Already initialized");
```

Suggestion:

It is recommended to use Custom Errors.

Resolution:

This issue has been fixed. The client has implemented custom errors.

MBR-7 Don't Initialize Variables with Default Value

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#176

Descriptions:

Uninitialized variables are assigned with the types' default value. Explicitly initializing a variable with its default value costs unnecessary gas.

```
for (uint256 i = 0; i < _tokenIds.length; i++) {
```

Suggestion:

It is recommended not to use default values to initialize variables.

Resolution:

This issue has been fixed. The client doesn't initialize variables with default Value.

MBR-8 Long Revert Strings

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#348

Descriptions:

The error message string provided in the `require` function is too long and exceeds the EVM limit.

```
require(success, "TransferHelper: ETH_TRANSFER_FAILED");
```

Suggestion:

It is recommended to shorten the error message.

Resolution:

This issue has been fixed. The client has shortened the error message.

MBR-9 Functions Guaranteed To Revert When Called By Normal Users Can be Marked Payable

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#341

Descriptions:

If a function modifier such as onlyOwner is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are CALLVALUE(2),DUP1(3),ISZERO(3),PUSH2(3),JUMPI(10),PUSH1(3),DUP1(3),REVERT(0),JUMPDEST(1),PO which costs an average of about 21 gas per call to the function, in addition to the extra deployment cost.

```
function emergencyWithdraw() external onlyOwner {
```

Suggestion:

It is recommended that functions guaranteed to revert when called by normal users can be marked payable.

MNF-1 The `totalMintsByAddress` Was Not updated

Severity: Major

Status: Fixed

Code Location:

token/MicroNFT.sol#286-359

Descriptions:

In the `purchase` function's code, there is a restriction on the number of NFTs that can be purchased by the same address, capped at `maxSalePurchasePerAddress`. The condition set is that the total of `totalMintsByAddress` plus `quantity`, minus `presaleMintsByAddress`, should not exceed `maxSalePurchasePerAddress`. This means that `totalMintsByAddress` represents the aggregate number of NFTs minted in both purchase and purchasePresale.

```
  if (
    saleConfig.maxSalePurchasePerAddress != 0 &&
    totalMintsByAddress[_msgSender()].add(quantity).sub(
      presaleMintsByAddress[_msgSender()]
    ) >
    saleConfig.maxSalePurchasePerAddress
  ) {
    revert("Purchase_TooManyForAddress");
  }
```

However, in `purchasePresale`, `totalMintsByAddress` is not updated. Consequently, this may prevent users who participated in the presale from purchasing NFTs in the public sale, unless the number they wish to buy exceeds the amount purchased during the presale. Additionally, this could also prevent presale participants from being eligible for rewards.

Suggestion:

It is recommended to update the value of `totalMintsByAddress` within `purchasePresale`.

Resolution:

This issue has been fixed. The client has already updated the `totalMintsByAddress` within the `purchasePresale` function.

MNF-2 State Variables Should Be Cached in Stack Variables Rather Than Re-reading Them from Storage

Severity: Minor

Status: Fixed

Code Location:

token/MicroNFT.sol#557

Descriptions:

In Solidity, state variables ought to be cached in stack variables rather than repeatedly reading them from storage.

```
tokenId = sourceGetChainPrepend(_currentTokenId);
```

Suggestion:

It is recommended to cache state variables in local variables within functions to reduce repetitive reads from storage in Solidity.

MNF-3 `require()` / `revert()` Statements Should Have Descriptive Reason Strings

Severity: Minor

Status: Fixed

Code Location:

token/MicroNFT.sol#465

Descriptions:

In Solidity, it's essential to include descriptive reason strings within `require()` or `revert()` statements.

```
require(
```

Suggestion:

It is recommended to add reason strings to `require()` or `revert()` .

MBR-10 `++i` Costs Less Gas Than `i++`, Especially When It's Used in For-loops (`--i/i--` Too)

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#176

Descriptions:

The gas cost of `++i` is lower than `i++`, particularly when utilized in for-loops (`--i/i--` as well).

```
for (uint256 i = 0; i < _tokenIds.length; i++) {
```

Suggestion:

It is recommended to use `++i` instead of `i++`.

MBR-11 Using Private rather than Public for Constants, Saves Gas

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#11

Descriptions:

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that returns a tuple of the values of all currently-public constants. Saves 3406-3606 gas in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table.

```
uint16 public constant FUNCTION_TYPE_SEND = 1;
1
```

Suggestion:

It is recommended to use private rather than public for constants.

Resolution:

This issue has been fixed. The client used private for constants.

MBR-12 Use `!= 0` instead of `> 0` for Unsigned Integer Comparison

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#54

Descriptions:

When dealing with unsigned integer types, comparisons with `!= 0` are cheaper than with `> 0`.

```
_minGasToTransferAndStore > 0,
```

Suggestion:

It is recommended to use `!= 0` instead of `> 0` for unsigned integer comparison.

MBR-13 Event is Missing Indexed Fields

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#41

Descriptions:

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

```
event CreditStored(bytes32 _hashedPayload, bytes _payload);
```

Suggestion:

It is recommended to evaluate the necessity of indexing fields based on the specific needs of off-chain tools and gas usage. For events with three or more fields where gas consumption isn't critical, consider indexing up to three relevant fields. For events with fewer than three fields, index all available fields to optimize off-chain tool accessibility. Regularly assess and adjust indexing strategies based on the requirements of off-chain tools and gas efficiency.

MBR-14 Functions Not Used Internally Could be Marked External

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#64

Descriptions:

In Solidity, functions that are not internally used within the contract can be more suitably marked as external. This ensures clarity in code and signifies that these functions are intended to be accessed from outside the contract by other contracts or external entities.

```
function estimateSendFee(
function sendFrom(
function sendBatchFrom(
```

Suggestion:

It is recommended to consider marking functions that are not internally used within the contract as external to enhance code readability and explicitly indicate that these functions are meant to be accessed externally. This practice provides clarity and aligns with the intended usage of the functions, making the contract interface more understandable for external interactions.

Resolution:

This issue has been fixed. The client has already changed to external visibility.

MBR-15 Empty Function Body - Consider commenting why

Severity: Minor

Status: Fixed

Code Location:

MicroBridge.sol#44

Descriptions:

In smart contracts, having a function (especially a constructor) with an empty body can cause confusion for readers. Although sometimes a function with an empty body is justified (for example, when only the behavior of the base class constructor is needed), it can leave other developers or auditors puzzled as to why the function is empty and whether any crucial implementation code is missing.

```
constructor(address _lzEndpoint) NonblockingLzApp(_lzEndpoint) {}
```

Suggestion:

It is recommended to add a comment next to the empty function body, explaining why it is empty. For constructors, the comment should clarify that the constructor only calls the base class constructor and that no further initialization logic is necessary.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer