AUDITONE

Audit Report

humans

18.10.2023

Table of Contents

CONTENTS

INTRODUCTION

Smart Contract Security Analysis Report

Note: This report may contain sensitive information on potential vulnerabilities and exploitation methods. This must be referred internally and should be only made available to the public after issues are resolved (to be confirmed prior by the client and AuditOne).

INTRODUCTION

Defsec, Eugeniocirc and Zuhaibmohd, who are auditors at AuditOne, successfully audited the smart contracts (as indicated below) of Humans.ai. The audit has been performed using manual analysis. This report presents all the findings regarding the audit performed on the customer's smart contracts. The report outlines how potential security risks are evaluated. Recommendations on quality assurance and security standards are provided in the report.

01-PROJECT DESCRIPTION

Description

02-Project and Audit Information

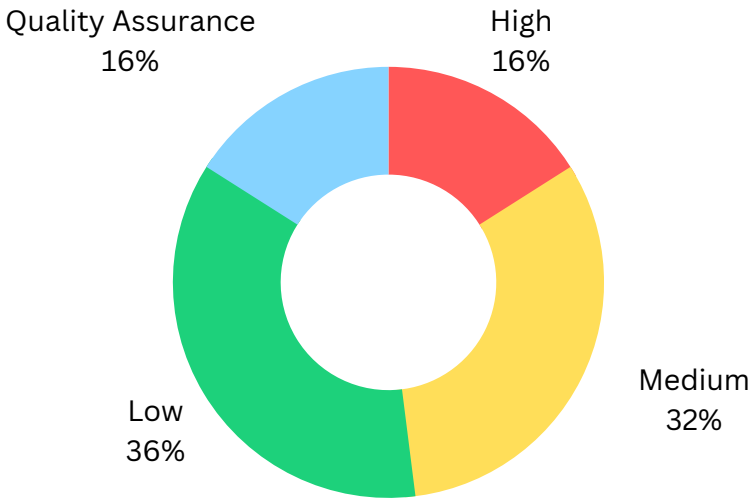
Term	Description
Auditor	Defsec, Eugenioclrc and Zuhaibmohd
Reviewed by	Gracious
Type	Bridge
Language	Solidity
Ecosystem	NA
Methods	Manual Review
Repository	https://github.com/ToDy95/bridgeAuditone
Commit hash (at audit start)	f8016c67749fbdf7345c3987f79fcd95616bcf2f
Commit hash (after resolution)	eb57c4c2dcd4d39c9f847b90432312ef94477f8d
Documentation	[Added once the whitepaper is published by the project]
Unit Testing	No
Website	https://humans.ai/
Submission date	22-09-2023
Finishing date	18-10-2023

03-Contracts in Scope

- contracts\SynapseBridge.sol
- api\routes\getHistory.route.js
- api\routes\signRefund.route.js
- api\routes\signWithdraw.route.js
- api\lib\depositListener.js
- api\lib\renounceListener.js
- api\lib\withdrawListener.js
- web\lib\utils.js
- web\lib\utils_sol.js
- web\pages\context\Context.js
- web\pages\2_ConnectDestination.js (connect wallet)
- web\pages\3_ConnectSource.js (connect wallet)
- web\pages\5_ApproveERC20.js (approve token)
- web\pages\6_Deposit.js (deposit function call)
- web\pages\7_Retry.js (retry user flow of the withdraw function call)
- web\pages\8_Refund.js (refund user flow: renounce function call + withdraw function call)
- web\pages\9_Withdraw.js (withdraw function call)

04-Executive summary

Humans AI smart contracts were audited between 22-09-2023 and 18-10-2023 by Defsec, Eugenioclrc and Zuhaibmohd. Manual analysis was carried out on the code base provided by the client. The following findings were reported to the client. For more details, refer to the findings section of the report.



Issue Category	Issues Found	Resolved	Acknowledged
High	4	4	0
Medium	8	3	5
Low	9	2	7
Quality Assurance	4	4	0

05-Severity Definitions

Risk factor matrix	Low	Medium	High
Occasional	L	M	H
Probable	L	M	H
Frequent	M	H	H

High: Funds or control of the contracts might be compromised directly. Data could be manipulated. We recommend fixing high issues with priority as they can lead to severe losses.

Medium: The impact of medium issues is less critical than high but still probable with considerable damage. The protocol or its availability could be impacted or leak value with a hypothetical attack path with stated assumptions.

Low: Low issues impose a small risk on the project. Although the impact is not estimated to be significant, we recommend fixing them on a long-term horizon. Assets are not at risk: state handling, function incorrect as to spec, issues with comments.

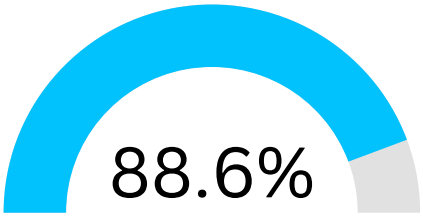
Quality Assurance: Informational and Optimization - Depending on the chain, performance issues can lead to slower execution or higher gas fees. For example, code style, clarity, syntax, versioning, off-chain monitoring (events, etc.)

