Nexa - CAT ERC Standards

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **June 13th, 2023 - June 27th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/21/2023	István Böhm
0.2	Draft Version	06/27/2023	István Böhm
0.3	Draft Review	06/27/2023	Manuel Garcia
0.4	Draft Review	06/27/2023	Grzegorz Trawinski
0.5	Draft Review	06/27/2023	Piotr Cielas
0.6	Draft Review	06/27/2023	Gabi Urrutia
1.0	Remediation Plan	07/11/2023	István Böhm
1.1	Remediation Plan Review	07/12/2023	Grzegorz Trawinski
1.2	Remediation Plan Review	07/12/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Manuel Garcia	Halborn	Manuel.Diaz@halborn.com
István Böhm	Halborn	Istvan.Bohm@halborn.com
Grzegorz Trawinski	Halborn	Grzegorz.Trawinski@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

The CAT ERC standards enable users to use their ERC20 and ERC721 tokens on multiple chains.

Nexa engaged Halborn to conduct a security assessment on their smart contracts beginning on June 13th, 2023 and ending on June 27th, 2023. The security assessment was scoped to the smart contracts provided in the NEXA-NETWORK/CAT GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided 2 weeks for the engagement and assigned a team of one full-time security engineer to verify the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessments is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks that were addressed and accepted by Nexa. The main ones were the following:

- Use the OpenZeppelin's SafeERC20 wrapper and the safeTransferFrom function to transfer the payment tokens from the callers to the CATERC20Proxy contracts.
- Get the exact received amount of the tokens being transferred by calculating the difference of the token balance before and after the transfer in the CATERC20Proxy contracts.

- Implement an allowlist to only allow tokens compatible with the CATERC20Proxy contracts.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX).
- Static Analysis of security for scoped contract, and imported functions. (Slither).
- Testnet deployment (Foundry, Brownie).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

EXECUTIVE OVERVIEW

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. CAT ERC Standards

- Repository: NEXA-NETWORK/CAT
- Commit ID: 9e677120f3c4a149c8e5549c5e82c19a101c881c
- Smart contracts in scope:
  - contracts/ERC20/CATERC20Proxy.sol
  - contracts/ERC20/CATERC20.sol
  - contracts/ERC20/Structs.sol
  - contracts/ERC20/Getters.sol
  - contracts/ERC20/Setters.sol
  - contracts/ERC20/Governance.sol
  - contracts/ERC20/State.sol
  - contracts/ERC721/CATERC721.sol
  - contracts/ERC721/CATERC721Proxy.sol
  - contracts/ERC721/Structs.sol
  - contracts/ERC721/Getters.sol
  - contracts/ERC721/Setters.sol
  - contracts/ERC721/Governance.sol
  - contracts/ERC721/State.sol
  - contracts/libraries/BytesLib.sol
- Final fix commit ID: 4e08af4308e15f336f04d07bab88439511f5d5b5

Out-of-scope:

- Third-party libraries and dependencies.
- Economic attacks.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	1	9	4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) UNSAFE HANDLING OF ERC20 TRANSFER RESULTS	Critical (9.4)	SOLVED - 07/05/2023
(HAL-02) INCOMPATIBILITY WITH TRANSFER-ON-FEE OR DEFLATIONARY TOKENS	Medium (5.0)	SOLVED - 07/06/2023
(HAL-03) RECIPIENT CHAIN ID IS NOT VALIDATED	Low (2.5)	SOLVED - 07/07/2023
(HAL-04) LACK OF CHAIN FORK VERIFICATION	Low (2.5)	SOLVED - 07/06/2023
(HAL-05) TRANSFER AMOUNTS ARE NOT NORMALIZED	Low (2.5)	SOLVED - 07/11/2023
(HAL-06) SIGNATURES CAN BE REUSED	Low (2.5)	FUTURE RELEASE
(HAL-07) LACK OF EMERGENCY STOP PATTERN	Low (2.5)	RISK ACCEPTED
(HAL-08) MISTAKENLY SENT TOKENS AND ETHER CANNOT BE RECOVERED FROM THE CONTRACTS	Low (2.5)	RISK ACCEPTED
(HAL-09) OWNER CAN RENOUNCE OWNERSHIP	Low (2.0)	RISK ACCEPTED
(HAL-10) SINGLE STEP OWNERSHIP TRANSFER PROCESS	Low (2.0)	RISK ACCEPTED
(HAL-11) CENTRALIZED OPERATION	Low (2.0)	RISK ACCEPTED
(HAL-12) EMITTER CHAIN ID IS NOT ALWAYS VALIDATED	Informational (1.7)	FUTURE RELEASE
(HAL-13) USING REVERT STRINGS INSTEAD OF CUSTOM ERRORS	Informational (0.0)	FUTURE RELEASE
(HAL-14) MISSING NATSPEC COMMENTS	Informational (0.0)	FUTURE RELEASE
(HAL-15) UNECESSARY LIBRARY IMPORTS	Informational (0.0)	SOLVED - 07/07/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) UNSAFE HANDLING OF ERC20 TRANSFER RESULTS - CRITICAL(9.4)

Description:

It was identified that the bridgeOut function in the CATERC20Proxy contract does not verify the return value of the transferFrom function call, which facilitates the token transfer from the caller to the contract. Some tokens (e.g., ZRX) return false instead of reverting in the event of failure or insufficient balance. Setting up the contract with such tokens could enable users to transfer funds to other chains without transferring any tokens to the CATERC20Proxy contract. In these cases, by exploiting this vulnerability, a malicious user can extract the funds from the contract by depositing a huge amount with actual funds, then withdrawing it from the contract.

It was also identified that the CATERC20Proxy contract indirectly uses the ERC20 interface for the transferFrom function calls, and expects the function to return a boolean value. However, the transferFrom function of some ERC-20 tokens (e.g., USDT, BNB) does not return any values. If the CATERC20Proxy contract is configured with such tokens, then the transferFrom calls reverts, preventing the users from depositing or withdrawing.

