CHAIN TROOPERS

# Axelar Network
# Satellite & SDK
# Security Assessment Report

April 25th, 2022

Version 1.0

CONFIDENTIAL

# Table of Contents

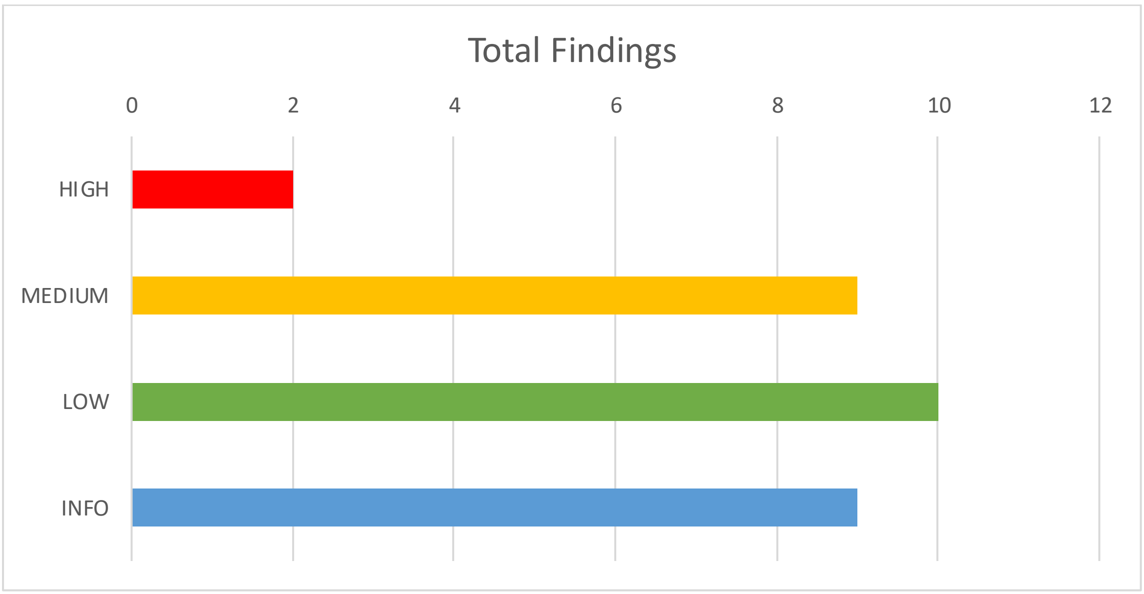
# 1 Executive Summary

## 1.1 Introduction

The report contains the results of Axelar Satellite App and SDK security assessment that took place from April 14th, 2022, to April 22nd, 2021. The security engineers performed an in-depth manual analysis of the provided functionalities, and uncovered issues that may be used by adversaries to affect the confidentiality, the integrity, and the availability of the in-scope components.

All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity

In total, the team identified twenty-one (21) vulnerabilities. There were also nine (9) informational issues of no-risk.



All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity.

## 1.2 Assessment Results

The assessment results revealed that the in-scope application components were mainly vulnerable to two (2) Information Disclosure and Authentication issue of HIGH risk. Regarding the Information Disclosure issue, it was found that the AlexarJS SDK was storing the generated mnemonic phrase of the wallet in the localStorage of the browser in plaintext *("5.1.1 - [SDK] Mnemonic wallet stored in plaintext in local browser storage")*. An adversary who could execute JavaScript code in the context of the page or had local access in the user's workstation would be able to extract the specific phrase and steal the wallet. In reference to Authentication issue, it was found that the generated One Time Code (OTC) for a wallet address could be overwritten by adversary who was aware of the public address of the locally generated wallet *("5.1.2 - [Nest] It is possible to overwrite the OTC for a specific address*[Nest] It is possible to overwrite the OTC for a specific address*")*. This issue could be abused to block the legitimate owner of the wallet from being able to create a transaction.

The in-scope components were also affected by nine (9) Authentication, Transmission Security, Information Disclosure and Data Validation vulnerabilities of MEDIUM risk. Regarding the MEDIUM risk Authentication issues, it was found that the string comparison that is used for the signature verification is vulnerable to timing attacks *("5.2.1 - [Nest] Timing attacks are possible in signature verification")*, allowing adversaries to conduct brute force attacks efficiently. Furthermore, it was found that the password-authentication control for the Redis instance is not enabled ("5.2.2 - [Nest] Unauthenticated Redis instance"*)*. Regarding the Transmission Security issues, the team identified that the certificate verification is disabled for Redis communication in both Nest and Nest worker *("5.2.3 - [Nest] Certificate verification is disabled for Redis communication even if a TLS/SSL URL is used", "5.2.4 - [Nest Worker] Certificate verification is disabled for Redis communication even if a TLS/SSL URL is used")*, while the currently deployed Redis instance does not support TLS/SSL at all *("5.2.5 - [Nest] TLS/SSL is not used in Redis Communication")*. Finally, the team identified that a number of weak ciphers were allowed in the TLS/SSL configuration of the Nest host *("5.2.9 - [Nest Host] Weak SSL/TLS configuration in-use")*.

In reference to the MEDIUM-risk Information Disclosure issues, it was identified that the OTC is exposed in the debug logs at the Nest host *("5.2.6 - [Nest] OTC leaked in debug logs")*, and the error messages of the Keplr Wallet were remotely transmitted to the backend service *(“5.2.8 - [Web] Potentially sensitive KeplrWallet error data are transmitted at the backend service”).* Finally, regarding to the Data Validation issues of MEDIUM risk, it was found that the OTC can be reused due to a , time-of-check to time-of-use (TOCTOU, TOCTTOU or TOC/TOU) issue *(“5.2.7 - [Nest] OTC Re-use is possible due to TOCTOU”).*

There were also ten (10) vulnerabilities of LOW risk and nine (9) findings of no-risk (INFORMATIONAL). Chaintroopers recommend the immediate mitigation of all HIGH and MEDIUM-risk issues. It is also advisable to address all LOW and INFORMATIONAL findings to enhance the overall security posture of the components.

## 1.3 Summary of Findings

The following findings were identified in the examined source code:

| Vulnerability Name | Status | Page |
|---|---|---|
| [SDK] Mnemonic wallet stored in plaintext in local browser storage | HIGH | 16 |
| [Nest] It is possible to overwrite the OTC for a specific address | HIGH | 19 |
| [Nest] Timing attacks are possible in signature verification | MEDIUM | 23 |
| [Nest] Unauthenticated Redis instance | MEDIUM | 25 |
| [Nest] Certificate verification is disabled for Redis communication even if a TLS/SSL URL is used | MEDIUM | 27 |
| [Nest Worker] Certificate verification is disabled for Redis communication even if a TLS/SSL URL is used | MEDIUM | 30 |
| [Nest] TLS/SSL is not used in Redis Communication | MEDIUM | 32 |
| [Nest] OTC leaked in debug logs | MEDIUM | 34 |
| [Nest] OTC Re-use is possible due to TOCTOU | MEDIUM | 37 |
| [Web] Potentially sensitive KeplrWallet error data are transmitted at the backend service | MEDIUM | 40 |
| [Nest Host] Weak SSL/TLS configuration in-use | MEDIUM | 45 |
| [SDK] Debug information in public repository | LOW | 47 |
| [SDK] Unvalidated user-provided parameters used in outgoing requests | LOW | 50 |
| [Nest] Log injection in "logMessageController" is possible | LOW | 54 |
| [Nest] Single standalone node of Redis in-use | LOW | 58 |

| | | |
|---|---|---|
| [SDK] HTTPS not enforced in Rest Service | LOW | 61 |
| [SDK] WSS not enforced in Web Socket Service | LOW | 64 |
| [Nest] Logging levels are not changed in production mode | LOW | 67 |
| [Web] Autocomplete is not disabled at password input | LOW | 70 |
| [Web] Hardcoded shortened URL in-use | LOW | 72 |
| [Nest] OTC is stored unencrypted in Redis database | LOW | 74 |
| [Nest] Spoofable client IP address due to custom decorator | INFO | 77 |
| [Nest] Multiple logging mechanisms are used | INFO | 80 |
| [Nest] Excessive signature verification attempts are allowed per wallet address | INFO | 83 |
| [Nest] HTTP Strict Transport Security (HSTS) not used | INFO | 85 |
| [Web] HTTP Strict Transport Security (HSTS) not used | INFO | 88 |
| [Web] Frame attacks (Click-Jacking) are not prevented | INFO | 90 |
| [Nest] MIME sniffing is allowed | INFO | 92 |
| [Web] Debug routes in production | INFO | 97 |
| [Web] Third-party log service "datadoghq" in use | INFO | 100 |

# 2 Assessment Description

## 2.1 Target Description

Axelar network is a scalable cross-chain communication platform. Blockchain platform builders can use it to seamlessly plug-in their blockchains to all other blockchain ecosystems. Application developers can choose the best blockchain to host their applications and use Axelar's cross-chain communication protocols to lock, unlock, and transfer assets, as well as communicate with applications on any other chain.

The AxelarJS SDK empowers developers to make cross-chain transfers using the Axelar network from their frontend. The Satellite web app is used to facilitate cross-chain communication on Axelar Network.

## 2.2 In-Scope Components

- Web App https://github.com/axelarnetwork/axelar-web-app/tree/develop
- JS SDK https://github.com/axelarnetwork/axelarjs-sdk/tree/0.5.0-alpha.11
- Nest https://github.com/axelarnetwork/axelar-bridge-nestjs
- Nest Worker https://github.com/axelarnetwork/axelar-bridge-worker-nestjs
- Host https://satellite.axelar.network/

| Component | Commit Identifier |
|---|---|
| *Web App* | *04750a4af9b33289445251baa336998df62de12c (origin/develop)* |
| *SDK* | *547a562ac47e3a5ab1e7e335adca838db8bdf610 (origin/0.5.0-alpha.11)* |
| *Nest* | *516d9c23dc21b71f4baed92aea43033b9d07a581 (origin/main)* |

| | |
|---|---|
| *Nest Worker* | *b5b70773dbc6b5a38ee21cc6b4d10f961 ff2aa0c (origin/main)* |

# 3 Methodology

## 3.1 Assessment Methodology

Chaintroopers' methodology attempts to bridge the penetration testing and source code reviewing approaches in order to maximize the effectiveness of a security assessment.

Traditional pentesting or source code review can be done individually and can yield great results, but their effectiveness cannot be compared when both techniques are used in conjunction.