Occasional: This category might represent risks with a moderate chance of occurring. These risks are not common but have happened in similar situations.

Probable: This category represents risks that are likely to happen. There's a high probability based on past experiences, current conditions, or future projections.

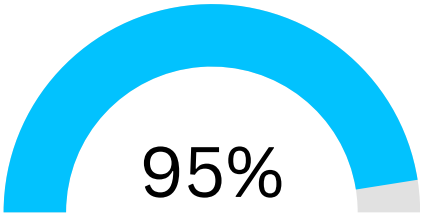
Frequent: This category represents risks that occur regularly. In a security audit, this might refer to common vulnerabilities or threats consistently observed in similar systems or environments.

06-Audit Overview



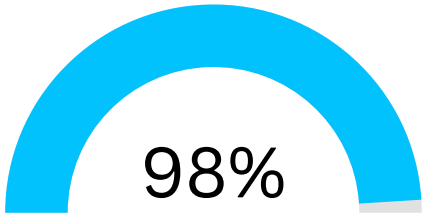
Security score

Security score is a numerical value generated based on the vulnerabilities in smart contracts. The score indicates the contract's security level and a higher score implies a lower risk of vulnerability.



Code quality

Code quality refers to adherence to standard practices, guidelines, and conventions when writing computer code. A high-quality codebase is easy to understand, maintain, and extend, while a low-quality codebase is hard to read and modify.



Documentation quality

Documentation quality refers to the accuracy, completeness, and clarity of the documentation accompanying the code. High-quality documentation helps auditors to understand business logic in code well, while low-quality documentation can lead to confusion and mistakes.

07-Findings

Finding: #1

Issue: Underflow in `emergencyWithdrawNative` method can brick emergency withdraw

Severity: High

Where: `emergencyWithdrawNative`: SynapseBridge.sol#L530

Impact: If the `nativeTokenBalance` is less than the actual balance of the contract, calling the `emergencyWithdrawNative` method will result in a revert due to underflow, bricking this emergency functionality and potentially locking funds.

Description: In the `emergencyWithdrawNative` method, the contract attempts to withdraw all native tokens from the contract and update the `nativeTokenBalance` state variable accordingly. However, if the `nativeTokenBalance` is less than the actual balance **(address(this).balance)**, subtracting the actual balance from `nativeTokenBalance` will underflow and cause the transaction to revert.
This discrepancy between `nativeTokenBalance` and the actual balance can occur due to direct transfers to the contract address, as the contract has a `receive` function that updates `nativeTokenBalance` accordingly, but it does not prevent direct transfers.

Here is the problematic line of code:

```
nativeTokenBalance -= balance;
```

POC:

```
contract TestPoc {
   function test_BridgeHack() public {
        vm.chainId(1); // mainnet
        new Bomb{value: 1}(address(bridgeHumansEthereum));

        vm.prank(ADMIN);
        bridgeHumansEthereum.emergencyWithdrawNative():

    }
}

contract Bomb {
    constructor(address t) payable {
        selfdestruct(payable(t));
    }
}
```

Recommendations:

- **Safest Approach:** Remove the usage of **nativeTokenBalance** from the contract, and wherever needed, use **address(this).balance** directly, as it gives the actual balance of the contract.
- **Alternative Approach:** If the usage of **nativeTokenBalance** is necessary, then in the **emergencyWithdrawNative** method, instead of subtracting the balance, set **nativeTokenBalance** to 0 after transferring all the funds, like so:

```
nativeTokenBalance = 0;
```

Status: Resolved.

Finding: #2

Issue: Cross reentrancy issue due to missing **nonReentrant** modifier on **renounceClaim.**

Severity: High

Where: SynapseBridge.sol#L421-L433
SynapseBridge.sol#L384-L385

Impact: This vulnerability could allow a malicious actor to perform a cross reentrancy attack by interacting with **withdrawNative** and subsequently calling **renounceClaim**, enabling the user to claim the tokens and simultaneously renounce the claim, potentially leading to financial loss or unexpected behavior of the contract.

Description: In the **withdrawNative**, ether is sent to the user in the middle of execution, allowing user control before updating the state. Although the **reentrancyGuard** modifier is in **withdrawNative**, it is notably missing in the **renounceClaim** function, despite comments indicating its necessity. This absence of the **nonReentrant** modifier in **renounceClaim** could potentially allow cross reentrancy attacks.

```
// @notice renounceClaim() allows users to renounce their claim on a deposit
// @dev This function is non-reentrant and can only be called when the bridge is not paused.
function renounceClaim(
    address destinationNetworkToken,
    address sender,
    uint256 amount,
    uint256 nonce,
    bytes memory signature
) public whenNotPaused {
```

Recommendations: Apply the **nonReentrant** modifier to the **renounceClaim** function to prevent reentrancy attacks. Adhere to the checks-effects-interactions pattern to avoid unexpected behaviours, even when using reentrancy guards.