Code Location:

The SafeERC20 wrapper is not used for transferring tokens from the depositor in the bridgeOut function of the CATERC20Proxy contract:

Listing 1: contracts/ERC20/CATERC20Proxy.sol

```
        // Transfer in contract and lock the tokens in this
    ↳ contract
        nativeAsset().transferFrom(_msgSender(), address(this),
    ↳ normalizedAmount);
```

The transferFrom function of some tokens (e.g., ZRX) return false instead of reverting in the event of failure or insufficient balance:

Listing 2: The implementation of the transferFrom function of the ZRX token

```
    function transferFrom(address _from, address _to, uint _value) returns (bool) {
        if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value && balances[_to] + _value >= balances[_to]) {
            balances[_to] += _value;
            balances[_from] -= _value;
            allowed[_from][msg.sender] -= _value;
            Transfer(_from, _to, _value);
            return true;
        } else { return false; }
    }
```

The nativeAsset in the CATERC20Proxy contract uses the IERC20Extended interface:

Listing 3: contracts/ERC20/Getters.sol

```
    function nativeAsset() public view returns (IERC20Extended) {
        return IERC20Extended(_state.nativeAsset);
    }
```

The IERC20Extended interface extends the IERC20 interface:

Listing 4: contracts/interfaces/IERC20Extended.sol

```
interface IERC20Extended is IERC20 {
    function decimals() external view returns (uint8);
}
```

The IERC20 interface excepts a return value from the transferFrom function:

Listing 5: @openzeppelin/contracts/token/ERC20/IERC20.sol

```
    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) external returns (bool);
```

However, the transferFrom function of some tokens (e.g., USDT) does not return any value, which would cause the contract to revert:

Listing 6: The ERC20 Interface Used By USDT

```
contract ERC20 is ERC20Basic {
    function allowance(address owner, address spender) public constant returns (uint);
    function transferFrom(address from, address to, uint value) public;
    function approve(address spender, uint value) public;
    event Approval(address indexed owner, address indexed spender, uint value);
}
```

Proof of Concept:

1. A CATERC20Proxy is deployed on the Mainnet, configured with the ZRX token.
2. A malicious user exploits the lack of SafeERC20 wrapper and deposits tokens to the bridge without actually depositing any funds.
3. The deposit is successful and the LogMessagePublished event is emitted because the ZRX token returns false instead of reverting, and the CATERC20Proxy contracts do not check the return value.
4. The malicious user then withdraws their token balance from the CATERC20Proxy contract.
5. The deficit of funds prevents other users from withdrawing their ZRX tokens from the CATERC20Proxy contract.

The following example is a demonstration of a successful ZRX deposit with alice, who holds zero ZRX tokens:

```
>>> zrx
<ERC20 Contract '0xE41d2489571d322189246DaFA5ebDe1F4699F498'>
>>> zrx.balanceOf(alice)
0
>>> amount = 1000 * 10**18
recipientChain = 2  # ethereum
recipient = address_to_bytes32(alice.address)
nonce = 123123123
tx1 = caterc20proxy.bridgeOut(amount, recipientChain, recipient, nonce, {'from': alice})
Transaction sent: 0x4b3b5a2274a574e070f6eecd1a24438e3f4baed5827886d15319773feb150550
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 0
  CATERC20Proxy.bridgeOut confirmed   Block: 17570860   Gas used: 262244 (3.90%)

>>> tx1.info()
Transaction was Mined
---------------------
Tx Hash: 0x4b3b5a2274a574e070f6eecd1a24438e3f4baed5827886d15319773feb150550
From: 0x622b919CBDEa0c3B00f5040cFe8f43349fad670B
To: 0x8C4904ad03eA3C5C2b1495D2bDF9833B1dAE67E7
Value: 0
Function: CATERC20Proxy.bridgeOut
Block: 17570860
Gas Used: 262244 / 6721975 (3.9%)

Events In This Transaction
--------------------------
├── MockWormhole (0x017bE397294b59d80A92cA19985826d9C47f12b3)
│   └── LogMessagePublished
│       ├── sender: 0x8C4904ad03eA3C5C2b1495D2bDF9833B1dAE67E7
│       ├── sequence: 1
│       ├── nonce: 123123123
│       ├── payload: 0x00000000000000000000000000000000000000000000003635c9adc5dea000000000000000000000000000000008c4904ad03e
a3c5c2b1495d2bdf9833b1dae67e70001000000000000000000000000622b919cbdea0c3b00f5040cfe8f43349fad670b0002
│       └── consistencyLevel: 1
│
└── CATERC20Proxy (0x8C4904ad03eA3C5C2b1495D2bDF9833B1dAE67E7)
    └── bridgeOutEvent
        ├── tokenAmount: 1000000000000000000000
        ├── fromChain: 1
        ├── toChain: 2
        ├── fromAddress: 0x000000000000000000000000622b919cbdea0c3b00f5040cfe8f43349fad670b
        └── toAddress: 0x000000000000000000000000622b919cbdea0c3b00f5040cfe8f43349fad670b
```

1. A CATERC20Proxy is deployed on the Mainnet, configured with the USDT token.
2. The users try to deposit. However, they cannot deposit because the BridgeOut function reverts every time they call it.

The following example demonstrates the revert of the BridgeOut function when the contract was configured with the USDT token:

```
>>> caterc20proxy.nativeAsset() == usdt
True
>>> amount = 1000 * 10**6
recipientChain = 2
recipient = address_to_bytes32(alice.address)
nonce = 113331
tx1 = caterc20proxy.bridgeOut(amount, recipientChain, recipient, nonce, {'from': alice})

Transaction sent: 0x5a1f60bf92c5a163e44a75b4d12dcbfbbf5c4c70fd388818d83af840a696c299
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 2
  CATERC20Proxy.bridgeOut confirmed (reverted)   Block: 17571444   Gas used: 72262 (1.08%)

>>> tx1.info()
Transaction was Mined (reverted)
---------------------
Tx Hash: 0x5a1f60bf92c5a163e44a75b4d12dcbfbbf5c4c70fd388818d83af840a696c299
From: 0x622b919CBDEa0c3B00f5040cFe8f43349fad670B
To: 0x8C4904ad03eA3C5C2b1495D2bDF9833B1dAE67E7
Value: 0
Function: CATERC20Proxy.bridgeOut
Block: 17571444
Gas Used: 72262 / 6721975 (1.1%)

Events In This Transaction
--------------------------
└── Tether USD (0xdAC17F958D2ee523a2206206994597C13D831ec7)
    └── Transfer
        ├── from: 0x622b919CBDEa0c3B00f5040cFe8f43349fad670B
        ├── to: 0x8C4904ad03eA3C5C2b1495D2bDF9833B1dAE67E7
        └── value: 1000000000

>>> tx1.error()
Trace step 1584, program counter 6042:
  File "contracts/ERC20/CATERC20Proxy.sol", line 64, in CATERC20Proxy.bridgeOut:
    );

    // Transfer in contract and lock the tokens in this contract
    nativeAsset().transferFrom(_msgSender(), address(this), normalizedAmount);

    CATERC20Structs.CrossChainPayload memory transfer = CATERC20Structs.CrossChainPayload({
        amount: normalizedAmount,
>>>
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:H/Y:N/R:N/S:U (9.4)

Recommendation:

It is recommended to use OpenZeppelin's SafeERC20 wrapper and the safeTransferFrom function to transfer the payment tokens from the callers to the contract.

Remediation Plan:

SOLVED: The Nexa team solved the issue in commit 912abf2 by using the OpenZeppelin's SafeERC20 wrapper and the safeTransferFrom function to transfer the payment tokens from the callers to the contract.

4.2 (HAL-02) INCOMPATIBILITY WITH TRANSFER-ON-FEE OR DEFLATIONARY TOKENS - MEDIUM (5.0)

Description:

It was identified that the bridgeOut function in the CATERC20Proxy contract assumes that the transferFrom call transfers the full amount of tokens. This may not be true if the tokens being transferred are transfer-on-fee tokens, causing the received amount to be lesser than the accounted amount, resulting in an increasing deficit of funds in the CATERC20Proxy contract. If the contract does not have the full token amounts, the following bridgeIn function calls may revert because of insufficient funds, preventing users from withdrawing their full balances.

The CATERC20Proxy contract also assumes that its native asset token balances does not change over time without any token transfers, which not be true if the native asset is deflationary/inflationary/rebasing tokens. For example, the supply of AMPL (Ampleforth) tokens automatically increases or decreases every 24 hours to maintain the AMPL target price. In these cases, the contract might not have the full token amounts and the following bridgeIn function calls may revert because of insufficient funds.

Code Location:

Listing 7: contracts/ERC20/CATERC20Proxy.sol (Lines 64,67)