In our approach, the application is stress tested in all viable scenarios though utilizing penetration testing techniques with the intention to uncover as many vulnerabilities as possible. This is further enhanced by reviewing the source code in parallel to optimize this process.

When feasible our testing methodology embraces the Test-Driven Development process where our team develops security tests for faster identification and reproducibility of security vulnerabilities. In addition, this allows for easier understanding and mitigation by development teams.

Chaintroopers' security assessments are aligned with OWASP TOP10 and NIST guidance.

This approach, by bridging penetration testing and code review while bringing the security assessment in a format closer to engineering teams has proven to be highly effective not only in the identification of security vulnerabilities but also in their mitigation and this is what makes Chaintroopers' methodology so unique.

## 3.2 Web Application Assessment

Chaintroopers' web application testing methodology used is based on the latest version of OWASP TOP 10 (https://owasp.org/www-project-top-ten/). This approach is enhanced by incorporating best practices for the specific technologies used by the target application, system or source code.

The following is a non-exhaustive list of security vulnerabilities that are identified by our methodology during the examination of the in-scope application or source code:

- Broken access control
- Cryptographic failures
- Injection
- Insecure design
- Security misconfigurations
- Vulnerable and outdated components
- Identification and authentication failures
- Software and data integrity failures
- Security logging and monitoring failures
- Server-side request forgery (SSRF)

## 3.3 Smart Contracts

The testing methodology used is based on the empirical study "Defining Smart Contract Defects on Ethereum" by J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo and T. Chen, in IEEE Transactions on Software Engineering, and the security best practices as described in "Security Considerations" section of the solidity wiki.

The following is a non-exhaustive list of security vulnerabilities that are identified by our methodology during the examination of the in-scope contract:

- Unchecked External Calls
- Strict Balance Equality
- Transaction State Dependency
- Hard Code Address
- Nested Call
- Unspecified Compiler Version
- Unused Statement
- Missing Return Statement

- Missing Reminder
- High Gas Consumption Function Type
- DoS Under External Influence
- Unmatched Type Assignment
- Re-entrancy
- Block Info Dependency
- Deprecated APIs
- Misleading Data Location
- Unmatched ERC-20 standard
- Missing Interrupter
- Greedy Contract
- High Gas Consumption Data Type

# 4 Scoring System

## 4.1 CVSS

All issues identified as a result of Chaintroopers' security assessments are evaluated based on Common Vulnerability Scoring System version 3.1 (https://www.first.org/cvss/).

With the use of CVSS, taking into account a variety of factors a final score is produced ranging from 0 up to 10. The higher the number goes the more critical an issue is.

The following table helps provide a qualitative severity rating:

| Rating | CVSS Score |
|---|---|
| None/Informational | 0.0 |
| Low | 0.1-3.9 |
| Medium | 4.0-6.9 |
| High | 7.0-8.9 |
| Critical | 9.0-10.0 |

Issues reported in this document contain a CVSS Score section, this code is provided as an aid to help verify the logic of the team behind the evaluation of a said issue. A CVSS calculator can be found in the following URL:

https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

# 5 Identified Findings

## 5.1 High Severity Findings

### *5.1.1 [SDK] Mnemonic wallet stored in plaintext in local browser storage*

| Description | HIGH |
|---|---|

The team identified that the SDK stores the generated mnemonic phrase of the wallet in the *localStorage* of the browser in plaintext. A secret recovery phrase, also called a mnemonic or seed phrase is a set of typically either 12 or 24 words which can be used to derive the wallet's private key.

In the specific case, the "*ethers.Wallet.createRandom*" function is used to create a new Wallet with a random private key, generated from cryptographically secure entropy sources. Wallets created using this method will have a mnemonic phrase. Then, the mnemonic is stored unprotected in the *localStorage* of the browser. To restore the wallet from the browser's *localStorage*, the "*ethers.Wallet.fromMnemonic*" function is used. Unfortunately, the "*localStorage*" is not a secure location to store sensitive data in plaintext.

The issue exists in the following location:

```
File: /axelarjs-sdk-0.5.0-alpha.11/src/utils/wallet.ts
export function createWallet() {
  if (globalThis.localStorage) {
    const mnemonic = globalThis.localStorage.getItem("axelar-wallet");
    if (mnemonic) {
      const wallet = ethers.Wallet.fromMnemonic(mnemonic);
      return wallet;
    } else {
      const wallet = ethers.Wallet.createRandom();
      globalThis.localStorage.setItem(
        "axelar-wallet",
        wallet._mnemonic().phrase
      );
      return wallet;
    }
  } else {
    const wallet = ethers.Wallet.createRandom();
    return wallet;
  }
}
```

Initially, the algorithm checks if the "*localStorage*" exists in the global object using the "*globalThis*". The "*LocalStorage*" is an HTML5 web storage object for storing data on the client and has no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year. The "*globalThis*" consolidate the fragmented ways of accessing the global object. In general, each environment has its own object model and provides a different syntax to access the global object. For example, in the web browser, the global object is accessible via "window", "self", or "frames", while in Node.js, the "global" is used instead. All popular browsers, including Chrome 71+, Firefox 65+, and Safari 12.1+, and Node.js 12+ support the "*globalThis*" to access the global object.

However, the "*LocalStorage*" does not provide more security that the normal HTTP Cookies and is not a recommended location to store sensitive data. Initially, any JavaScript payload that executes in the context of the page can access the stored information. In case that an adversary can perform a single Cross Site Scripting attack (XSS), it would be possible to extract all the data from the storage. Furthermore, any user who has local access in the workstation and the browser has full control on the *LocalStorage*. For example, an adversary who gains temporal local access in the browser (e.g. as part of an evil-maid attack), can extract the mnemonic from the browser's *localStorage*.

## Impact

An adversary who is able to execute JavaScript code in the context of the page (e.g. as part of a Cross Site Scripting attack - XSS) or is able to gain local access in the user's workstation (e.g. in a shared workstation environment), will be able to extract the mnemonic from the *localStorage* of the wallet that was created using the examined SDK. The attack will work even if the victim has previously terminated the browser session.

## Recommendation

It is recommended to avoid storing the wallet's mnemonic in the client side. If this is not possible, it is recommended to store the data in the local storage of the browser in an encrypted JSON format, which would require a password provided by the user to be decrypted and be used. The "*Wallet.fromEncryptedJson*()" function can be used to derive the wallet.

For example, a similar implementation is described in the following URL:

> *https://github.com/Dreammaster13/blockchain-for-devs-book/blob/master/content/part-2-dapps-ethereum-and-solidity/dapp-architectures/exercises-client-side-wallets/client-side-ethereum-wallets.md*

Finally, it is recommended to use the object *sessionStorage* instead of *localStorage* if persistent storage is not needed. *sessionStorage* object is available only to that window/tab until the window is closed.

## CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### *5.1.2 [Nest] It is possible to overwrite the OTC for a specific address*

**Description** **HIGH**

It was found that an adversary can cancel a generated One Time Code (OTC) for a wallet address, by requesting a new one and overwriting the previous one. In general, the application is using OTC to verify that a transaction originates from a specific user and avoid double transactions. However, since a new OTC is generated at each API call, an adversary can abuse this issue to block a legitimate user from completing a transaction.

The issue exists in the following location:

**File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts**

```
async getOneTimeCode(dto: GetOtcDto, traceId?: string): Promise<OTC> {
    try {
      const newOTC = this.utils.getOTC();
      await this.cache.set<string>(dto.publicAddress, newOTC);

      this.logger.debug({
        from: this.getOneTimeCode.name,
        traceId: traceId || 'none',
        newOTC,
        dto,
      });

      return {
        validationMsg: this.utils.getOTCMessage(newOTC),
        otc: newOTC,
      };
    } catch (error) {
      this.logger.debug({
        from: this.getOneTimeCode.name,
        traceId: traceId || 'none',
        error,
        dto,
      });
```

```
        throw error;
      }
    }

```

For example, it is possible to repeat the following HTTP request to block the legitimate user who owns the wallet with the specific public address to perform a transaction:

```
GET /otc?publicAddress=0x5c892C4cFbf8944DE65d30814d494855e51DB0a2
HTTP/1.1
Host: nest-server-testnet.axelar.dev
X-Trace-Id: 63e06795-819a-4db4-90ac-7d5a14e29524
Dnt: 1
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.88 Safari/537.36
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="100", "Google
Chrome";v="100"
Sec-Ch-Ua-Platform: "macOS"
Content-Type: application/json
Accept: */*
Origin: https://bridge.testnet.axelar.dev
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://bridge.testnet.axelar.dev/
Accept-Language: el-GR,el;q=0.9
```

And the HTTP response would be:

```
HTTP/1.1 200 OK
Date: Sun, 17 Apr 2022 19:43:23 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 127
Connection: keep-alive
CF-Ray: 6fd7a63b4e7d6f43-ATH
```

```
Access-Control-Allow-Origin: *
ETag: W/"7f-XTp0kVffvZN4olvYvfWYYL6iotM"
Strict-Transport-Security: max-age=15552000; includeSubDomains
Via: 1.1 vegur
CF-Cache-Status: DYNAMIC
Content-Security-Policy: default-src 'self';base-uri 'self';block-all-mixed-content;font-src 'self' https: data:;form-action 'self';frame-ancestors 'self';img-src 'self' data:;object-src 'none';script-src 'self';script-src-attr 'none';style-src 'self' https: 'unsafe-inline';upgrade-insecure-requests
Cross-Origin-Embedder-Policy: require-corp
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Expect-Ct: max-age=0
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Ratelimit-Limit: 10
X-Ratelimit-Remaining: 7
X-Ratelimit-Reset: 41
X-Xss-Protection: 0
Set-Cookie: __cf_bm=A812bY7XITIdAv5iT7qer5wm5ZaVSwc9riLBj5qnKxI-1650224603-0-ASjRQ2i/+rYnPkkcJTRPTFJA/j0aE0LsQ7nIcWBu3HyMBNE3k64t+PTw3iVD4xCMKTzpVYB6P4TN58YkkqyMMkQ=; path=/; expires=Sun, 17-Apr-22 20:13:23 GMT; domain=.axelar.dev; HttpOnly; Secure; SameSite=None
Server: cloudflare

{"validationMsg":"Verify I'm a real user with this one-time-code: iRNnz3W9cq (This will not cost any fees)","otc":"iRNnz3W9cq"}
```

## Impact

An adversary, who is aware of the public address of the locally generated wallet for a user, can perform multiple HTTP requests for generating new OTC codes for the specific address and block the legitimate owner of the wallet from being able to create a transaction. As a result, this issue can lead to a Denial of Service (DoS) attack.