Status: Resolved.

Finding: #3

Issue: Potential brick of **depositNativeToken**

Severity: High

Where:
https://github.com/ToDy95/bridgeAuditone/blob/main/contracts/contracts/SynapseBridge.sol#L230-L232

Impact: If someone force sends ether to the contract, the **depositNativeToken** method could become unusable. This would cause the function to consistently revert with **InvalidTransfer(),** potentially locking up funds.

Description: The **depositNativeToken** function checks if the contract's balance has increased by the deposited amount. However, if someone forcibly sends ether to the contract without using the deposit function, this check will fail, causing the function to revert.
The problematic code is:

```
        // Check if token balance has increased by amount
        if (address(this).balance - msg.value != nativeTokenBalance)
            revert InvalidTransfer();
```

POC:

```
contract TestPoc {
    // .......
    function test_BridgeHack() public {
        vm.chainId(1089);
        new Bomb{value: 1}(address(bridgeHumansEthereum));

        bridgeHumansEthereum.depositNativeToken{value: 100}(makeAddr("receiver"));
    }
    // .......
}

contract Bomb {
    constructor(address t) payable {
        selfdestruct(payable(t));
    }
}
```

Recommendations: Ether transfers don't involve fees, so the check is unnecessary. To resolve this, remove the lines SynapseBridge.sol#L230-L232

Status: Resolved.

Finding: #4

Issue: Unsecured Storage of Sensitive Credentials in **.env** File

Severity: High

Where: .env.example#L4

Impact: With direct access to the database through credentials, data can be manipulated, leading to system malfunction or data integrity issues.

Description: Upon inspection, sensitive credentials and configuration details, including database login information is being stored in plain text within the **.env** file. This approach to storing secrets could expose them to unintended users or malicious actors, compromising system security.

```
STORE_DATABASE=synapse
STORE_USERNAME=synapse
STORE_PASSWORD=synapse
STORE_ENCRYPTION_KEY=synapse
```

Recommendations: Implement tools such as AWS Secrets Manager, HashiCorp Vault, or Google Cloud Secret Manager. These tools provide robust mechanisms to store, retrieve, and manage sensitive data securely.

Status: Resolved.

Finding: #5

Issue: Missing Container Security Context ReadOnlyRootFilesystem in Kubernetes Configuration File

Severity: Medium

Where: deploy-dev.yml

Impact: One setting that controls whether a container is able to write into its filesystem is readOnlyRootFilesystem,a feature you definitely want enabled, if possible, to provide additional defense-in-depth for your containerized workloads. If an attacker or bad actor manages to breach a cluster, readOnlyRootFilesystem will prevent them from tampering with the application or writing executables to a disk.

Description: An immutable root filesystem prevents applications from writing to their local disk. This is desirable in the event of an intrusion as the attacker will not be able to tamper with the filesystem or write foreign executables to disk.

Recommendations: Makes sure that all pods have a security context with read only filesystem set.
Reference : https://kubesec.io/basics/containers-securitycontext-readonlyrootfilesystem-true/

Status: Acknowledged

Finding: #6

Issue: Withdrawals Can Be Blocked if All Signer Roles are Revoked

Severity: Medium

Where: SynapseBridge.sol#L573

Impact: Withdrawal blockage

Description: The current implementation of the token bridge allows the admin to revoke the **SIGNER_ROLE** from addresses via the **removeSigner** function. While it's essential to have the ability to revoke signer roles, there is a potential issue in a situation where all signer roles are revoked, and there are no addresses left with the **SIGNER_ROLE** assigned. This situation would render the bridge unable to process withdrawals, as the signatures for withdrawal operations cannot be validated, thereby essentially blocking withdrawals on the bridge.

```
// @notice removeSigner() allows the admin to revoke the signer role from an address
// @dev This function can only be called by the admin
// @param _signer The address to revoke the signer role from
function removeSigner(
    address _signer
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (_signer == address(0)) revert AddressZeroCheck();
    _revokeRole(SIGNER_ROLE, _signer);
}
```

Recommendations: Implement checks within the removeSigner function to ensure that there is at least one signer left after a **SIGNER_ROLE** revocation, preventing a situation where there are zero addresses with **SIGNER_ROLE.**

Status: Acknowledged

Finding: #7

Issue: Assignment of Critical Roles to Externally Owned Accounts (EOA) through **initialize** function.

Severity: Medium

Where: SynapseBridge.sol#L45

Impact: If the EOA associated with the roles gets compromised, the attacker would gain full control over the contract, including pausing operations, upgrading the contract, and more.