```
        // Transfer in contract and lock the tokens in this
↳ contract
        nativeAsset().transferFrom(_msgSender(), address(this),
↳ normalizedAmount);

        CATERC20Structs.CrossChainPayload memory transfer =
↳ CATERC20Structs.CrossChainPayload({
            amount: normalizedAmount,
            tokenAddress: tokenAddress,
```

```
            tokenChain: tokenChain,
            toAddress: recipient,
            toChain: recipientChain
        });
```

Proof of Concept:

Using a transfer-on-fee token:

1. A CATERC20Proxy is deployed on the Mainnet configured with a transfer-on-fee token.
2. The CATERC20Proxy contract is incorrectly accounting for the deposits.
3. The users withdraw more funds than their actual deposit, resulting in an increasing deficit of funds in the CATERC20Proxy contract.
4. The deficit of funds prevents other users from withdrawing their tokens from the CATERC20Proxy contract.

Using a deflationary token:

1. A CATERC20Proxy is deployed on the Mainnet configured with a deflationary token.
2. The CATERC20Proxy contract is incorrectly accounting for the deposits as the token balance of the contract decreases over time.
3. The users withdraw their original deposit amount, resulting in an increasing deficit of funds in the CATERC20Proxy contract.
4. The deficit of funds prevents other users from withdrawing their tokens from the CATERC20Proxy contract.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to get the exact received amount of the tokens being transferred by calculating the difference of the token balance before and after the transfer to handle transfer-on-fee tokens.

It is recommended to state in the documentation that the contracts are incompatible with deflationary/inflationary/rebasing tokens.

Remediation Plan:

SOLVED: The Nexa team solved the issue in commits 970a9d6, 12163c3 and 23e9f7e by calculating the exact received amounts. A comment with a disclaimer was added in the CATERC20Proxy contract to warn users about the incompatibility with deflationary or inflationary tokens.

4.3 (HAL-03) RECIPIENT CHAIN ID IS NOT VALIDATED - LOW (2.5)

Description:

Users can bridge tokens to different chains using the BridgeOut functions of the CATERC20, CATERC20Proxy, CATERC721 and CATERC721Proxy contracts. However, it was identified that these functions do not validate the recipient recipientChain parameter, containing the recipient chain ID. This presents a risk, as users may inadvertently send tokens to chains where the requisite bridge contracts are absent (e.g., the users confuse the Wormhole chain ID with the network chain ID). In such instances, the users would not be able to withdraw, and their tokens would remain locked in the contract until the requisite bridge contracts are not deployed.

Code Location:

The recipientChain parameters are not validated in the bridgeOut functions. For example, in the CATERC20 contract:

Listing 8: contracts/ERC20/CATERC20.sol

```
    function bridgeOut(
        uint256 amount,
        uint16 recipientChain,
        bytes32 recipient,
        uint32 nonce
    ) external payable returns (uint64 sequence) {
        require(isInitialized() == true, "Not Initialized");

        uint256 fee = wormhole().messageFee();
        require(msg.value >= fee, "Not enough fee provided to publish message");
```

Note that an allowlist was implemented for the bridgeIn function.

Listing 9: contracts/ERC20/CATERC20.sol (Line 102)

```
100         require(
101             bytesToAddress(vm.emitterAddress) == address(this) ||
102                 tokenContracts(vm.emitterChainId) == vm.
    ↳ emitterAddress,
103             "Invalid Emitter"
104         );
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:F/S:U (2.5)

Recommendation:

It is recommended to create an allowlist and use it inside the bridgeOut function to only allow token transfers to supported chains.

Remediation Plan:

SOLVED: The Nexa team solved the issue in commit 252f468 by allowing token transfers to supported chains only.

4.4 (HAL-04) LACK OF CHAIN FORK VERIFICATION - LOW (2.5)

Description:

It was identified that the bridgeIn and bridgeOut functions of the CATERC20 , CATERC20Proxy, CATERC721 and CATERC721Proxy contracts do not check whether the current blockchain is a fork or not. Executing the bridgeIn or the bridgeOut functions on a forked chain may result in the users receiving the transferred tokens both from the original and on the forked chains.

Code Location:

For example, in the CATERC20 contract, the bridgeIn function does not check whether the current blockchain is a fork or not:

Listing 10: contracts/ERC20/CATERC20.sol

```
    function bridgeIn(bytes memory encodedVm) external returns (bytes memory) {
        require(isInitialized() == true, "Not Initialized");

        (IWormhole.VM memory vm, bool valid, string memory reason) = wormhole().parseAndVerifyVM(
            encodedVm
        );
        require(valid, reason);
        require(
            bytesToAddress(vm.emitterAddress) == address(this) ||
                tokenContracts(vm.emitterChainId) == vm.emitterAddress,
            "Invalid Emitter"
        );

        CATERC20Structs.CrossChainPayload memory transfer = decodeTransfer(vm.payload);
        address transferRecipient = bytesToAddress(transfer.toAddress);

```

```
        require(!isTransferCompleted(vm.hash), "transfer already
↳ completed");
        setTransferCompleted(vm.hash);

        require(transfer.toChain == wormhole().chainId(), "invalid
↳ target chain");
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:H/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to save the chain ID in the contract and verify in the bridgIn function that its value matches with block.chainid.

Remediation Plan:

SOLVED: The Nexa team solved the issue in commit ad7853f by checking the chain ID.

4.5 (HAL-05) TRANSFER AMOUNTS ARE NOT NORMALIZED - LOW (2.5)

Description:

It was identified that the bridgeOut and bridgeIn functions of the CATERC20 and CATERC20Proxy contracts do not normalize and denormalize the transfer amounts of the cross-chain messages. Since transfer amounts are not normalized and denormalized, the users lose funds if the bridges sending and receiving tokens use different decimals.

Note that on some blockchains, tokens are limited to have a maximum of 8 decimal places, which may require the transfer amounts to be normalized. However, the current implementations of the contracts are intended to be only used on EVM compatible chains.

Code Location:

Instead of calculating the normalizedAmount in the bridgeOut function, the transfer amount is rounded by first normalizing and then denormalizing it. This operation is used in other contracts to avoid any loss of deposited funds due to the decimal shift, and the actual normalization is missing from the function. For example, in the CATERC20 contract:

Listing 11: contracts/ERC20/CATERC20.sol