It should be noted that it is very difficult for an adversary to be able to predict the public address of the generated wallet for a user that would be used for the first time. However, due to the issue 1002 the wallet is stored in the browser's local storage and does not change. As a result, if the adversary monitors the transactions, it would be very easy to target a random user and block them from further transactions.

## Recommendation

The issue can be mitigated in multiple ways. By returning the same OTC every time until it is eventually consumed in a valid transaction, the attack won't be possible. However, in this case the adversary would also be able to retrieve a valid OTC code that would be used by a user.

As a result, it is also advisable to change the "getOneTimeCode" to an authenticated call in which the user will have to submit a signed timestamp with its locally generated wallet.

Furthermore, the risk could be reduced to LOW by mitigating issue 1002 and avoid using the same wallet more than one time.

## CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

## 5.2 Medium Severity Findings

### *5.2.1 [Nest] Timing attacks are possible in signature verification*

**Description** **MEDIUM**

The team identified that the string comparison that is used for the signature verification is vulnerable to timing attacks. A timing attack is a side channel attack which allows adversaries to infer potentially sensitive information from a targeted web application by observing the normal behavior of the response times for different tasks.

In a web app, processing time may leak information on what is being done in the application/system background processes. For example, in the specific case, the platform provided string comparison with the "==" or "===" works by iterating the two (equal length) strings, comparing one character at a time, and stopping when a character differs. So, if a user wants to compare the string "foo" and the string "bar", the loop will iterate once, while if the user wants to compare the string "foo" and the string "fox" the loop would iterate three times to compare all three (3) chars, taking longer to complete.

An adversary who can observe the difference in the response times may be able to brute force the bytes of the signature "one - by - one" reducing the overall possible combinations. So instead of having to brute force a string with total combinations as many as the "possible number of characters" ^ "string length", the adversary has to brute force the significantly smaller "possible number of characters" * "string length".

Based on previous research (https://www.cs.rice.edu/~dwallach/pub/crosby-timing2009.pdf), it is known that adversaries can measure events with a 15-100μs accuracy over the internet.

The issue exists in the following location:

```
File:                   axelar-bridge-nestjs-main/src/bridge/service/bridge-
utils.service.ts
```

```
  verifySignature(otc: string, signature: string, address: string): boolean {
    const message = this.getOTCMessage(otc);

    try {
      const signinAddress = verifyMessage(message, signature);
      if (address !== signinAddress) return false;

      return true;
    } catch (error) {
       ...
    }
  }
```

## Impact

A diligent adversary can exploit this issue to conduct an efficient brute force attack for a signature string with significantly less possible combinations.

## Recommendation

It is advisable to implement a constant-time comparison. For example, the constant-time "crypto.timingSafeEqual()" function can be used. More information can be found in the following URL:

*https://nodejs.org/dist/latest-v6.x/docs/api/crypto.html#crypto_crypto_timingsafeequal_a_b*

## CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.2.2 [Nest] Unauthenticated Redis instance

**Description** **MEDIUM**

The team identified that the password-authentication control for the Redis instance is not enabled. As it is mentioned in Redis documentation, Redis is designed to be accessed by trusted clients inside trusted environments. This means that usually it is not a good idea to expose the Redis instance directly to the internet or, in general, to an environment where untrusted clients can directly access the Redis TCP port or UNIX socket. In general, untrusted access to Redis should always be mediated by a layer implementing ACLs, validating user input, and deciding what operations to perform against the Redis instance. Unfortunately, many users fail to protect Redis instances from being accessed from external networks. Many instances are simply left exposed on the internet with public IPs. In other cases, adversaries may manage to compromise another system in the internal network, and use it as a pivot point to attack Redis.

While Redis does not try to implement Access Control, it provides a tiny layer of optional authentication that is turned on by editing the redis.conf file. When the authorization layer is enabled, Redis will refuse any query by unauthenticated clients. A client can authenticate itself by sending the AUTH command followed by the password.

The issue exists in the following location:

**File: axelar-bridge-nestjs-main/docker-compose.yml**

```
version: '3.9'
services:
  redis:
    image: redis:6
    ports:
      - 6379:6379
  redis-commander:
    container_name: redis-commander
    hostname: redis-commander
    image: rediscommander/redis-commander:latest
    environment:
```

```
    - REDIS_HOSTS=local:redis:6379
    ports:
    - 8081:8081
```

## Impact

An adversary, who is located in the adjacent network between the nodes (e.g. Nest worker and Redis Database, or Nest app and Redis database), will be able to access Redis and tamper with the stored records.

For example, the adversary would be able to delete stored OTC codes, in order to perform a denial-of-service attack, or inject fake deposit messages.

## Recommendation

It is recommended to

- block the access to the HTTP management interface
- use a strong password authentication for the Redis service

```
redis:
    image: [name:version]
    command: redis-server --requirepass [password]
    ...
redis-commander:
    ...
    environment:
    - REDIS_PASSWORD=[password]
```

## CVSS Score

AV:A/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.2.3 [Nest] Certificate verification is disabled for Redis communication even if a TLS/SSL URL is used

**Description** **MEDIUM**

It was identified that the certificate verification is disabled even if a Redis TLS/SSL URL is provided. Transport Layer Security (TLS) uses encryption to secure network communication and is supported by Redis starting with version 6 as an optional feature that needs to be enabled at compile time.

More precisely, in the specific case the "*rejectUnauthorized*" attribute is defined for the Redis connection, even when the TLS/SSL link is provided. If the "*rejectUnauthorized*" attribute is true, the server certificate is verified against the list of supplied CAs, and an 'error' event is emitted if verification fails; the 'error' event contains the OpenSSL error code. However, when the attribute is set to false, the certificate will not be validated, compromising the security of the SSL/TLS connection.

The issue exists in the following location:

**File: axelar-bridge-nestjs-main/src/app.module.ts**
```
@Module({
  imports: [
    ConfigModule,
    CacheModule.registerAsync<any>({
      ...
      useFactory: (config: ConfigService) => {
        const url = config.get('REDIS_TLS_URL') || config.get('REDIS_URL');
        const redis = new Redis(url, {
          tls: {
            rejectUnauthorized: false,
          },
        });
        return {
        ...
        };
      },
    }),
    ThrottlerModule.forRootAsync({
      ...
      useFactory: (config: ConfigService) => {
        const url = config.get('REDIS_TLS_URL') || config.get('REDIS_URL');
        const redis = new Redis(url, {
          tls: {
            rejectUnauthorized: false,
          },
        });
        return {
         ...
        };
      },
    }),
    BridgeModule,
    HealthModule,
    SocketModule,
    LogModule,
    LoadtestingModule.register(),
  ],
})
```

## Impact

In case that TLS/SSL is enabled at Redis in production mode, then the certificate validation will still be disabled. As a result, an adversary in the intermediate channel will be able to eavesdrop or tamper with the exchanged data.

## Recommendation

It is advisable to remove the "rejectUnauthorized: false" configuration when the application is deployed in the production environment.

| CVSS Score |
|---|
| AV:A/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X |

### 5.2.4 *[Nest Worker] Certificate verification is disabled for Redis communication even if a TLS/SSL URL is used*

**Description** **MEDIUM**

It was identified that the certificate verification is disabled even if a Redis TLS/SSL URL is provided. Transport Layer Security (TLS) uses encryption to secure network communication and is supported by Redis starting with version 6 as an optional feature that needs to be enabled at compile time.

More precisely, in the specific case the "rejectUnauthorized" attribute is defined for the Redis connection, even when the TLS/SSL link is provided. If the "rejectUnauthorized" attribute is true, the server certificate is verified against the list of supplied CAs and an 'error' event is emitted if verification fails; err.code contains the OpenSSL error code. However, when the attribute is set to false, the certificate will not be validated, compromising the security of the SSL/TLS connection.

**File: axelar-bridge-worker-nestjs-main/src/redis.adapter.ts**

```
  async connectToRedis(): Promise<void> {
    if (this.pubClient?.isOpen && this.subClient?.isOpen) return;

    this.pubClient = createClient({
      url: this.config.get('REDIS_TLS_URL') || this.config.get('REDIS_URL'),
      socket: {
        tls: true,
        rejectUnauthorized: false,
      },
    });
```

**Impact**

In case that TLS/SSL is enabled at Redis in production mode, then the certificate validation will still be disabled. As a result, an adversary in the intermediate channel will be able to eavesdrop or tamper with the exchanged data.

## Recommendation

It is advisable to remove the "rejectUnauthorized: false" configuration when the application is deployed in the production environment.

## CVSS Score

AV:A/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### *5.2.5 [Nest] TLS/SSL is not used in Redis Communication*

**Description** **MEDIUM**

It was identified that TLS/SSL is not enabled for the communication with the Redis instance. TLS uses encryption to secure network communication and is supported by Redis starting with version 6 as an optional feature that needs to be enabled at compile time.

The issue exists in the following location:

```
File: axelar-bridge-nestjs-main/docker-compose.yml
version: '3.9'
services:
  redis:
    image: redis:6
    ports:
      - 6379:6379
  redis-commander:
    container_name: redis-commander
    hostname: redis-commander
    image: rediscommander/redis-commander:latest
    environment:
    - REDIS_HOSTS=local:redis:6379
    ports:
    - 8081:8081
```

**Impact**

An adversary, who is located in the adjacent network between the nodes (e.g., Nest worker and Redis Database, or Nest app and Redis database), will be able to eavesdrop or tamper with the exchanged data.

Furthermore, lack of TLS/SSL in the nodes' communication can introduce compliance issues.

## Recommendation

It is recommended to enable the TLS/SSL support for the connection with the Redis database.

It should be noted that this issue can also be mitigated by correcting the configuration of the final deployed environment (e.g., isolated network for the nodes). The exact environment details are currently not taken into account as part of the code review process.

Redis commander supports the parameter REDIS_TLS to enable TLS/SSL:

*https://github.com/joeferner/redis-commander/blob/master/docs/connections.md#connection-parameters*

## CVSS Score

AV:A/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### *5.2.6 [Nest] OTC leaked in debug logs*

**Description** **MEDIUM**

The team identified that sensitive information is included in the logging mechanism. Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. In the specific case it was found that when the application attempts to store logging information with "debug" level of the "getOneTimeCode" functionality, the "newOTC" is stored in the logs without being masked or encrypted.