Description: The contract's initialize function is granting the DEFAULT_ADMIN_ROLE, PAUSER_ROLE, and UPGRADER_ROLE to the sender **(_msgSender())** of the transaction, which in most deployment scenarios will be an Externally Owned Account (EOA). EOAs are susceptible to loss or compromise, and if they are given high-level permissions within a contract, it represents a single point of failure.

```
function initialize(
    address _evmERC20Token,
    uint256 _nativeChainId,
    uint256 _evmChainId
) public initializer {
    if (_evmERC20Token == address(0)) revert AddressZeroCheck();
    if (_nativeChainId == 0) revert ChainIdZeroCheck();
    if (_evmChainId == 0) revert ChainIdZeroCheck();

    __Pausable_init();
    __AccessControl_init();
    __UUPSUpgradeable_init();
    __ReentrancyGuard_init();

    _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
    _grantRole(PAUSER_ROLE, _msgSender());
    _grantRole(UPGRADER_ROLE, _msgSender());

    // Set the native chain ID (where the bridge transfers native tokens)
    nativeChainId = _nativeChainId;

    // Set the evm chain ID (where the bridge transfers ERC20 tokens)
    evmChainId = _evmChainId;

    // Address of the ERC20 token contract that the bridge will transfer on the evm chain
    evmERC20Token = IERC20Upgradeable(_evmERC20Token);
}
```

Recommendations: Instead of granting roles to an EOA, consider assigning them to a multi-signature wallet.

Status: Acknowledged.

Finding: #8

Issue: Unrestricted Emergency Withdrawal Functions

Severity: Medium

Where: SynapseBridge.sol#L521-L542

Impact: The admin can intentionally or unintentionally withdraw funds at any time, possibly leading to loss of users' funds or disruption in the bridge's operations.

Description: The contract **SynapseBridge** includes emergency withdrawal functions **emergencyWithdrawNative** and **emergencyWithdrawERC20** that allow the admin to withdraw either native Ether or any ERC20 token from the bridge, respectively. However, these functions are not restricted to be called only when the contract is paused, giving the admin unfettered access to the funds, even during normal operations.

```
function emergencyWithdrawNative() external onlyRole(DEFAULT_ADMIN_ROLE){}
function emergencyWithdrawERC20(IERC20Upgradeable token) external onlyRole(DEFAULT_ADMIN_ROLE) {}
```

Recommendations: Modify the emergency withdrawal functions to include the **whenPaused** modifier. This ensures that these functions can only be called when the contract is in a paused state, adding an extra layer of protection against unauthorized or unexpected withdrawals.

Status: Acknowledged.

Finding: #9

Issue: Direct Ether Sending Without Validation or Restriction

Severity: Medium

Where:
https://github.com/ToDy95/bridgeAuditone/blob/main/contracts/contracts/SynapseBridge.sol#L599C16-L599C16

Impact: Without specific validation or conditions, there is no way to ensure that the Ether received by the contract corresponds to any valid cross-chain or bridging operation. This could lead to accounting discrepancies or inconsistencies.
With the self.destruct, that can cause DOS.

Description: The contract **SynapseBridge** has a public payable **receive()** function that allows it to accept native Ether. This function increases the nativeTokenBalance of the contract by the amount of Ether sent, and emits a **PaymentReceived** event. However, there is no restriction or validation on who can send Ether to the contract, or under what conditions.

```
// @notice receive() allows the contract to receive native tokens
receive() external payable virtual {
    emit PaymentReceived(_msgSender(), msg.value);

    // Increase native token balance
    nativeTokenBalance += msg.value;
}
```

Recommendations: Implement a function that allows admins or specific roles to refund unintended Ether deposits.

Status: Resolved.

Finding: #10

Issue: Inadequate Security Headers Configuration in Humans AI Web Application

Severity: Medium

Where: next.config.js

Impact:

- **Clickjacking**: Without the **X-Frame-Options** header set to "DENY", attackers can embed your web page in an iframe and trick users into clicking on something different from what they perceive, potentially leading to unintended actions.
- **Cross-Site Scripting (XSS):** Without a well-defined **Content-Security-Policy**, malicious scripts can be executed in the context of the visitor's browser.
- **Content Sniffing:** Absence of the **X-Content-Type-Options: nosniff** header makes browsers guess the content type, which can be exploited by attackers to execute malicious content.
- **Privacy Violations**: Not having a **Permissions-Policy** can allow third-party websites to access device functionalities without the user's knowledge.
- **Referrer Leaks**: Without the **Referrer-Policy** set to **origin-when-cross-origin**, full URLs can be leaked to external sources when following links, potentially exposing sensitive information.

Description: Upon review of the configuration for the Humans AI web application, it was observed that the application does not have security-related HTTP response headers configured, compared to a secure reference configuration. The absence of these headers increases the vulnerability of the application to several types of attacks, including clickjacking, content sniffing, and cross-site scripting.

Recommendations:

- **Enable reactStrictMode**: It helps in catching potential problems in an application during the development phase.
- **Configure Security Headers:** Adopt the secure headers configuration provided in the reference to enhance the security posture of the application

Status: Acknowledged.

Finding: #11

Issue: Missing check for source **chainId** may lead to users losing funds.