```
        uint256 normalizedAmount = deNormalizeAmount(
            normalizeAmount(amount, decimals()),
            decimals()
        );
        _burn(_msgSender(), normalizedAmount);

        CATERC20Structs.CrossChainPayload memory transfer = CATERC20Structs.CrossChainPayload({
            amount: normalizedAmount,
            tokenAddress: tokenAddress,
            tokenChain: tokenChain,
            toAddress: recipient,
```

```
            toChain: recipientChain
        });
```

The same issue can be identified in the nativeAmount calculation of the bridgeIn functions. For example, in the CATERC20 contract:

Listing 12: contracts/ERC20/CATERC20.sol

```
        uint256 nativeAmount = deNormalizeAmount(
            normalizeAmount(transfer.amount, decimals()),
            decimals()
        );

        _mint(transferRecipient, nativeAmount);
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to examine whether there is a need to normalize the token transfer amounts. If normalization is not necessary, it is recommended to remove the unnecessary code from the bridgeOut functions.

Remediation Plan:

SOLVED: The Nexa team solved the issue in commits 0cbbb8c and 4e08af4 by normalizing the transfer amounts.

4.6 (HAL-06) SIGNATURES CAN BE REUSED - LOW (2.5)

Description:

In the CATERC20Governance and CATERC721Governance contracts, custodians authorized by signatures can also execute the registerChain, registerChains and updateFinality functions. The signature can only be used by the specified custodian until its validity period. However, it was identified that signatures can be reused arbitrarily times until they are not expired. It is also impossible to revoke the validity of the signatures. If the custodian accounts get compromised, the attacker can execute the registerChain, registerChains and updateFinality functions until the signature is not expired.

Code Location:

The onlyOwnerOrOwnerSignature modifier does not check whether the signature has been used or revoked. For example, in the CATERC20Governance contract:

Listing 13: contracts/ERC20/Governance.sol (Lines 68-73)

```
    modifier onlyOwnerOrOwnerSignature(
        CATERC20Structs.SignatureVerification memory signatureArguments
    ) {
        if (_msgSender() == owner()) {
            _;
        } else {
            bytes32 encodedHashData = prefixed(
                keccak256(
                    abi.encodePacked(signatureArguments.custodian, signatureArguments.validTill)
                )
            );
            require(signatureArguments.custodian == _msgSender(), "custodian can call only");
```

```
            require(signatureArguments.validTill > block.timestamp
↳ , "signed transaction expired");
            require(
                verifySignature(encodedHashData,
↳ signatureArguments.signature, owner()),
                "unauthorized signature"
            );
            _;
        }
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to extend the onlyOwnerOrOwnerSignature modifier to also check if the signature has been used or revoked.

Remediation Plan:

PENDING: The Nexa team updated the code to prevent signature reuse in commit ebc0400. The Nexa team will implement the signature revoke feature in a future release.

4.7 (HAL-07) LACK OF EMERGENCY STOP PATTERN - LOW (2.5)

Description:

The CATERC20, CATERC20Proxy, CATERC721 and CATERC721Proxy contracts do not implement any kind of emergency stop pattern. Such a pattern allows the project team to pause crucial functionalities while being in a state of emergency, e.g., being under an adversary attack. The most prevalent application of the emergency stop pattern is the Pausable contract from the OpenZeppelin's library.

If the emergency stop pattern is not used, functions such as bridgeOut, bridgeIn cannot be temporarily disabled.

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:H/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to use the emergency stop pattern in the contracts.

Remediation Plan:

RISK ACCEPTED: The Nexa team made a business decision to accept the risk of this finding.

4.8 (HAL-08) MISTAKENLY SENT TOKENS AND ETHER CANNOT BE RECOVERED FROM THE CONTRACTS - LOW (2.5)

Description:

It was identified that the CATERC20, CATERC20Proxy, CATERC721 and CATERC721Proxy contracts are missing functions to sweep/recover accidental token and Ether transfers. Mistakenly sent tokens and Ether are locked in the contracts indefinitely.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to add a function to recover accidental token and Ether transfers.

Remediation Plan:

RISK ACCEPTED: The Nexa team made a business decision to accept the risk of this finding.

4.9 (HAL-09) OWNER CAN RENOUNCE OWNERSHIP - LOW (2.0)

Description:

The owner of the contract is usually the account that deploys the contract. As a result, the owner can perform some privileged functions. In the CATERC20, CATERC20Proxy, CATERC721 and CATERC721Proxy contracts, the renounceOwnership function can be used to renounce the owner permission. Renouncing ownership would result in the contract having no owner, eliminating the ability to call privileged functions.

Code Location:

The contracts are inherited from the Ownable contract, and therefore, their ownership can be renounced with the renounceOwnership function:

Listing 14: openzeppelin-contracts/contracts/access/Ownable.sol

```
    function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(address(0));
    }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (2.0)

Recommendation:

It is recommended that the owner cannot call renounceOwnership without first transferring ownership to another address. In addition, if a multi-signature wallet is used, the call to the renounceOwnership function should be confirmed for two or more users.

Remediation Plan:

RISK ACCEPTED: The Nexa team made a business decision to accept the risk of this finding.

4.10 (HAL-10) SINGLE STEP OWNERSHIP TRANSFER PROCESS - LOW (2.0)

Description:

Ownership of the contracts can be lost as the CATERC20, CATERC20Proxy, CATERC721 and CATERC721Proxy contracts are inherited from the Ownable contract and their ownership can be transferred in a single-step process. The address the ownership is changed to should be verified to be active or willing to act as the owner.

Code Location:

The contracts are inherited from the Ownable contract, and therefore, their ownership can be transferred in a single step process using the transferOwnership function:

Listing 15: openzeppelin-contracts/contracts/access/Ownable.sol

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _transferOwnership(newOwner);
}
```

Listing 16: openzeppelin-contracts/contracts/access/Ownable.sol

```
function _transferOwnership(address newOwner) internal virtual {
    address oldOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(oldOwner, newOwner);
}
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (2.0)

Recommendation:

It is recommended to use the Ownable2Step library or similar implementations to split the current ownership transfer process into two steps.

Remediation Plan:

RISK ACCEPTED: The Nexa team made a business decision to accept the risk of this finding.

4.11 (HAL-11) CENTRALIZED OPERATION - LOW (2.0)

Description:

Users are able to deposit their funds using the CATERC20Proxy and CATERC721Proxy contracts and transfer them to other chains using the CATERC20Proxy and CATERC721Proxy contracts.