The app is using the logger service of  Nest. Nest comes with a built-in text-based logger which is used during application bootstrapping and several other circumstances such as displaying caught exceptions (i.e., system logging). This functionality is provided via the Logger class in the @nestjs/common package:

```
File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
01: import {
...
08:   Logger,
...
10: } from '@nestjs/common';

Then, the logger service is initialized:

File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
21: @Injectable()
22: export class BridgeService {
23:   environment: string;
24:   private readonly logger = new Logger(BridgeService.name);
25:   ...
```

Finally, it is used to store information such as "newOTC":

```
File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
```

```
  async getOneTimeCode(dto: GetOtcDto, traceId?: string): Promise<OTC> {
    try {
      const newOTC = this.utils.getOTC();
        ...
      this.logger.debug({
        from: this.getOneTimeCode.name,
        traceId: traceId || 'none',
        newOTC,
        dto,
      });
...
    } catch (error) {
...
    }
  }

```

## Impact

In case that the application is deployed in the production environment, the OTC code will be disclosed to the default logging output. An adversary or a malicious insider (e.g., a disgruntled employee) that is able to gain access to this logging output, will be able to extract the OTC code and use it to affect the transaction. Furthermore, log records may be accidentally exposed (e.g., by a reckless employee) revealing any recorded sensitive information.

## Recommendation

It is advisable to remove or mask sensitive information from the logging output, even when it is used for debug purposes.

## CVSS Score

AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.2.7 *[Nest] OTC Re-use is possible due to TOCTOU*

**Description** **MEDIUM**

It was identified that the one-time code (OTC) can be reused due to a TOCTOU issue. Some applications use a second factor to check whether an authorized user is performing sensitive operations. A common example is wire transfer authorization, typically used in online or mobile banking applications. In software development, time-of-check to time-of-use (TOCTOU, TOCTTOU or TOC/TOU) is a class of software bugs caused by a race condition involving the checking of the state of a part of a system (such as a security credential) and the use of the results of that check. In general, the risk of TOCTOU vulnerabilities in Node.js is reduced to the event loop architecture which is mainly single-threaded based. Although such issues can still occur when there is a shared resource outside of the NodeJS environment (e.g., in the examined case Redis is used as a shared resource, however Redis is single threaded and concurrent actions are not allowed), that is not running in the same thread as node.js, or if multiple workers are utilized.

Currently, the cluster mode with two (2) workers is enabled:

**File axelar-bridge-nestjs-main/pm2.config.js**

```
module.exports = {
  apps: [
    {
      name: 'bridge-api',
      script: 'dist/src/main.js',
      exec_mode: 'cluster',
      instances: 2,
    },
  ],
};

```

More precisely, it was identified that in the "transferAssets" function, the application retrieves the OTC for the provided address from Redis cache in line 126 and verifies the signature in line 129. However, the OTC is not deleted from the Redis cache before line 226, allowing the possibility of reusing it for other transactions.

The issue exists in the following location:

```
File:  axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
124:   async transferAssets(dto: TransferAssetsDto_v2, traceId?: string) {
125:     // verify that the otc is still valid
126:     const otc = await this.getOtcFromAddress(dto.publicAddress);
127:
128:     // verify signature
129:     this.verifySignature(otc, dto.signature, dto.publicAddress);
130:
131:     /**
132:        * Verifies that the provided chains exist by comparing them against
133:      * the chains registered in the sdk. Additionally return the chain module
134:      * for later use
135:      */
        …
224:
225:     // clean cached otc to avoid double transaction
226:     await this.cache.del(dto.publicAddress);
227:
        …
247:
248:     return {
249:       data: {
250:         …
251:       },
252:     };
253:   }
```

## Impact

An adversary who is able to exploit this issue, would be able to replay a transaction with a valid signature. However, exploiting a TOCTOU race condition requires precise timing to ensure that the attacker's operations interleave properly with the legitimate server's operations.

Moreover, in the specific case, only two (2) NodeJS workers are allowed.

## Recommendation

It is advisable to perform actions atomically and don't rely on previously queried information about the OTC to perform actions later. For example, an atomic transaction to read and delete the OTC can be used. (https://redis.io/docs/manual/transactions/)

An alternative way is to avoid sharing a “global” state whenever possible, but especially with concurrency. If sharing state among concurrent routines is required, consider introducing a mutex. For example, a DLM (Distributed Lock Manager) implemented with Redis can be used

## CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.2.8 [Web] Potentially sensitive KeplrWallet error data are transmitted at the backend service

**Description** **MEDIUM**

It was identified that potentially sensitive error messages from Keplr wallet are transmitted at the backend application. Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. While logging all information may be helpful during development stages, it is important that any sensitive content is filtered or masked before a product is deployed so that any potentially sensitive user data and system information are not accidentally exposed to adversaries.

In this specific case, it was found that the application is using the "*SendLogsToServer.info()*" function to send the error exception of Keplr wallet to the backend service. Since the exact error messages are not controlled by the organization, they can contain any information that is related to the user and the wallet (e.g., user identifiers, wallet addresses, transaction history, memory data or even passwords). As a result, this information will be sent to the backend service, introducing compliance and confidentiality issues.

The issue exists at the following location:

```
File: axelar-web-app-develop/src/hooks/wallet/KeplrWallet.ts
      } catch (e2: any) {
        console.log("and yet there is a problem in trying to do that too", e2)
        SendLogsToServer.info(
          "KeplrWallet_connectToWallet",
          JSON.stringify(e2),
          "NO_UUID"
        )
        return text
      }
    }
```

The same issue exists at the following location:

**File: axelar-web-app-develop/src/pages/app/parts/swap-widget/TransactionStatusWindow/StatusList/DepositFromWallet.tsx**

```
    } catch (error: any) {
      setDepositAmount("")
      results = error
    }
    handleKeplrTxResult(results)
  }

  const handleKeplrTxResult = (results: any) => {
    // this is the case where you get immediate feedback in the results
    let stringifiedResults: string = results?.toString()?.toLowerCase() || ""

    // this is the case where the request is sent to the network and raw logs are returned, so we also want to check this for any of the below issues
    if (results?.rawLog) {
      stringifiedResults += results.rawLog.toString()
    }
    const outOfGas: boolean = stringifiedResults.includes("out of gas")
    const accountSequenceMismatch: boolean = stringifiedResults.includes(
      "account sequence mismatch"
    )
    const inSufficientFunds: boolean =
      stringifiedResults.includes("insufficient funds")
    const requestRejected: boolean =
      stringifiedResults.includes("request rejected")

    const hasAnyErrors =
      outOfGas ||
      accountSequenceMismatch ||
      inSufficientFunds ||
      requestRejected

    if (
      results &&
      (results.transactionHash || results.txhash) &&
      results.height >= 0 &&
      !hasAnyErrors
    ) {
      setSentSuccess(true)
      setTxHash(results.transactionHash || results.txhash)
      setDepositTimestamp(new Date().getTime())
      SendLogsToServer.info(
        "DEPOSIT_CONFIRMATION",
        "deposit made within app: " + results,
        transactionTraceId
      )
    } else {
      setButtonText("Something went wrong, try again?")
      const msg = "user failed to send tx: " + results
      SendLogsToServer.info("DEPOSIT_CONFIRMATION", msg, transactionTraceId)
    }
  }
```

Then, the logs will be transmitted at the following location:

**File: axelar-web-app-develop/src/api/SendLogsToServer.ts**

```
export class SendLogsToServer {
  private static NODE_SERVER_URL: string = getConfigs(
    process.env.REACT_APP_STAGE as string
  )?.resourceUrl
  private static LOGGING_ENDPOINT: string =
    SendLogsToServer.NODE_SERVER_URL + "/logMessageController"
  private static queuedLogs: []

  private static async baseSend(
    messageType: messageType,
    topic: string,
    message: string,
    traceId: string
  ) {
    const res = await axios.post(
      SendLogsToServer.LOGGING_ENDPOINT,
      { messageType, topic, message } as LogMessageObject,
      { headers: { "x-traceid": traceId } }
    )
    return res
  }
  public static error(topic: string, message: string, traceId: string)
{
    return SendLogsToServer.baseSend("error", topic, message, traceId)
  }
  public static info(topic: string, message: string, traceId: string)
{
    return SendLogsToServer.baseSend("info", topic, message, traceId)
  }
}

```

## Impact

In case that sensitive data are returned by the Keplr wallet in the error code, they will be transmitted to the backend service, introducing confidentiality and compliance issues. The data can range from simple error codes to user identifiers, wallet addresses, transaction history, memory data or even passwords.

## Recommendation

It is recommended to completely disable the remote logging functionality at production deployment. If this is not possible, it is recommended to parse the logs and send to the backend service only a number of expected error codes.

## CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### *5.2.9 [Nest Host] Weak SSL/TLS configuration in-use*

**Description** **MEDIUM**

It was identified that the "*nest-server-testnet.axelar.dev*" host is using a weak TLS/SSL configuration to protect the HTTP traffic. HTTP is a clear-text protocol, and it is normally secured via an SSL/TLS tunnel, resulting in HTTPS traffic. The use of this protocol ensures not only confidentiality, but also integrity and server authentication. Servers are authenticated using digital certificates and it is also possible to use client certificates for mutual authentication. However, it was found that the existing configuration had several weaknesses:

First, weak cipher suites were allowed for the TLS 1.2. Regarding the CBC mode, in 2013, researchers demonstrated a timing attack against several TLS implementations using the CBC encryption algorithm. A fix has been introduced with TLS 1.2 in the form of the GCM mode which is not vulnerable to the BEAST attack. GCM should be preferred over CBC.

- *TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023) ECDH x25519 (eq. 3072 bits RSA) FS WEAK 128*
- *TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024) ECDH x25519 (eq. 3072 bits RSA) FS WEAK 256*
- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027) ECDH x25519 (eq. 3072 bits RSA) FS WEAK 128*
- *TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c) WEAK 128*
- *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028) ECDH x25519 (eq. 3072 bits RSA) FS WEAK 256*
- *- TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d) WEAK 256*

While the following cipher modes support SHA1, the SHA1 has been proven to be insecure as of 2017 (see shattered.io):

- *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH x25519 (eq. 3072 bits RSA) FS WEAK 128*
- *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH x25519 (eq. 3072 bits RSA) FS WEAK 256*

- *TLS_RSA_WITH_AES_256_CBC_SHA (0x35)   WEAK       256*
- *TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)   WEAK       128*
- *TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)   ECDH secp256r1 (eq. 3072 bits RSA)   FS   WEAK       128*

Moreover, the following ciphers do not support Perfect Forward Secrecy (PFS) which is recommended, so attackers cannot decrypt the complete communication stream.