Severity: Medium

Where:
https://github.com/ToDy95/bridgeAuditone/blob/f8016c67749fbdf7345c3987f79fcd95616bcf2f/contracts/contracts/SynapseBridge.sol#L173-L176
https://github.com/ToDy95/bridgeAuditone/blob/f8016c67749fbdf7345c3987f79fcd95616bcf2f/contracts/contracts/SynapseBridge.sol#L222-L224
https://github.com/ToDy95/bridgeAuditone/blob/f8016c67749fbdf7345c3987f79fcd95616bcf2f/contracts/contracts/SynapseBridge.sol#L270-L275
https://github.com/ToDy95/bridgeAuditone/blob/f8016c67749fbdf7345c3987f79fcd95616bcf2f/contracts/contracts/SynapseBridge.sol#L346-L351
https://github.com/ToDy95/bridgeAuditone/blob/f8016c67749fbdf7345c3987f79fcd95616bcf2f/contracts/contracts/SynapseBridge.sol#L599

Impact: The user who deposits funds via the wrong function, might end up losing the funds forever.

Description: The identified vulnerability arises from a lack of **chainId** validation in the contract code. Specifically, the contract allows functions meant for Ethereum's mainnet **(chainId 1)** to be executed on Humans.ai, which is using **chainId 1089**. This may not be possible in case the user in comfortable using the UI, but mostly the case wherein they are interacting with the smart contract.

Example Scenario:

1. A user decides to interact with the smart contract directly and calls **depositNativeToke**n on Ethereum's mainnet.
2. **depositNativeToken** is supposed to be called on Humans.ai chain to deposit $HEART tokens and not native Ether.
3. As a result, the user now cannot claim, the deposited ether on Humans.ai since there is zero liquidity of Ether on Humans.ai chain.

Even though the percentage of people interacting with the smart contract is very low, the impact of the issue is high.

Recommendation: Have a check for all the functions to verify the chainId is the correct one.

Sample Fix for Ethereum's mainnet.

```
require(getChainID() == 1, "The Function can be called on Ethereum's mainnet.")
```

Sample Fix for Humans.ai

```
require(getChainID() == 1089, "The Function can be called on Humans.ai.")
```

Status: Resolved.

Finding: #12

Issue: SynapseBridge.sol lacks reserved space for future upgrades.

Severity: Medium

Where:
https://github.com/ToDy95/bridgeAuditone/blob/main/contracts/contracts/SynapseBridge.sol

Impact: Without storage gap, the variable in child contract might be overwritten by the upgraded base contract if new variables are added to the base contract.

Description: SynapseBridge.sol which is upgradeable, lacks storage gaps. It is considered a best practice in upgradeable contracts to include a state variable named **__gap**. This **__gap** state variable will be used as a reserved space for future upgrades. It allows adding new state variables freely in the future without compromising the storage compatibility with existing deployments.
The size of the **__gap** array is usually calculated so that the amount of storage used by a contract always adds up to the same number (usually 50 storage slots).
Reference:
OpenZeppelin's storage gap -
https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gap

Recommendations: It is recommended to add a state variable named **__gap** as a reserved space for future upgrades in the contract.

Expected Fix :
uint256[50] private __gap;

Status: Resolved.

Finding: #13

Issue: Utilize CapDropAny & CapDropAll on the configuration file

Severity: Low

Where: deploy-dev.yml

Impact: Reducing kernel capabilities available to a container limits its attack surface and safeguards against certain types of escalations and attacks. The current configuration doesn't utilize CapDropAny & CapDropAll settings which, if applied, can enhance the security posture of the deployed containers.

Description: Upon review of the Kubernetes configuration file, deploy-dev.yml, it has been observed that the securityContext settings are not currently utilized to minimize kernel capabilities for the running containers. Specifically, capabilities dropping (CapDropAny & CapDropAll) is not configured.
Kernel capabilities provide a finer-grained control over the permissions granted to containers. When all capabilities are dropped (CapDropAll) and only the necessary ones are added back explicitly, it restricts the actions available to the container processes, thus limiting their attack surface and potential impact in case of a security incident.

Recommendations: Consider setting CapDropAny & CapDropAll on the configuration file.

Status: Acknowledged.

Finding: #14

Issue: Security Risk due to Lack of **.env.example** Inclusion in **.gitignore**

Severity: Low

Where: .gitignore

Impact: While .env.example files typically should not contain sensitive data and should serve as a template, caution should be exercised to avoid revealing too much about underlying data structures, expected data formats, or other information that might be useful to malicious actors.

Description: The .env.example file, which potentially contains sensitive or critical data templates, is not specified in the .gitignore file. This oversight poses a risk of unintentional exposure or misinterpretation of environment setup details.

Recommendation: Add .env.example to .gitignore and validate the repository for any unintentional exposure in past commits.

Status: Resolved.

Finding: #15

Issue: Lack of **RunAsNonRoot** definition on the configuration file

Severity: Low

Where: deploy-dev.yml

Impact: Containers must be required to run as non-root users.

Description: Even though a container uses namespaces and cgroups to limit its process(es), all it takes is one misconfiguration in its deployment settings to grant those processes access to resources on the host. If that process runs as root, it has the same access as the host root account to those resources. Additionally, if other pod or container settings are used to reduce constraints (i.e. procMount or capabilities), having a root UID compounds the risks of any exploitation of them.