However, it is noted that the owner of the contracts or the custodian users authorized by signatures can mint tokens on the other chains or configure trust relations between the bridges using the register chain functions. If malicious users compromise any of these high-privilege accounts, they can be used to withdraw the funds out of the contracts by creating fake tokens or registering malicious contracts.

Code Location:

Example mint and registerChain functions from the CATERC20 and CATERC20Governance contracts:

Listing 17: contracts/ERC20/CATERC20.sol

```
    function mint(address recipient, uint256 amount) public onlyOwner {
        require(mintedSupply() + amount <= maxSupply(), "MAX SUPPLY REACHED");
        setMintedSupply(mintedSupply() + amount);
        _mint(recipient, amount);
    }
```

Listing 18: contracts/ERC20/Governance.sol

```
    function registerChain(
        uint16 chainId,
        bytes32 tokenContract,
        CATERC20Structs.SignatureVerification memory signatureArguments
```

```
    ) public onlyOwnerOrOwnerSignature(signatureArguments) {
        setTokenImplementation(chainId, tokenContract);
    }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (2.0)

Recommendation:

It is recommended to employ multi-signature access for high-privileged accounts. It is recommended to monitor the operation of the contracts and use an emergency pause function to stop them if any suspicious behavior is detected.

Remediation Plan:

RISK ACCEPTED: The Nexa team made a business decision to accept the risk of this finding.

4.12 (HAL-12) EMITTER CHAIN ID IS NOT ALWAYS VALIDATED - INFORMATIONAL (1.7)

Description:

Users can receive the bridged tokens using the bridgeIn functions of the CATERC20, CATERC20Proxy, CATERC721 and CATERC721Proxy contracts. However, it was identified that these functions do not validate the emitter's chain ID if the emitter's address matches the destination bridge contract's address.

On EVM compatible chains, it is not possible to get the same address without having the private key of the contract's deployer, and during our assessment, we did not identify any exploitable scenario. However, in the case that Wormhole supports a blockchain in the future where it is possible to generate an address that matches the bridge contract's address without having the required private key, then that blockchain could be used by an attacker to bypass the check and create authentic cross-chain messages.

Code Location:

The emitter's chain ID is not checked if the emitter's address is the same as the contract's address:

Listing 19: contracts/ERC20/CATERC20.sol (Line 101)

```
100     require(
101         bytesToAddress(vm.emitterAddress) == address(this) ||
102             tokenContracts(vm.emitterChainId) == vm.emitterAddress
    ↳ ,
103         "Invalid Emitter"
104     );
```

Note that the bytesToAddress function used in the verification reverts if the address is longer than 20 bytes. This prevents verifying addresses that are longer than the addresses used in EVM compatible chains.

Listing 20: contracts/ERC20/Getters.sol

```
    function bytesToAddress(bytes32 b) public pure returns (
↳ address) {
        require(bytes12(b) == 0, "invalid EVM address");
        return address(uint160(uint256(b)));
    }
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (1.7)

Recommendation:

It is recommended to follow best practices and add a check to the bridgeIn functions to verify that the emitter's chain ID is supported by the contracts.

Remediation Plan:

PENDING: The Nexa team will address this finding in a future release.

4.13 (HAL-13) USING REVERT STRINGS INSTEAD OF CUSTOM ERRORS - INFORMATIONAL (0.0)

Description:

Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. If the revert string uses strings to provide additional information about failures (e.g. `require(msg.value >= fee, "Not enough fee provided to publish message");`), but they are rather expensive, especially when it comes to deploying cost, and it is difficult to use dynamic information in them.

Gas Consumption Benchmark Tests:

- Deployment gas cost of the CATERC20 contract using revert strings: 2869498

```
caterc20 = deployer.deploy(CATERC20, name, symbol, decimal)
Transaction sent: 0x528cd474a4d29f2d0016de6f8550c0ff92a17772289254961757a331188f5dad
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 9
  CATERC20.constructor confirmed   Block: 17569370   Gas used: 2869498 (42.69%)
  CATERC20 deployed at: 0x8C4904ad03eA3C5C2b1495D2bDF9833B1dAE67E7
```

- Deployment gas cost of the CATERC20 contract using custom errors: 2798496

```
caterc20 = deployer.deploy(CATERC20, name, symbol, decimal)
Transaction sent: 0xbc95474ac765b2a8d140a6fe66e38fa717f819e824f9135f39b0f8f50edf8084
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 9
  CATERC20.constructor confirmed   Block: 17569406   Gas used: 2798496 (41.63%)
  CATERC20 deployed at: 0x8C4904ad03eA3C5C2b1495D2bDF9833B1dAE67E7
```

Difference: 71002

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to implement custom errors instead of reverting strings.

An example implementation of the initialization checks using custom errors:

Listing 21: Using Custom Errors

```
error ErrNotInitialized();

function bridgeOut(
    uint256 amount,
    uint16 recipientChain,
    bytes32 recipient,
    uint32 nonce
) external payable returns (uint64 sequence) {
    // require(isInitialized() == true, "Not Initialized");
    if (isInitialized() == false) revert ErrNotInitialized();
```

Remediation Plan:

PENDING: The Nexa team will address this finding in a future release.

4.14 (HAL-14) MISSING NATSPEC COMMENTS - INFORMATIONAL (0.0)

Description:

Several contract functions are missing NatSpec comments. Since **NatSpec** is an important part of the code documentation, this affects the understandability, auditability, and usability of the code.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider adding full **NatSpec** comments so that all the functions are fully documented across all the codebase.

Remediation Plan:

PENDING: The Nexa team will address this finding in a future release.

4.15 (HAL-15) UNECESSARY LIBRARY IMPORTS - INFORMATIONAL (0.0)

Description:

It was identified that the CATERC20, CATERC20Proxy, CATERC721, CATERC721Proxy, CATERC20Governance and CATERC721Governance contracts do not utilize the BytesLib library, as it is only used in the CATERC20Getters and CATERC721Getters contracts. Removing these unused libraries can lead to gas savings and reduced complexity.

Code Location:

contracts/ERC20/CATERC20.sol
- Line 16 using BytesLib for bytes;

contracts/ERC20/CATERC20Proxy.sol
- Line 14 using BytesLib for bytes;

contracts/ERC20/Governance.sol
- Line 8 import "../libraries/BytesLib.sol";
- Line 17 using BytesLib for bytes;

contracts/ERC721/CATERC721.sol
- Line 13 import "../libraries/BytesLib.sol";
- Line 31 using BytesLib for bytes;

contracts/ERC721/CATERC721Proxy.sol
- Line 10 import "../libraries/BytesLib.sol";
- Line 19 using BytesLib for bytes;

contracts/ERC721/Governance.sol
- Line 8 import "../libraries/BytesLib.sol";
- Line 17 using BytesLib for bytes;

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider reviewing the contracts and removing any unnecessary libraries from them.

Remediation Plan:

SOLVED: The Nexa team solved the issue in commit d287288 by removing the unnecessary libraries.