- *TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)   WEAK       128*
- *TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)   WEAK       256*

**Impact**

An adversary who is located in the adjacent network (e.g., ISP, Wi-Fi hotspot), may be able to exploit these issues to intercept or eavesdrop the exchanged data.

**Recommendation**

It is recommended to disable the support for the specific ciphersuites in the host server.

**CVSS Score**

AV:A/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

## 5.3 Low Severity Findings

### *5.3.1 [SDK] Debug information in public repository*

**Description** **LOW**

It was identified that debug information is uploaded in a public repository. It is very common for developers to include detailed comments, debug logs and other metadata in their source code. However, logs and metadata included into the repository might reveal internal information that should not be available to potential attackers. In this specific case, an error log file was found containing the name of the workstation user account in which the software was tested.

The issue exists in the following location:

```
File: axelarjs-sdk-0.5.0-alpha.11/yarn-error.log
Arguments:
  /Users/rpzy/.nvm/versions/node/v14.17.6/bin/node /Users/rpzy/.nvm/versions/node/v14.17.6/bin/yarn compile

PATH:
  /Users/rpzy/.nvm/versions/node/v14.17.6/bin:/Users/rpzy/bin:/usr/local/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/go/bin:/opt/X11/bin:/Library/Apple/usr/bin:/Users/rpzy/.nvm/versions/node/v14.17.6/bin:/Users/rpzy/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/Users/rpzy/.cargo/bin:/Users/rpzy/.fzf/bin:/Applications/Visual Studio Code - Insiders.app/Contents/Resources/app/bin:/Users/rpzy/go/bin:/Users/rpzy/.cargo/bin:/usr/local/go/bin:/bin:/Applications/Visual Studio Code - Insiders.app/Contents/Resources/app/bin:/Users/rpzy/go/bin

Yarn version:
  1.22.17

Node version:
  14.17.6

```

```
Platform:
  darwin arm64

Trace:
      SyntaxError: /Users/rpzy/workspace/axelarjs-sdk/package.json: Unexpected token
   in JSON at position 754
      at JSON.parse (<anonymous>)
      at /Users/rpzy/.nvm/versions/node/v14.17.6/lib/node_modules/yarn/lib/cli.js:1625:59
      at Generator.next (<anonymous>)
      at step (/Users/rpzy/.nvm/versions/node/v14.17.6/lib/node_modules/yarn/lib/cli.js:310:30)
      at /Users/rpzy/.nvm/versions/node/v14.17.6/lib/node_modules/yarn/lib/cli.js:321:13
```

## Impact

Revealing system data or debugging information helps an adversary learn about the system and perform further attacks. For example, in the specific case the "*rpzy*" user account could be used in phishing or social engineering attacks.

## Recommendation

It is recommended to add all the log files in "*.gitignore*" and remove the specific file from git history. A *gitignore* file specifies intentionally untracked files that Git should ignore. For example:

```
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

More information can be found in the following URL:

*https://www.toptal.com/developers/gitignore/api/node*

To entirely remove unwanted files from a repository's history you can use either the git filter-repo tool or the BFG Repo-Cleaner open-source tool. GitHub provides detailed information on how to remove sensitive data from a repository, that can be found in the following URL:

*https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository#purging-a-file-from-your-repositorys-history%E2%80%A8*

| CVSS Score |
|---|
| AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N/E:F/RL:X/RC:R/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X |

### 5.3.2 [SDK] Unvalidated user-provided parameters used in outgoing requests

**Description** **LOW**

It was found that several unvalidated user-provided parameters are used to construct URIs and to perform outgoing HTTP requests. Input validation is the first step in sanitizing the type and content of user-provided data. It is a technique for identifying potentially dangerous inputs and for ensuring that the user-provided inputs are safe for processing within the code, or when communicating with other third-party components. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

In the examined case, when the "*AxelarAssetTransfer*" functionality is trying to obtain the "Fee" or the "OTC" from the Rest service, the user-provided parameters "chain", "asset", and "signerAddress" are directly injected in the constructed URI. An adversary who is able to add special symbols ("#", "\r\n", "&", etc.) will be able to tamper with the URL and affect the component's operation. The issue exists in the following locations:

```
File: axelarjs-sdk-0.5.0-alpha.11/src/libs/AxelarAssetTransfer.ts
  public async getFeeForChainAndAsset(
    chain: string,
    asset: string
  ): Promise<any> {
    return this.api
      .get(`${CLIENT_API_GET_FEE}?chainName=${chain}&assetCommonKey=${asset}`)
      .then((response) => response)
      .catch((error) => {
        throw error;
      });
  }

File: axelarjs-sdk-0.5.0-alpha.11/src/libs/AxelarAssetTransfer.ts
```

```
  public async getOneTimeCode(
    signerAddress: string,
    traceId: string
  ): Promise<OTC> {
    const otc: OTC = await this.api
        .get(`${CLIENT_API_GET_OTC}?publicAddress=${signerAddress}`,
traceId)
      .then((response) => response)
      .catch((error) => {
        throw error;
      });

    return otc;
  }
```

A similar issue exists in the "RestService" interface, in which the "traceid" parameter is directly injected in the outgoing HTTP packet headers:

```
File: axelar/axelarjs-sdk-0.5.0-alpha.11/src/services/RestService.ts
  post(url: string, body: any, traceId?: string) {
    const requestOptions = {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "x-trace-id": traceId || "none",
      },
      body: JSON.stringify(body),
    };

    return this.execRest(url, requestOptions);
  }

  get(url: string, traceId?: string) {
    const requestOptions = {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        "x-trace-id": traceId || "none",
      },
    };

    return this.execRest(url, requestOptions);
  }
```

## Impact

An adversary, who is able to provide specially crafted parameters in the reported functionalities, will be able to tamper with the constructed URLs and affect the SDK operations. For example, certain symbols will allow the adversary to inject or remove specific HTTP parameters, to inject arbitrary headers, etc.

## Recommendation

It is recommended to always validate any user-provided input before processing them in the application code. For example, in the examined cases only alphanumeric characters should be sufficient for the application operation. As a result, the following regular expression can be used to verify that no symbols exist:

```
/^[a-z0-9]+$/i
```

Moreover, it is advisable to perform escaping on all HTML elements before they are presented to the user interface or encoding before used in external functionalities. Escaping can be performed using HTML-encode, by replacing the offending symbol with its corresponding HTML entities. In the specific case, the parameters should be URL encoded before appended in the constructed URL:

```
encodeURI()
```

| CVSS Score |
|---|
| AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X |

### 5.3.3 [Nest] Log injection in "logMessageController" is possible

**Description** **LOW**

The team identified that the app is vulnerable to log injection attacks since the custom logging mechanism allows users to supply arbitrary log records. Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. Writing invalidated user input to log files can allow an attacker to forge log entries or inject malicious content into the logs. Successful log injection attacks can cause:

- Injection of new/bogus log events (log forging via log injection)
- Injection of XSS attacks, hoping that the malicious log event is viewed in a vulnerable web application
- Injection of commands that parsers could execute

The issue is located at the following file in which the "logMessageController" HTTP POST request is defined:

```
File: axelar-bridge-nestjs-main/src/log/log.controller.ts
import { Controller, Post, Headers, Body } from '@nestjs/common';
import { LogPayload } from './dto';
import { LogService } from './log.service';

@Controller('logMessageController')
export class LogController {
  constructor(private logService: LogService) {}

  @Post()
  log(@Headers('x-traceid') traceId: string, @Body() dto: LogPayload) {
    return this.logService.log(dto, traceId);
  }
}

```

The the LogService is defined at the following location:

```
File: axelar-bridge-nestjs-main/src/log/log.service.ts
import { Injectable } from '@nestjs/common';
import { LogPayload } from './dto';

@Injectable()
export class LogService {
  log(dto: LogPayload, traceId: string) {
    console.log({
      timestamp: new Date(),
      message: dto.message.replace(/(\t)\s+/g, '$1'),
      trace_id: traceId,
      level: dto.messageType,
    });
  }
}

```

For example, the following HTTP POST request was used to create a fake log message:

```
POST /logMessageController HTTP/1.1
Host: nest-server-testnet.axelar.dev
Content-Length: 63
X-Trace-Id: bzxczx
Dnt: 1
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.88 Safari/537.36
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
Sec-Ch-Ua-Platform: "macOS"
Content-Type: application/json
Accept: */*
Origin: https://bridge.testnet.axelar.dev
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
```

```
Sec-Fetch-Dest: empty
Referer: https://bridge.testnet.axelar.dev/
Accept-Language: el-GR,el;q=0.9

{"messageType":"debug",
"topic":"injected",
"message":"test"}
```

And the HTTP response would be:

```
HTTP/1.1 201 Created
Date: Sat, 16 Apr 2022 17:42:18 GMT
Content-Length: 0
Connection: keep-alive
CF-Ray: 6fceb77bd9defd6e-ATH
Access-Control-Allow-Origin: *
Strict-Transport-Security: max-age=15552000; includeSubDomains
Via: 1.1 vegur
CF-Cache-Status: DYNAMIC
Access-Control-Allow-Methods: GET,HEAD,POST,OPTIONS
Content-Security-Policy:  upgrade-insecure-requests;  connect-src  'self'
wss:;
Cross-Origin-Embedder-Policy: require-corp
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Expect-Ct: max-age=0
Origin-Agent-Cluster: ?1
Permissions-Policy: microphone 'none'; geolocation 'none'
Referrer-Policy: strict-origin-when-cross-origin
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: DENY
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 1; mode=block
Set-Cookie:          __cf_bm=Ze2GXhR9F1kiF1WUnl5DvlVLvtnmRg6QOr1TgSnMmi0-
1650130938-0-
AfCJy/XvG01DOYK2+cuzlHiI5YwqVDo21urTD7bgDH7IlkZHVL4M9tmaCzuJQrCfyAB/JlG+P
```