Recommendations: The Kubernetes Pod SecurityContext provides two options **runAsNonRoot** and **runAsUser** to enforce non root users. You can use both options separate from each other because they test for different configurations.

Status: Acknowledged.

Finding: #16

Issue: Container is missing a readinessProbe

Severity: Low

Where: deploy-dev.yml

Impact: The project owner is able to mint an infinite amount of COINC to their wallet, which makes it impossible to ensure that the value of COINC is equal to 1 USDC.

Description: A readinessProbe should be used to indicate when the service is ready to receive traffic. Without it, the Pod is risking to receive traffic before it has booted. It's also used during rollouts, and can prevent downtime if a new version of the application is failing.

Recommendations: Setup a readinessProbe that responds with a healthy status when: The application is fully booted, and ready to serve traffic.

Status: Acknowledged.

Finding: #17

Issue: Missing Container Ephemeral Storage Request and Limit

Severity: Low

Where: deploy-dev.yml

Impact: Lack of protection for DDOS attacks.

Description: Ephemeral Storage limit is not set. Resource limits are recommended to avoid resource DDOS.

Recommendations: Makes sure all pods have ephemeral-storage requests and limits set.

Status: Acknowledged.

Finding: #18

Issue: Potential Blocking of Withdrawals Due to Non-Assignment of Signer Role in **Initializer**.

Severity: Low

Where: SynapseBridge.sol#L31

Impact: This can prevent users from withdrawing their assets If the signers are not assigned.

Description: In the assessment of the contract logic, it has been identified that the **SIGNER_ROLE** is not assigned during the contract initialization in the initialize() function. Given that certain operations within the contract (especially those pertaining to withdrawal and cross-chain transactions) might require signature validations, the non-assignment of a signer at initialization could expose the contract to an operational risk where withdrawals and other signer-dependent functions can be unintentionally blocked or disrupted.

```
function initialize(
    address _evmERC20Token,
    uint256 _nativeChainId,
    uint256 _evmChainId
) public initializer {
    if (_evmERC20Token == address(0)) revert AddressZeroCheck();
    if (_nativeChainId == 0) revert ChainIdZeroCheck();
    if (_evmChainId == 0) revert ChainIdZeroCheck();

    __Pausable_init();
    __AccessControl_init();
    __UUPSUpgradeable_init();
    __ReentrancyGuard_init();

    _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
    _grantRole(PAUSER_ROLE, _msgSender());
    _grantRole(UPGRADER_ROLE, _msgSender());

    // Set the native chain ID (where the bridge transfers native tokens)
    nativeChainId = _nativeChainId;

    // Set the evm chain ID (where the bridge transfers ERC20 tokens)
    evmChainId = _evmChainId;

    // Address of the ERC20 token contract that the bridge will transfer on the evm chain
    evmERC20Token = IERC20Upgradeable(_evmERC20Token);
}
```

Recommendation: Consider setting signer in the **initialize** function or do not allow deposits until signer is set.

Status: Acknowledged.

Finding: #19

Issue: Prefer **abi.encode** over **abi.encodePacked**

Severity: Low

Where: SynapseBridge.sol#L284

Impact: Possible hash collusion

Description: Use of **abi.encodePacked** in **SynapseBridge** contract is safe, but unnecessary and not recommended. **abi.encodePacked** can result in hash collisions when used with two dynamic arguments (string/bytes). **SynapseBridge** does not use any dynamic types, but for maximum safety against future mistakes, using **abi.encode** is recommended.

```
bytes32 messageHash = keccak256(
    abi.encodePacked(
        address(0),
        address(evmERC20Token),
        sender,
        _msgSender(),
        amount,
        nativeChainId,
        evmChainId,
        nonce
    )
);
```

Recommendations: Consider using **abi.encode** instead.

Status: Resolved

Finding: #20

Issue: Vulnerable Dependencies Detected in Project

Severity: Low

Where: package.json

Impact: Vulnerable dependencies can cause DOS attack.

Description: ammo Dependency Vulnerability

Severity: High

- **Issue**: Denial of Service in ammo
- **Details**: GHSA-mg85-8mv5-ffjr
- **Fix:** A fix is available using npm audit fix --force. This will install hapi-router@5.1.0, which is a breaking change.
- **Dependency Path:** node_modules/hapi/node_modules/ammo -> hapi -> hapi-router (<=4.0.1)

Severity: Moderate

- **Versions Affected**: <=5.7.1 || 6.0.0 - 6.3.0 || 7.0.0 - 7.5.1
- **Issue**: Regular Expression Denial of Service in semver
- **Details**: GHSA-c2qf-rxjj-qqgw
- **Fix**: A remedy is available through npm audit fix --force. This will install nodemon@3.0.1, which is a breaking change.
- **Dependency Path:** node_modules/make-dir/node_modules/semver -> nodemon -> simple-update-notifier (1.0.7 - 1.1.0)

Severity: High

- **Issues:**
- **Prototype Pollution in subtext:** GHSA-g64q-3vg8-8f93
- **Denial of Service in subtext:** GHSA-5854-jvxx-2cg9 & GHSA-2mvq-xp48-4c77
- **Fix:** A solution is available using npm audit fix --force. This will install hapi-router@5.1.0, which introduces breaking changes.