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

contracts/ERC20/CATERC20.sol

Slither results for CATERC20.sol	
Finding	**Impact**
CATERC20.initialize(uint16,address,uint8,uint256).chainId (contracts/ERC20/CATERC20.sol#24) shadows: - CATERC20Getters.chainId() (contracts/ERC20/Getters.sol#23-25) (function)	Low
CATERC20Governance.registerChain(uint16,bytes32,CATERC20Structs .SignatureVerification).chainId (contracts/ERC20/Governance.sol#80) shadows: - CATERC20Getters.chainId() (contracts/ERC20/Getters.sol#23-25) (function)	Low
CATERC20.initialize(uint16,address,uint8,uint256).maxSupply (contracts/ERC20/CATERC20.sol#27) shadows: - CATERC20Getters.maxSupply() (contracts/ERC20/Getters.sol#43-45) (function)	Low
CATERC20.initialize(uint16,address,uint8,uint256).wormhole (contracts/ERC20/CATERC20.sol#25) shadows: - CATERC20Getters.wormhole() (contracts/ERC20/Getters.sol#19-21) (function)	Low

Finding	Impact
CATERC20Governance.updateFinality(uint8,CATERC20Structs .SignatureVerification).finality (contracts/ERC20/Governance.sol#100) shadows: - CATERC20Getters.finality() (contracts/ERC20/Getters.sol#35-37) (function)	Low
CATERC20Governance.registerChains(uint16[],bytes32[],CATERC20Structs .SignatureVerification).chainId (contracts/ERC20/Governance.sol#88) shadows: - CATERC20Getters.chainId() (contracts/ERC20/Getters.sol#23-25) (function)	Low
CATERC20.initialize(uint16,address,uint8,uint256).finality (contracts/ERC20/CATERC20.sol#26) shadows: - CATERC20Getters.finality() (contracts/ERC20/Getters.sol#35-37) (function)	Low
Reentrancy in CATERC20.bridgeOut(uint256,uint16,bytes32,uint32) (contracts/ERC20/CATERC20.sol#52-91): External calls: - sequence = wormhole().publishMessage{value: msg.value}(nonce,encodeTransfer(transfer),finality()) (contracts/ERC20/CATERC20.sol#78-82) Event emitted after the call(s): - bridgeOutEvent(amount,tokenChain,recipientChain,addressToBytes(_-msgSender()),recipient) (contracts/ERC20/CATERC20.sol#84-90)	Low
End of table for CATERC20.sol	

contracts/ERC20/CATERC20Proxy.sol

Slither results for CATERC20Proxy.sol	
Finding	**Impact**
CATERC20Proxy.bridgeOut(uint256,uint16,bytes32,uint32) (contracts/ERC20/CATERC20Proxy.sol#46-87) ignores return value by nativeAsset().transferFrom(_-msgSender(),address(this),normalizedAmount) (contracts/ERC20/CATERC20Proxy.sol#64)	High

Finding	Impact
CATERC20Proxy.bridgeIn(bytes) (contracts/ERC20/CATERC20Proxy.sol#89-121) ignores return value by nativeAsset().transfer(transferRecipient,nativeAmount) (contracts/ERC20/CATERC20Proxy.sol#116)	High
CATERC20Governance.registerChain(uint16,bytes32,CATERC20Structs .SignatureVerification).chainId (contracts/ERC20/Governance.sol#80) shadows: - CATERC20Getters.chainId() (contracts/ERC20/Getters.sol#23-25) (function)	Low
CATERC20Proxy.initialize(uint16,address,address,uint8).wormhole (contracts/ERC20/CATERC20Proxy.sol#23) shadows: - CATERC20Getters.wormhole() (contracts/ERC20/Getters.sol#19-21) (function)	Low
CATERC20Governance.updateFinality(uint8,CATERC20Structs .SignatureVerification).finality (contracts/ERC20/Governance.sol#100) shadows: - CATERC20Getters.finality() (contracts/ERC20/Getters.sol#35-37) (function)	Low
CATERC20Governance.registerChains(uint16[],bytes32[],CATERC20Structs .SignatureVerification).chainId (contracts/ERC20/Governance.sol#88) shadows: - CATERC20Getters.chainId() (contracts/ERC20/Getters.sol#23-25) (function)	Low
CATERC20Proxy.initialize(uint16,address,address,uint8).chainId (contracts/ERC20/CATERC20Proxy.sol#21) shadows: - CATERC20Getters.chainId() (contracts/ERC20/Getters.sol#23-25) (function)	Low
CATERC20Proxy.initialize(uint16,address,address,uint8).finality (contracts/ERC20/CATERC20Proxy.sol#24) shadows: - CATERC20Getters.finality() (contracts/ERC20/Getters.sol#35-37) (function)	Low

AUTOMATED TESTING

Finding	Impact
Reentrancy in CATERC20Proxy.bridgeOut(uint256,uint16,bytes32,uint32) (contracts/ERC20/CATERC20Proxy.sol#46-87): External calls: - nativeAsset().transferFrom(_-msgSender(),address(this),normalizedAmount) (contracts/ERC20/CATERC20Proxy.sol#64) - sequence = wormhole().publishMessage{value: msg.value}(nonce,encodeTransfer(transfer),finality()) (contracts/ERC20/CATERC20Proxy.sol#74-78) External calls sending eth: - sequence = wormhole().publishMessage{value: msg.value}(nonce,encodeTransfer(transfer),finality()) (contracts/ERC20/CATERC20Proxy.sol#74-78) Event emitted after the call(s): - bridgeOutEvent(amount,tokenChain,recipientChain,addressToBytes(_-msgSender()),recipient) (contracts/ERC20/CATERC20Proxy.sol#80-86)	Low
Reentrancy in CATERC20Proxy.bridgeIn(bytes) (contracts/ERC20/CATERC20Proxy.sol#89-121): External calls: - nativeAsset().transfer(transferRecipient,nativeAmount) (contracts/ERC20/CATERC20Proxy.sol#116) Event emitted after the call(s): - bridgeInEvent(nativeAmount,transfer.tokenChain,transfer.toChain, transfer.toAddress) (contracts/ERC20/CATERC20Proxy.sol#118)	Low
End of table for CATERC20Proxy.sol	

contracts/ERC20/Structs.sol

Slither did not identify any vulnerabilities in the contract.