```
nZB8/C/X/0bVu0=;    path=/;    expires=Sat,    16-Apr-22    18:12:18    GMT;
domain=.axelar.dev; HttpOnly; Secure; SameSite=None
Server: cloudflare
```

## Impact

Forged or otherwise, corrupted log files can be used to cover an attacker's tracks or even to implicate another party in the commission of a malicious act.

## Recommendation

It is advisable to restrict the access to the specific endpoint only to a list of whitelisted hosts.

In case that it is required to accept logs from external anonymous users, it is advisable to store them in a separate output file and perform strict validation on the reported events.

## CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.3.4 [Nest] Single standalone node of Redis in-use

| Description | LOW |
|---|---|

It was identified that Redis is configured to operate in a single, standalone node, introducing potential availability issues. Redis is a fast in-memory NoSQL database and cache. Currently, it is configured to be operating in a standalone node, which is in general not suitable for data reliability, there is no backup(slave) node for real-time data synchronization.

Furthermore, since Redis is a single-threaded mechanism, its performance is limited by the processing capacity of a single-core CPU, which restricts the number of instructions that can be performed on a single computer. On the contrary, the NodeJS instance is configured to operate in a cluster mode with two (2) instances.

The issue exists in the following location:

**File: axelar-bridge-nestjs-main/docker-compose.yml**

```
version: '3.9'
services:
  redis:
    image: redis:6
    ports:
      - 6379:6379
  redis-commander:
    container_name: redis-commander
    hostname: redis-commander
    image: rediscommander/redis-commander:latest
    environment:
    - REDIS_HOSTS=local:redis:6379
    ports:
    - 8081:8081
```

However, the nodejs is configured to operate in cluster mode with two (2) instances:

```
File axelar-bridge-nestjs-main/pm2.config.js
module.exports = {
  apps: [
    {
      name: 'bridge-api',
      script: 'dist/src/main.js',
      exec_mode: 'cluster',
      instances: 2,
    },
  ],
};

```

## Impact

An adversary can exploit this issue and attempt to exhaust the available resources by making constantly new requests that will add or search for a key in the Redis database, in order to exhaust the available resources. Currently, only a single instance of Redis is used, while there are two (2) available instances of NodeJS app, allowing easier exploitation of the issue. Redis server is a single-threaded system. Long running commands can cause latency or timeouts on the client side because the server can't respond to any other requests while it's busy working on a long running command.

Since this control is an added layer of security, which is also mitigated by Cloudflare in the production environment, the issue is marked as LOW.

## Recommendation

It is advisable to investigate if you can use Redis Sentinel or Redis Cluster. Both solutions will provide high availability to the system.

More information can be found in the following URLs:

- *Redis Sentinel: https://redis.io/docs/manual/sentinel/*
- *Redis Cluster: https://redis.io/docs/manual/scaling/*

Redis Sentinel is also supported in "redis-commander":

*https://github.com/joeferner/redis-commander*

Furthermore, it is advisable to add monitoring on server load to ensure that notifications would be sent when high server load occurs. Monitoring can help the organization understand the application constraints. Then, it would be possible to work proactively to mitigate issues. It is recommended to keep server load under 80% to avoid negative performance effects.

| **CVSS Score** |
|---|
| AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X |

### *5.3.5 [SDK] HTTPS not enforced in Rest Service*

**Description** **LOW**

It was identified that the SDK does not verify if the user-defined Rest URL is valid and is using the HTTPS protocol in production mode. In the examined case, the SDK is using the "fetch()" method of the "Fetch API" to perform the external request. The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network. However, the SDK does not validate if the URL is using the HTTPS protocol.

The issue exists in the following location:

```
File: axelarjs-sdk-0.5.0-alpha.11/src/services/RestService.ts
  private async execRest(endpoint: string, requestOptions: any) {
    return fetch(this.host + endpoint, requestOptions)
      .then((response) => {
        if (!response.ok) throw response;
        return response;
      })
      .then((response) => response.json())
      .catch(async (err) => {
        const _err = await err.json();
        throw {
          message: "AxelarJS-SDK uncaught post error",
          uncaught: true,
          fullMessage: _err.message,
        };
      });
  }
}
```

**Impact**

In case that the SDK is not configured to use the TLS/SSL protocol, the intermediate communication between the SDK and the backend Rest service will be susceptible to man-in-the-middle (MiTM) attacks. Adversaries located in the adjacent network (e.g., local network, Wi-Fi hotspot, ISP) would be able to compromise the communication channel and eavesdrop or tamper with the exchanged information.

**Recommendation**

The application configuration should ensure that TLS/SSL is used for all the controlled resources in production mode, and should make it impossible to access any of the resources without TLS/SSL.

It is advisable to verify if the user-provided Rest URL is valid and it is using the HTTPS protocol. For example, the following code can be used that works with Node's URL API:

```
function isValidHttpUrl(string) {
  let url;

  try {
    url = new URL(string);
  } catch (_) {
    return false;
  }

  return url.protocol === "https:";
}
```

This can be combined with manual checks, such as:

```
If (!url.startsWith('https://')){

}
```

| CVSS Score |
|---|
| AV:A/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N/E:P/RL:X/RC:R/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X |

### 5.3.6 [SDK] WSS not enforced in Web Socket Service

**Description** **LOW**

It was identified that the SDK does not verify if the user-defined "*resourceUrl*" is valid and is using the HTTPS protocol in production mode. The specific parameter is used to establish a WebSocket connection with the backend service. WebSocket is an HTML5 protocol that simplifies and speeds up communication between clients and servers. Once a connection is established through a handshake, messages can be passed back and forth while keeping the connection open. A WSS URI identifies a WebSocket server and resource name and indicates that traffic over that connection is to be protected via TLS (including standard benefits of TLS such as data confidentiality and integrity, and endpoint authentication).

The issue exists in the following location:

```
File: axelar/axelarjs-sdk-0.5.0-alpha.11/src/services/SocketService.ts
public async createSocket() {
  if (this.testMode) {
   ...
  } else {
    this.socket = io(this.resourceUrl, {
      transports: ["websocket"],
      reconnectionDelayMax: 10000,
      extraHeaders: {
        "User-Agent":
               "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0)
Gecko/20100101 Firefox/47.0",
      },
    });
  }
```

**Impact**

In case that the SDK is not configured to use the TLS/SSL protocol, the intermediate communication between the SDK and the backend Rest service will be susceptible to man-in-the-middle (MiTM) attacks. Adversaries located in the adjacent network (e.g., local network, Wi-Fi hotspot, ISP) would be able to compromise the communication channel and eavesdrop or tamper with the exchanged information.

## Recommendation

It is advisable to verify if the user-provided Rest URL is valid and it is using the HTTPS protocol. For example, the following code can be used that works with Node's URL API:

```
function isValidWSSUri(string) {
  let url;

  try {
    url = new URL(string);
  } catch (_) {
    return false;
  }

  return url.protocol === "wss:";
}
```

This can be combined with manual checks, such as:

```
If (!url.startsWith('https://')){

}
```

## CVSS Score

AV:A/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N/E:P/RL:X/RC:R/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### *5.3.7 [Nest] Logging levels are not changed in production mode*

**Description** **LOW**

It was identified that the application does not set different logging levels depending on the deployed environment. While logging all information may be helpful during development stages, it is important that logging levels be set appropriately before a product is deployed so that sensitive user data and system information are not accidentally exposed to potential attackers. Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information.

More precisely, the app is using the logger service of Nest. Nest comes with a built-in text-based logger which is used during application bootstrapping and several other circumstances such as displaying caught exceptions (i.e., system logging). This functionality is provided via the Logger class in the @nestjs/common package:

```
File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
import {
...
  Logger,
...
} from '@nestjs/common';

Then, the logger service is initialized with the default attributes:

File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
@Injectable()
export class BridgeService {
  environment: string;
  private readonly logger = new Logger(BridgeService.name);
  ...
```

**Impact**

In case that the application is deployed in the production environment, sensitive information might be disclosed to the default logging output.

Adversaries or malicious insiders that are able to gain access to this logging output, will be able to extract the data and infer the corresponding user actions.

The issue is marked as LOW as debug information is not necessarily sensitive.

**Recommendation**

According to the documentation, it is possible to disable the logging or change the logging levels in the following way in production mode:

```
const app = await NestFactory.create(AppModule, {
  logger: false,
});
await app.listen(3000);
```

or

```
const app = await NestFactory.create(AppModule, {
  logger: ['error', 'warn'],
});
await app.listen(3000);
```

For example:

```
const app = await NestFactory.create(AppModule, {
  logger:  process.env.NODE_ENV  ===  'development'  ?  ['log',  'debug',
'error', 'verbose', 'warn'] : ['error', 'warn'],
});
```

More information can be found in the following URL:

*https://docs.nestjs.com/techniques/logger*

While the implementation and ways to override the default filters can be found in the following URL:

*https://github.com/nestjs/nest/blob/master/packages/common/services/logger.service.ts*

| CVSS Score |
|---|
| AV:L/AC:L/PR:H/UI:N/S:U/C:L/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X |

### 5.3.8 [Web] Autocomplete is not disabled at password input

**Description** **LOW**

The team identified that the password input allows autocomplete. Modern browsers can remember user credentials that are entered into HTML forms. However, this functionality is configurable both by the end-users and by applications. If the function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application.

```
File: axelar-web-app-develop/src/hooks/usePasswordInput.tsx
  return [
    password,
    <div style={{ width: `50%`, position: `relative` }}>
      <InputForm
        name={"usePasswordInput"}
        placeholder={"Admin Password"}
        value={password}
        type={(passwordShown ? "text" : "password") as any}
        onChange={(e: any) => setPassword(e.target.value)}
        handleOnEnterPress={handleOnKeyPress}
      />
      <div
        style={{
          position: `absolute`,
          marginTop: `-32px`,
          right: `20px`,
          cursor: `pointer`,
        }}
      >
        <SVGImage
          src={require(`assets/svg/show-password.svg`)?.default}
          height={"30px"}
          width={"30px"}
          onClick={togglePassword}
        />
      </div>
      </div>,
  ] as const
}

```

## Impact

An adversary; who is able to gain access to the victim's browser (e.g., shared family laptop, cafe / library workstation), could steal the stored password and compromise the account.

## Recommendation

It is advisable to introduce the following attribute in the input tag:

```
autocomplete="off"
```

## CVSS Score

AV:L/AC:L/PR:H/UI:N/S:U/C:L/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.3.9 [Web] Hardcoded shortened URL in-use

**Description** **LOW**

It was identified that the application is using a hardcoded shortened URL to redirect users to a tutorial for the satellite app. URL shortening is a technique to make ordinary URLs substantially shorter and still direct to the required page. The security risk with a shortened URL is that it is impossible to tell where the redirection will occur. Users have to trust the sender of thatlink. As a result, some organizations guide their employees to avoid shortened URLs, or completely block them at their network gateway. In the specific case, the application has to trust the URL shortening service.

The issue exists at the following location:

```
File: axelar-web-app-develop/src/pages/app/parts/support-widget/QA.tsx
26:              <PopoutLink
27:              text={"Transfer Terra assets to EVM chains using Satellite"}
28:                  onClick={() => window.open("https://shorturl.at/dtDY8",
"_blank")}
29:              />
30:            </div>
```

**Impact**

In case that the HTTP redirection of the used shortened URL is changed, canceled or expired, this issue might introduce an Open-Redirect vulnerability. Currently, the HTTP redirection is controlled by a third-party vendor. Furthermore, it is possible that shortened URLs are blocked by some networks.

Since there is no known security incident with the third-party service that was used for generating this URL at the time of the assessment, this issue is marked as LOW

## Recommendation

It is recommended to use a stable, full, URL that is controlled by the organization.

## CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:L/E:U/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### *5.3.10 [Nest] OTC is stored unencrypted in Redis database*

**Description** **LOW**

It was found that the OTC codes are maintained in the Redis cache in plaintext. In general, Redis is an open-source, in-memory data store. However, Redis supports two (2) different modes of persistency: (a) the RDS in which it performs point-in-time snapshots of the dataset at specified intervals, and (b) the AOF in which it logs every write operation received by the server, and then it will replay them again at server startup, reconstructing the original dataset. The second mode is enabled by default. In both cases, it is possible that the data will be written to a durable storage, such as a solid-state drive (SSD).

The issue exists at the following location:

```
File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
36:   async getOneTimeCode(dto: GetOtcDto, traceId?: string): Promise<OTC>
{
37:     try {
38:       const newOTC = this.utils.getOTC();
39:       await this.cache.set<string>(dto.publicAddress, newOTC);
40:
41:       this.logger.debug({
42:         from: this.getOneTimeCode.name,
43:         traceId: traceId || 'none',
44:         newOTC,
45:         dto,
46:       });
47:
48:       return {
49:         validationMsg: this.utils.getOTCMessage(newOTC),
50:         otc: newOTC,
51:       };
52:     } catch (error) {
53:       this.logger.debug({
54:         from: this.getOneTimeCode.name,
55:         traceId: traceId || 'none',
56:         error,
          dto,
        });
        throw error;
      }
    }

```

## Impact

An adversary, who is able to gain access in the Redis persistent storage (e.g., a malicious insider, or malware in a compromised node in the same network), will be able to extract the mapping of the unused OTC and public addresses.

## Recommendation

For enhanced security, it is recommended to either encrypt the OTC before saving it to the cache, or disable Redis persistence.

For example, the following function can be used:

```
window.crypto.subtle.encrypt()
```

To disable Redis persistence, the following actions can be performed:

- Disable AOF by setting the *appendonly* configuration directive to no (It is disabled by default and can be skipped):

```
appendonly no
```

- Disable RDB snapshotting by commenting the save configuration directives and setting the empty string:

```
#save 900 1
#save 300 10
#save 60 10000
save ""
```

| CVSS Score |
|---|
| AV:L/AC:H/PR:H/UI:N/S:U/C:L/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X |

## 5.4 Informational Findings

### *5.4.1 [Nest] Spoofable client IP address due to custom decorator*

**Description** **INFO**

It was identified that a custom decorator of the Nest framework is using a spoofable parameter to track the IP address of the connected user. In general, Nest provides a set of param decorators that can be used together with the HTTP route handlers. Additionally, developers are able to create their own custom decorators, such as the "GetRealIp". In the specific case, the "x-forwarded-for" header is preferred instead of the connection IP address. If an application trusts an HTTP request header like X-Forwarded-For to accurately specify the remote IP address of the connecting client, then malicious clients can spoof their IP address. This behavior does not necessarily constitute a security vulnerability; however, some applications use client IP addresses to enforce access controls and rate limits.

The issue exists at the following location:

**File: axelar-bridge-nestjs-main/src/bridge/decorator/get-real-ip.decorator.ts**

```
import { createParamDecorator, ExecutionContext } from '@nestjs/common';
import { Request } from 'express';

export const GetRealIp = createParamDecorator(
  (data: undefined, ctx: ExecutionContext) => {
    const request = ctx.switchToHttp().getRequest<Request>();
    const forwardedIp = request.headers['x-forwarded-for'];

    return forwardedIp || request?.socket?.remoteAddress || request.ip;
  },
);

```

The application will use either the "forwardedIp" (if it exists), or the connection IP address.

## Impact

HTTP request headers such as X-Forwarded-For, True-Client-IP, CF-Connecting-IP, and X-Real-IP cannot be trusted.

In the examined case, no usage of the affected @GetRealIP decorator was found, and the issue is marked as INFORMATIONAL

## Recommendation

In case that the client's IP address is used for security controls (e.g., rate limiting), it is advisable to either use the client connection's IP address or to use a list of client connection's IP address and the headers' addresses (e.g., CF-Connecting-IP, X-Forwarded-For)

Furthermore, if a third-party provider is used as a stacked CDN or load balancer, it is advisable to verify that the correct configuration is enabled to mitigate spoofed headers. For example, in Cloudflare the "Enable True-Client-IP Header" must be enabled as shown in the following URL:

> *https://support.cloudflare.com/hc/en-us/articles/206776727#h_4bf7CC7xR9dZJjR4y6wwcG*

## CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MA
C:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.4.2 [Nest] Multiple logging mechanisms are used

**Description** **INFO**

It was identified that the application is using multiple logging mechanisms. Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging.

Initially, the bridge service is using the logger service of Nest. Nest comes with a built-in text-based logger which is used during application bootstrapping and several other circumstances such as displaying caught exceptions (i.e., system logging). This functionality is provided via the Logger class in the @nestjs/common package:

```
File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
import {
...
  Logger,
...
} from '@nestjs/common';
```

Then, the logger service is initialized with the default attributes:

```
File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
@Injectable()
export class BridgeService {
  environment: string;
  private readonly logger = new Logger(BridgeService.name);
  ...
```

However, there is a second logging mechanism that is used to collect events remotely:

```
File: axelar-bridge-nestjs-main/src/log/log.controller.ts
```

```
import { Controller, Post, Headers, Body } from '@nestjs/common';
import { LogPayload } from './dto';
import { LogService } from './log.service';

@Controller('logMessageController')
export class LogController {
  constructor(private logService: LogService) {}

  @Post()
  log(@Headers('x-traceid') traceId: string, @Body() dto: LogPayload) {
    return this.logService.log(dto, traceId);
  }
}

```

The the LogService is defined at the following location:

```
File: axelar-bridge-nestjs-main/src/log/log.service.ts
import { Injectable } from '@nestjs/common';
import { LogPayload } from './dto';

@Injectable()
export class LogService {
  log(dto: LogPayload, traceId: string) {
    console.log({
      timestamp: new Date(),
      message: dto.message.replace(/(\t)\s+/g, '$1'),
      trace_id: traceId,
      level: dto.messageType,
    });
  }
}

```

As a result, a simple "console.log" function is used in this case. Any changes in the first logging mechanism (e.g., log levels, masking, output) will not affect this logging mechanism.

## Impact

Having multiple logging mechanisms in the same application is a bad practice, since any changes in the one logging mechanism (e.g., log levels, masking, output) will not affect the others.

However, this is not considered a security vulnerability and is marked as INFORMATIONAL.

## Recommendation

It is advisable to follow a common logging format and approach within the system and across systems of an organization.

## CVSS Score

AV:L/AC:L/PR:H/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.4.3 *[Nest] Excessive signature verification attempts are allowed per wallet address*

#### Description

**INFO**

The team identified that the application does not limit the maximum amount of signature verification attempts that can be performed for a generated OTC code, allowing adversaries to conduct brute force attacks. A brute-force attack is an attempt to discover a secret by systematically trying every possible combination of letters, numbers, and symbols until the one correct combination that works is discovered.

The issue exists in the following location:

```
File: axelar-bridge-nestjs-main/src/bridge/service/bridge.service.ts
  verifySignature(otc: string, sig: string, address: string) {
    const sigMatches = this.utils.verifySignature(otc, sig, address);
    if (!sigMatches)
      throw new ForbiddenException(
          `Provided signature for public address: ${address} is not valid`,
      );
  }
```

#### Impact

An adversary, who has access to the wallet, can conduct a brute force attack against the specific endpoint.

Since the OTC code is mainly used to avoid double transactions, the cache TTL for redis store is set to 5 minutes, and a per-IP basic throttling control is already integrated for the APIs, the issue is marked as INFORMATIONAL.

#### Recommendation

It is advisable to introduce anti-automation security controls (e.g., CAPTCHA) and lockdown policies (e.g., throttling per-address) for addresses with multiple failed verification attempts.

For example, it is recommended to limit the number of verification attempts by implementing a throttling mechanism for the affected endpoint or a lock mode for a specific amount of time, when numerous failed attempts are detected.

| CVSS Score |
|---|
| AV:N/AC:H/PR:H/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X |

### 5.4.4 [Nest] HTTP Strict Transport Security (HSTS) not used

**Description** **INFO**

It was identified that the application is not using the HTTP Strict Transport Security (HSTS) header. The HSTS is an opt-in security enhancement that is specified by a web application through the use of a special response header, and instructs the browser to prevent any communications from being sent over HTTP to the specified domain.

As of September 2019, HSTS is supported by all modern browsers, with the only notable exception being Opera Mini.

**Impact**

The application fails to prevent users from connecting to it over unencrypted connections. An adversary who is suitably positioned in the adjacent network will be able to intercept and modify the victim's network traffic.

It should be noted that in the staging environment this issue is fixed (Maybe in the Cloudflare layer). As a result, it is marked as INFORMATIONAL. For example, the following HTTP GET request can be used:

```
GET /otc?publicAddress=0xB283a80A211c0e9fE151cE3E4b8996bc89fe884c
HTTP/1.1
Host: nest-server-testnet.axelar.dev
Cookie: _ga=GA1.2.1033642504.1650041394; _gid=GA1.2.421275231.1650041394;
__cf_bm=D9sUDUIxoKV5PpJZZ5geP4YuNvSk2C_Mp2KZxPOIVKw-1650043353-0-
ARZ+ppqbzmpqbRq47FlqkgsYD9Eaz6mzP/bWTkgiJ2sbd6Nz12K+tNL45ff5yKbFc6yXqxey2
8p6o4WMedekFMFTjh8t7/7QBZeKDRknS03FvK7n2/Vak2YBjl2RXmsKDzDSfkea1Q3mZAls9C
v4jVXJlhgfYSvxUmDy6xDzHjKo
Cache-Control: max-age=0
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="100", "Google
Chrome";v="100"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "macOS"
Dnt: 1
Upgrade-Insecure-Requests: 1
```