Recommendations: Consider upgrading vulnerable dependencies.

Status: Acknowledged.

Finding: #21

Issue: Docker image running as root

Severity: Low

Where: Dockerfile#L1
Dockerfile#L1

Impact: This allows for unrestricted container management, meaning a user could install system packages, edit configuration files, bind privileged ports, etc.

Description: Docker containers generally run with root privileges by default. This allows for unrestricted container management, meaning a user could install system packages, edit configuration files, bind privileged ports, etc. During static analysis, it was observed that the docker image is maintained through the root user.

```
FROM node:lts-alpine
WORKDIR /usr/src/backend
COPY package*.json ./
RUN apk add openssl1.1-compat ca-certificates
RUN npm install
COPY . .
EXPOSE 8000
CMD [ "npm","run","start" ]
```

Recommendations: It is recommended to build the **Dockerfile** and run the container as a non-root user.
USER 1001: this is a non-root user UID, and here it is assigned to the image to run the current container as an unprivileged user. By doing so, the added security and other restrictions mentioned above are applied to the container.

Status: Acknowledged.

Finding: #22

Issue: QA Report

Severity: QA

Where: Whole codebase

Impact:

Description:

- NC-01 Avoid open pragma

Setup a pragma version, avoid open pragma, this will help to avoid unexpected behavior due to compiler updates.

- NC-02 Use named imports

This will improve readability and will help to understand where are you importing from.

Example, instead of :
import "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol"

Use:
import {ReentrancyGuardUpgradeable} from "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol"

- NC-03 Use named mapping

This will improve readability.

Example, instead of:
mapping(uint256 => Deposit) public deposits;

Use:
mapping(uint256 id => Deposit) public deposits;

- NC-04 Use oz Address lib for transfer ether

On SynapseBridge.sol#L384-L385 and SynapseBridge.sol#L526-L527 instead of doing the raw call to send ether you can use **Address.sendValue** from oz lib.

- NC-05 There is no need of using getChainID() method

Since solidity 0.8 you can just use **block.chainid,** see https://docs.soliditylang.org/en/v0.8.0/units-and-global-variables.html.

- NC-06 stick to the check-effect-iterarions pattern

Even if you use a reentrancy modifier or is a owner function, you should stick to the check-effect-iterarions pattern, this will help to avoid unexpected behaviour.

- SynapseBridge.sol#L384-L385
- SynapseBridge.sol#L526-L527
- SynapseBridge.sol#L186
- SynapseBridge.sol#L308

Status: Resolved.

Finding: #23

Issue: Project has NPM Dependency which uses a vulnerable version : **@openzeppelin**

Severity: Low

Where:
https://github.com/ToDy95/bridgeAuditone/blob/f8016c67749fbdf7345c3987f79fcd95616bcf2f/contracts/package-lock.json#L18-L19

Impact: May negatively impact the contract.

Description: The current config file seems to be using a outdated version of **@openzeppelin contracts i.e., 4.8.2** which is vulnerable to some vulnerabilities which may negatively impact the contract as shown below.
Openzeppelin Official Security Advisories - https://github.com/OpenZeppelin/openzeppelin-contracts/security

Id	Title	Module	Severity	Last Updated	Affected Versions	Patched Versions
1091742	OpenZeppelin Contracts TransparentUpgradeableProxy clashing selector calls may not be delegated	@openzeppelin/contracts-upgradeable	moderate	4/18/2023	>=3.2.0 <4.8.3	>=4.8.3
1091743	OpenZeppelin Contracts TransparentUpgradeableProxy clashing selector calls may not be delegated	@openzeppelin/contracts	moderate	4/18/2023	>=3.2.0 <4.8.3	>=4.8.3
1091838	GovernorCompatibilityBravo may trim proposal calldata	@openzeppelin/contracts-upgradeable	high	4/27/2023	>=4.3.0 <4.8.3	>=4.8.3
1091839	GovernorCompatibilityBravo may trim proposal calldata	@openzeppelin/contracts	high	4/27/2023	>=4.3.0 <4.8.3	>=4.8.3
1092245	OpenZeppelin Contracts's governor proposal creation may be blocked by frontrunning	@openzeppelin/contracts-upgradeable	moderate	6/8/2023	>=4.3.0 <4.9.1	>=4.9.1
1092246	OpenZeppelin Contracts's governor proposal creation may be blocked by frontrunning	@openzeppelin/contracts	moderate	6/8/2023	>=4.3.0 <4.9.1	>=4.9.1
1092289	OpenZeppelin Contracts using MerkleProof multiproofs may allow proving arbitrary leaves for specific trees	@openzeppelin/contracts-upgradeable	moderate	6/19/2023	>=4.7.0 <4.9.2	>=4.9.2
1092290	OpenZeppelin Contracts using MerkleProof multiproofs may allow proving arbitrary leaves for specific trees	@openzeppelin/contracts	moderate	6/19/2023	>=4.7.0 <4.9.2	>=4.9.2
1092941	OpenZeppelin Contracts vulnerable to Improper Escaping of Output	@openzeppelin/contracts-upgradeable	moderate	8/11/2023	>=4.0.0 <4.9.3	>=4.9.3
1092942	OpenZeppelin Contracts vulnerable to Improper Escaping of Output	@openzeppelin/contracts	moderate	8/11/2023	>=4.0.0 <4.9.3	>=4.9.3