contracts/ERC20/Getters.sol

Slither results for Getters.sol	
Finding	**Impact**
CATERC20State._state (contracts/ERC20/State.sol#50) is never initialized. It is used in: - CATERC20Getters.isTransferCompleted(bytes32) (contracts/ERC20/Getters.sol#15-17) - CATERC20Getters.wormhole() (contracts/ERC20/Getters.sol#19-21) - CATERC20Getters.chainId() (contracts/ERC20/Getters.sol#23-25) - CATERC20Getters.evmChainId() (contracts/ERC20/Getters.sol#27-29) - CATERC20Getters.tokenContracts(uint16) (contracts/ERC20/Getters.sol#31-33) - CATERC20Getters.finality() (contracts/ERC20/Getters.sol#35-37) - CATERC20Getters.getDecimals() (contracts/ERC20/Getters.sol#39-41) - CATERC20Getters.maxSupply() (contracts/ERC20/Getters.sol#43-45) - CATERC20Getters.mintedSupply() (contracts/ERC20/Getters.sol#47-49) - CATERC20Getters.nativeAsset() (contracts/ERC20/Getters.sol#51-53) - CATERC20Getters.isInitialized() (contracts/ERC20/Getters.sol#55-57)	High
End of table for Getters.sol	

contracts/ERC20/Setters.sol

Slither did not identify any vulnerabilities in the contract.

contracts/ERC20/Governance.sol

Slither results for Governance.sol	
Finding	**Impact**
CATERC20Governance.registerChain(uint16,bytes32,CATERC20Structs.SignatureVerification).chainId (contracts/ERC20/Governance.sol#80) shadows: - CATERC20Getters.chainId() (contracts/ERC20/Getters.sol#23-25) (function)	Low

Finding	Impact
CATERC20Governance.updateFinality(uint8,CATERC20Structs .SignatureVerification).finality (contracts/ERC20/Governance.sol#100) shadows: - CATERC20Getters.finality() (contracts/ERC20/Getters.sol#35-37) (function)	Low
CATERC20Governance.registerChains(uint16[],bytes32[],CATERC20Structs .SignatureVerification).chainId (contracts/ERC20/Governance.sol#88) shadows: - CATERC20Getters.chainId() (contracts/ERC20/Getters.sol#23-25) (function)	Low
End of table for Governance.sol	

contracts/ERC20/State.sol

Slither did not identify any vulnerabilities in the contract.

contracts/ERC721/CATERC721.sol

Slither results for CATERC721.sol	
Finding	**Impact**
CATERC721.initialize(uint16,address,uint8,uint256,string).wormhole (contracts/ERC721/CATERC721.sol#40) shadows: - CATERC721Getters.wormhole() (contracts/ERC721/Getters.sol#19-21) (function)	Low
CATERC721Governance.registerChains(uint16[],bytes32[],CATERC721Structs .SignatureVerification).chainId (contracts/ERC721/Governance.sol#88) shadows: - CATERC721Getters.chainId() (contracts/ERC721/Getters.sol#23-25) (function)	Low
CATERC721.initialize(uint16,address,uint8,uint256,string).finality (contracts/ERC721/CATERC721.sol#41) shadows: - CATERC721Getters.finality() (contracts/ERC721/Getters.sol#35-37) (function)	Low
CATERC721.initialize(uint16,address,uint8,uint256,string).chainId (contracts/ERC721/CATERC721.sol#39) shadows: - CATERC721Getters.chainId() (contracts/ERC721/Getters.sol#23-25) (function)	Low

Finding	Impact
CATERC721Governance.updateFinality(uint8,CATERC721Structs .SignatureVerification).finality (contracts/ERC721/Governance.sol#100) shadows: - CATERC721Getters.finality() (contracts/ERC721/Getters.sol#35-37) (function)	Low
CATERC721.initialize(uint16,address,uint8,uint256,string).maxSupply (contracts/ERC721/CATERC721.sol#42) shadows: - CATERC721Getters.maxSupply() (contracts/ERC721/Getters.sol#43-45) (function)	Low
CATERC721Governance.registerChain(uint16,bytes32,CATERC721Structs .SignatureVerification).chainId (contracts/ERC721/Governance.sol#80) shadows: - CATERC721Getters.chainId() (contracts/ERC721/Getters.sol#23-25) (function)	Low
End of table for CATERC721.sol	

contracts/ERC721/CATERC721Proxy.sol

Slither results for CATERC721Proxy.sol	
Finding	**Impact**
CATERC721Governance.registerChains(uint16[],bytes32[], CATERC721Structs .SignatureVerification).chainId (contracts/ERC721/Governance.sol#88) shadows: - CATERC721Getters.chainId() (contracts/ERC721/Getters.sol#23-25) (function)	Low
CATERC721Governance.updateFinality(uint8,CATERC721Structs .SignatureVerification).finality (contracts/ERC721/Governance.sol#100) shadows: - CATERC721Getters.finality() (contracts/ERC721/Getters.sol#35-37) (function)	Low
CATERC721Proxy.initialize(uint16,address,address,uint8).finality (contracts/ERC721/CATERC721Proxy.sol#30) shadows: - CATERC721Getters.finality() (contracts/ERC721/Getters.sol#35-37) (function)	Low

AUTOMATED TESTING

Finding	Impact
CATERC721Proxy.initialize(uint16,address,address,uint8).chainId (contracts/ERC721/CATERC721Proxy.sol#27) shadows: - CATERC721Getters.chainId() (contracts/ERC721/Getters.sol#23-25) (function)	Low
CATERC721Governance.registerChain(uint16,bytes32,CATERC721Structs .SignatureVerification).chainId (contracts/ERC721/Governance.sol#80) shadows: - CATERC721Getters.chainId() (contracts/ERC721/Getters.sol#23-25) (function)	Low
CATERC721Proxy.initialize(uint16,address,address,uint8).wormhole (contracts/ERC721/CATERC721Proxy.sol#29) shadows: - CATERC721Getters.wormhole() (contracts/ERC721/Getters.sol#19-21) (function)	Low
Reentrancy in CATERC721Proxy.bridgeOut(uint256,uint16,bytes32,uint32) (contracts/ERC721/CATERC721Proxy.sol#57-99): External calls: - nativeAsset().safeTransferFrom(_msgSender(),address(this),tokenId) (contracts/ERC721/CATERC721Proxy.sol#69) - uriString = nativeAsset().tokenURI(tokenId) (contracts/ERC721/CATERC721Proxy.sol#73) - sequence = wormhole().publishMessage{value: msg.value}(nonce,encoded,finality()) (contracts/ERC721/CATERC721Proxy.sol#88) External calls sending eth: - sequence = wormhole().publishMessage{value: msg.value}(nonce,encoded,finality()) (contracts/ERC721/CATERC721Proxy.sol#88) Event emitted after the call(s): - bridgeOutEvent(payload.tokenID,payload.tokenChain,payload.toChain, addressToBytes(nativeAsset().ownerOf(tokenId)),recipient) (contracts/ERC721/CATERC721Proxy.sol#90-96)	Low

Finding	Impact
Reentrancy in CATERC721Proxy.bridgeIn(bytes) (contracts/ERC721/CATERC721Proxy.sol#101-132): External calls: - nativeAsset().safeTransferFrom(address(this),transferRecipient, transfer.tokenID) (contracts/ERC721/CATERC721Proxy.sol#122) Event emitted after the call(s): - bridgeInEvent(transfer.tokenID,transfer.tokenChain,transfer.toChain, transfer.toAddress) (contracts/ERC721/CATERC721Proxy.sol#124-129)	Low
End of table for CATERC721Proxy.sol	

contracts/ERC721/Structs.sol

Slither did not identify any vulnerabilities in the contract.