Recommendations: Upgrade OZ to the latest stable version - https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v4.9.3

Status: Resolved.

Finding: #24

Issue: GAS Optimization Report

Severity: QA

Where:

Impact:

Description:

- G-01: Use Immutables

The variables **evmERC20Token, nativeChainId,** and **evmChainId** can be set as immutable to save gas.

- G-02: Compact Structs

The **Deposit** and **Withdrawal** structs can be compacted. Currently, 7 storage slots are required per deposit or withdrawal. This can be reduced to 3 slots by removing redundant information and using smaller data types.
Example:

```
struct Deposit {
    address sender;
    address receiver;
    SourceChainEnum sourceChainId;// @audit G this struct could be packed (sourceChain and destinationChain coul
    uint128 amount;// @audit G we could assume amount is less than 2^128 and use uint128
    uint128 nonce;//@audit G we could assume amount is less than 2^128 and use uint128
}
```

- G-03: Remove Unnecessary Checks

- The native token in EVM doesn't have a fee on transfer, making certain checks is redundant. SynapseBridge.sol#L231-L232
- The safeTransfer function ensures the token is transferred, eliminating the need for subsequent balance checks. SynapseBridge.sol#L558
- Checking if the native token balance is zero is unnecessary in certain contexts. SynapseBridge.sol#L536

- G-04: Eliminate nativeTokenBalance

The variable **nativeTokenBalance** is redundant. Instead, **address(this).balance** can be used directly.

- G-05: Split the Contract

Consider splitting the contract into two separate contracts: one for the bridge on the mainnet and another for the bridge on the Humans chain. This will:

- Reduce the gas cost for deploying the bridge contract.
- Remove unnecessary functions, reducing the risk of user errors.
- Simplify the contract, leading to potential gas savings and a reduced attack surface.

- G-06: Avoid Comparisons with 0 for Unsigned Integers

Unsigned integer types (**uint**) cannot be negative. Therefore, comparisons like **<= 0** are unnecessary. Instead, use **== 0.**
Affected Lines:

- 180: if (amount <= 0) revert InvalidAmount();
- 228: if (msg.value <= 0) revert InvalidAmount();
- 278: if (amount <= 0) revert InvalidAmount();
- 354: if (amount <= 0) revert InvalidAmount();
- 438: if (amount <= 0) revert InvalidAmount();
- 523: if (balance <= 0) revert InsufficientAvailableBalance();
- 547: if (balance <= 0) {

- G-07: Avoid extra function like, _msgSender()

You are not using metatransactions, a raw **msg.sender** will be cheaper since is a variable less, and the **msg.sender** opcode is cheaper than a **push.**

- G-08: Avoid extra function like, getChainID()

Since solidity 0.8 you can just use **block.chainid,** see https://docs.soliditylang.org/en/v0.8.0/units-and-global-variables.html

Status: Resolved.

Finding: #25

Issue: Use latest Solidity version with a stable pragma statement

Severity: Low

Where:
https://github.com/ToDy95/bridgeAuditone/blob/f8016c67749fbdf7345c3987f79fcd95616bcf2f/contracts/contracts/SynapseBridge.sol#L2

Impact: An outdated compiler version that might introduce bugs that affect the contract system negatively.

Description: The project is using solidity version **^0.8.18.**
Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.
Reference - https://swcregistry.io/docs/SWC-103

Recommendations: Lock the pragma version and use the latest version of Solidity like **0.8.19** or higher.
Also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Status: Resolved.

08 - Disclaimer

The smart contracts provided to AuditOne have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). The ethical nature of the project is not guaranteed by a technical audit of the smart contract. Any owner-controlled functions should be carried out by the responsible owner. Before participating in the project, all investors/users are recommended to conduct due research.

The focus of our assessment was limited to the code parts associated with the items defined in the scope. We draw attention to the fact that due to inherent limitations in any software development process and product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which cannot be free from any errors or failures. These preconditions can impact the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure, which adds further inherent risks as we rely on correctly executing the included third-party technology stack itself. Report readers should also consider that over the life cycle of any software product, changes to the product itself or the environment in which it is operated can have an impact leading to operational behaviors other than initially determined in the business specification.

Contact

auditone.io

@auditone_team

hello@auditone.io

A trust layer of our multi-stakeholder world.