contracts/ERC721/Getters.sol

Slither results for Getters.sol	
Finding	**Impact**
CATERC721State._state (contracts/ERC721/State.sol#50) is never initialized. It is used in: - CATERC721Getters.isTransferCompleted(bytes32) (contracts/ERC721/Getters.sol#15-17) - CATERC721Getters.wormhole() (contracts/ERC721/Getters.sol#19-21) - CATERC721Getters.chainId() (contracts/ERC721/Getters.sol#23-25) - CATERC721Getters.evmChainId() (contracts/ERC721/Getters.sol#27-29) - CATERC721Getters.tokenContracts(uint16) (contracts/ERC721/Getters.sol#31-33) - CATERC721Getters.finality() (contracts/ERC721/Getters.sol#35-37) - CATERC721Getters.nativeAsset() (contracts/ERC721/Getters.sol#39-41) - CATERC721Getters.maxSupply() (contracts/ERC721/Getters.sol#43-45) - CATERC721Getters.mintedSupply() (contracts/ERC721/Getters.sol#47-49) - CATERC721Getters.isInitialized() (contracts/ERC721/Getters.sol#51-53) - CATERC721Getters.baseUri() (contracts/ERC721/Getters.sol#55-57)	High
End of table for Getters.sol	

AUTOMATED TESTING

contracts/ERC721/Setters.sol

Slither did not identify any vulnerabilities in the contract.

contracts/ERC721/Governance.sol

Slither results for Governance.sol	
Finding	**Impact**
CATERC721Governance.registerChains(uint16[],bytes32[], CATERC721Structs .SignatureVerification).chainId (contracts/ERC721/Governance.sol#88) shadows: - CATERC721Getters.chainId() (contracts/ERC721/Getters.sol#23-25) (function)	Low
CATERC721Governance.updateFinality(uint8,CATERC721Structs .SignatureVerification).finality (contracts/ERC721/Governance.sol#100) shadows: - CATERC721Getters.finality() (contracts/ERC721/Getters.sol#35-37) (function)	Low
CATERC721Governance.registerChain(uint16,bytes32,CATERC721Structs .SignatureVerification).chainId (contracts/ERC721/Governance.sol#80) shadows: - CATERC721Getters.chainId() (contracts/ERC721/Getters.sol#23-25) (function)	Low
End of table for Governance.sol	

contracts/ERC721/State.sol

Slither did not identify any vulnerabilities in the contract.

The findings obtained as a result of the Slither scan were reviewed. The high-risk vulnerabilities discovered in the CATERC20Proxy contract related to the transfer function were confirmed and included in the report. The other findings were determined as false positives.

5.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:

contracts/ERC20/CATERC20.sol

Report for contracts/ERC20/CATERC20.sol
https://dashboard.mythx.io/#/console/analyses/59eb6325-4879-42d2-ada7-447837caedb6

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
15	(SWC-123) Requirement Violation	Low	Requirement violation.
60	(SWC-123) Requirement Violation	Low	Requirement violation.

contracts/ERC20/CATERC20Proxy.sol

Report for contracts/ERC20/CATERC20Proxy.sol
https://dashboard.mythx.io/#/console/analyses/c7a93428-0229-4025-b2bc-284b751a71d1

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
13	(SWC-123) Requirement Violation	Low	Requirement violation.
54	(SWC-123) Requirement Violation	Low	Requirement violation.

contracts/ERC20/Structs.sol

Report for contracts/ERC20/Structs.sol
https://dashboard.mythx.io/#/console/analyses/c0acc1d6-aad5-4aa7-b215-3b5432d6dd14

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.

contracts/ERC20/Getters.sol

Report for contracts/ERC20/Getters.sol
https://dashboard.mythx.io/#/console/analyses/603d37a3-57bf-4895-9adb-72ecf939c8d1

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.

contracts/ERC20/Setters.sol

Report for Setters.sol
https://dashboard.mythx.io/#/console/analyses/5251d4b7-cadf-4ee6-821c-f3effc333c41
https://dashboard.mythx.io/#/console/analyses/d2743f6b-720d-4577-a997-deb0cc78d6de

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.

contracts/ERC20/Governance.sol

Report for contracts/ERC20/Governance.sol
https://dashboard.mythx.io/#/console/analyses/6be95fe4-9bd9-4de2-aba7-114faac2e31f

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.

contracts/ERC20/State.sol

Report for State.sol
https://dashboard.mythx.io/#/console/analyses/da2941e2-ffc2-4dfa-bdbb-dd8c146a9e99
https://dashboard.mythx.io/#/console/analyses/6e22ee21-a24f-47d7-b298-ce13e8357a91

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.
50	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

contracts/ERC721/CATERC721.sol

Report for contracts/ERC721/CATERC721.sol
https://dashboard.mythx.io/#/console/analyses/bb85bc22-ec6b-41ce-98f6-584657d19c15

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
21	(SWC-123) Requirement Violation	Low	Requirement violation.
139	(SWC-123) Requirement Violation	Low	Requirement violation.

contracts/ERC721/CATERC721Proxy.sol

Report for contracts/ERC721/CATERC721Proxy.sol
https://dashboard.mythx.io/#/console/analyses/cacb5090-9bde-40f1-a9e3-b1b8537d61db

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
18	(SWC-123) Requirement Violation	Low	Requirement violation.
65	(SWC-123) Requirement Violation	Low	Requirement violation.

contracts/ERC721/Structs.sol

MythX did not identify any vulnerabilities in the contract.

contracts/ERC721/Getters.sol

Report for contracts/ERC721/Getters.sol
https://dashboard.mythx.io/#/console/analyses/c6125bee-620e-453e-8c77-716281fe9268

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.

contracts/ERC721/Setters.sol

Report for Setters.sol
https://dashboard.mythx.io/#/console/analyses/5251d4b7-cadf-4ee6-821c-f3effc333c41
https://dashboard.mythx.io/#/console/analyses/d2743f6b-720d-4577-a997-deb0cc78d6de

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.

contracts/ERC721/Governance.sol

Report for contracts/ERC721/Governance.sol
https://dashboard.mythx.io/#/console/analyses/3998cdde-074e-4c3e-a1f4-0c97f60672dd

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.

contracts/ERC721/State.sol

Report for State.sol
https://dashboard.mythx.io/#/console/analyses/da2941e2-ffc2-4dfa-bdbb-dd8c146a9e99
https://dashboard.mythx.io/#/console/analyses/6e22ee21-a24f-47d7-b298-ce13e8357a91

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.
50	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

The findings obtained as a result of the MythX scan were examined, and they were not included in the report because they were determined false positives.

THANK YOU FOR CHOOSING

HALBORN