```
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.88 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
bp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: el-GR,el;q=0.9,en-US;q=0.8,en-GB;q=0.7,en;q=0.6
```

And the HTTP response would be:

```
HTTP/1.1 200 OK
Date: Fri, 15 Apr 2022 17:23:39 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
CF-Ray: 6fc65ec939e038d5-ATH
Access-Control-Allow-Origin: *
ETag: W/"7f-QErqx0SfvxNYCxnPFiaeEkPvzRc"
Strict-Transport-Security: max-age=15552000; includeSubDomains
Via: 1.1 vegur
CF-Cache-Status: DYNAMIC
Content-Security-Policy: default-src 'self';base-uri 'self';block-all-
mixed-content;font-src 'self' https: data:;form-action 'self';frame-
ancestors 'self';img-src 'self' data:;object-src 'none';script-src
'self';script-src-attr 'none';style-src 'self' https: 'unsafe-
inline';upgrade-insecure-requests
Cross-Origin-Embedder-Policy: require-corp
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Expect-Ct: max-age=0
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
```

```
X-Permitted-Cross-Domain-Policies: none
X-Ratelimit-Limit: 10
X-Ratelimit-Remaining: 9
X-Ratelimit-Reset: 0
X-Xss-Protection: 0
Vary: Accept-Encoding
Server: cloudflare
Content-Encoding: br
Content-Length: 117
....
```

## Recommendation

It is recommended to enable the HSTS header. The specification for the header has been released and published at the end of 2012 as RFC 6797 (HTTP Strict Transport Security (HSTS)) by the IETF. For example, to specify the header with a duration of one year the following should be set:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Since the app is using the Helmet ExpressJS middleware, it is possible to enable the header with the following function wrapper:

```
app.use(helmet.hsts());
```

## CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.4.5 [Web] HTTP Strict Transport Security (HSTS) not used

**Description** **INFO**

It was identified that the application is not using the HTTP Strict Transport Security (HSTS) header. The HSTS is an opt-in security enhancement that is specified by a web application through the use of a special response header and instructs the browser to prevent any communications from being sent over HTTP to the specified domain.

As of September 2019, HSTS is supported by all modern browsers, with the only notable exception being Opera Mini.

**Impact**

The application fails to prevent users from connecting to it over unencrypted connections. An adversary who is suitably positioned in the adjacent network will be able to intercept and modify the victim's network traffic.

It should be noted that in the staging environment this issue is fixed (Maybe in Cloudflare layer). As a result, it is marked as INFORMATIONAL. For example, the following HTTP GET request can be used:

```
GET /?source=axelar&token=axl&destination=avalanche HTTP/1.1
Host: bridge.testnet.axelar.dev
```

And the HTTP response would be:

```
HTTP/1.1 200 OK
Date: Sun, 17 Apr 2022 19:30:04 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
cache-control: public, max-age=0, must-revalidate
etag:
W/"da92de6b2d6f1fb55ea3f04e94e2059dbea2081bcd1118c4557c5c8123f7ddf1"
access-control-allow-origin: *
content-disposition: inline; filename="index.html"
Age: 394370
```

```
X-Vercel-Cache: HIT
server: Vercel
x-vercel-id: fra1:fra1::wjsfl-1650223804697-de41e3322102
strict-transport-security: max-age=63072000
Content-Encoding: br
Content-Length: 1445
....
```

## Recommendation

It is recommended to enable the HSTS header. The specification for the header has been released and published at the end of 2012 as RFC 6797 (HTTP Strict Transport Security (HSTS)) by the IETF. For example, to specify the header with a duration of one year the following should be set:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Since the app is using the Helmet ExpressJS middleware, it is possible to enable the header with the following function wrapper:

```
app.use(helmet.hsts());
```

## CVSS Score

AV:A/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.4.6 [Web] Frame attacks (Click-Jacking) are not prevented

**Description** **INFO**

In click-jacking, an adversary uses multiple transparent or opaque layers (frames) to trick a victim into clicking on a component (e.g., a button or a link) of the targeted legitimate page without knowing it, and as a result, forcing the victim to perform an unwanted action. The victim would think that is clicking on an element on the top level of the page, while the reality is that the click action is routed at the lower frame of the page (click "hijacking").

The issue occurs because the page does not set the appropriate X-Frame-Options or Content-Security-Policy HTTP header, or it does not validate if it is loaded within an iframe.

**Impact**

An adversary can exploit this issue to trick victims to perform unwanted actions. In the specific case, the victims can be tricked to visit the site and initiate a transaction, generating in this way a wallet that will be stored in the *localbrowser*.

It should be noted that in the staging environment this issue is fixed (Maybe in the Cloudflare layer). As a result, it is marked as INFORMATIONAL. For example, the following HTTP GET request can be used:

```
GET /?source=axelar&token=axl&destination=avalanche HTTP/1.1
Host: bridge.testnet.axelar.dev
```

And the HTTP response would be:

```
HTTP/1.1 200 OK
Date: Sun, 17 Apr 2022 19:30:04 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
cache-control: public, max-age=0, must-revalidate
etag:
W/"da92de6b2d6f1fb55ea3f04e94e2059dbea2081bcd1118c4557c5c8123f7ddf1"
```

```
access-control-allow-origin: *
content-disposition: inline; filename="index.html"
Age: 394370
X-Vercel-Cache: HIT
server: Vercel
x-vercel-id: fra1:fra1::wjsfl-1650223804697-de41e3322102
strict-transport-security: max-age=63072000
Content-Encoding: br
Content-Length: 1445
....
```

## Recommendation

In general, there are several recommended ways to prevent a click-jacking attack:

- First, it is recommended to prevent the browser from loading the page in frame using the X-Frame-Options or Content Security Policy (frame-ancestors) HTTP headers.
- Next, it is advisable to prevent session cookies from being included when the page is loaded in a frame using the Same Site cookie attribute. However, this is only effective if the hijacked action requires an authenticated session.
- Finally, another solution is to implement JavaScript code in the page to attempt to prevent it being loaded in a frame (known as a "frame-buster").

Since the app is using the Helmet ExpressJS middleware, it is possible to enable the header with the following function wrapper:

```
app.use(helmet.frameguard());
```

## CVSS Score

AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.4.7 *[Nest] MIME sniffing is allowed*

**Description** **INFO**

It was identified that the web application allows MIME sniffing. More precisely, it was found that the X-Content-Type-Options response HTTP header which is normally used by the server to prevent browsers from guessing the media type (MIME type), is not configured in the code of the web application. This issue is known as MIME sniffing in which the browser guesses the correct MIME type by looking at the contents of the resource. The absence of this header might cause browsers to transform non-executable JavaScript content (such as JSON payloads) into executable content.

In the examined app, there are many cases in which an arbitrary user-provided input is reflected in the HTTP response, and protected from execution due to the existence of a correct content type. For example:

```
File:axelar-bridge-nestjs-main/src/bridge/decorator/is-valid-asset.ts
@ValidatorConstraint({ name: 'isValidAsset', async: false })
export class IsValidAsset implements ValidatorConstraintInterface {
  validate(asset: string) {
    return assetNames.includes(asset);
  }

  defaultMessage(args: ValidationArguments) {
    return `Incorrect asset: ${args.value}`;
  }
}
```

If an adversary performs the following HTTP request:

```
GET /chain/fee?chainName=Axelar&assetCommonKey=<img+src=> HTTP/1.1
Host: nest-server-testnet.axelar.dev
Content-Length: 0
X-Trace-Id: bzxczx
```

The response would be:

```
HTTP/1.1 400 Bad Request
Date: Mon, 18 Apr 2022 18:59:36 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 82
Connection: keep-alive
CF-Ray: 6fdfa3785bba6f4d-ATH
Access-Control-Allow-Origin: *
ETag: W/"52-dSI5NAkbkULOUloKXU7WQNDU+7c"
Strict-Transport-Security: max-age=15552000; includeSubDomains
Via: 1.1 vegur
CF-Cache-Status: DYNAMIC
Content-Security-Policy: default-src 'self';base-uri 'self';block-all-
mixed-content;font-src 'self' https: data:;form-action 'self';frame-
ancestors 'self';img-src 'self' data:;object-src 'none';script-src
'self';script-src-attr 'none';style-src 'self' https: 'unsafe-
inline';upgrade-insecure-requests
Cross-Origin-Embedder-Policy: require-corp
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Expect-Ct: max-age=0
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 0
Set-Cookie: __cf_bm=seBBT3O2lO_OZvX9bE1BbttOb_aSIyKCG0RsbvSHJ.s-
1650308376-0-
AYJnkzyuZTuy6qVDwW/WvXxQWskWu7+2EtFPffMGS1DVYRLwnJ7B2VVL9TvKyxusroa0AXMab
qWTKyxO3vlQdlk=; path=/; expires=Mon, 18-Apr-22 19:29:36 GMT;
domain=.axelar.dev; HttpOnly; Secure; SameSite=None
Server: cloudflare
```

```
{"statusCode":400,"message":["Incorrect asset: <img src=>"],"error":"Bad
Request"}
```

## Impact

When MIME sniffing is not disabled, adversaries are able to exploit user-provided reflected values to perform Cross Site Scripting attacks (XSS).

It should be noted that in the staging environment this issue is fixed (Maybe in the Cloudflare layer). As a result, it is marked as INFORMATIONAL.

For example, the following HTTP GET request can be used:

```
GET /otc?publicAddress=0xB283a80A211c0e9fE151cE3E4b8996bc89fe884c
HTTP/1.1
Host: nest-server-testnet.axelar.dev
Cookie: _ga=GA1.2.1033642504.1650041394; _gid=GA1.2.421275231.1650041394;
__cf_bm=D9sUDUIxoKV5PpJZZ5geP4YuNvSk2C_Mp2KZxPOIVKw-1650043353-0-
ARZ+ppqbzmpqbRq47FlqkgsYD9Eaz6mzP/bWTkgiJ2sbd6Nz12K+tNL45ff5yKbFc6yXqxey2
8p6o4WMedekFMFTjh8t7/7QBZeKDRknS03FvK7n2/Vak2YBjl2RXmsKDzDSfkea1Q3mZAls9C
v4jVXJlhgfYSvxUmDy6xDzHjKo
Cache-Control: max-age=0
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="100", "Google
Chrome";v="100"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "macOS"
Dnt: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.88 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
bp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
```

```
Accept-Encoding: gzip, deflate, br
Accept-Language: el-GR,el;q=0.9,en-US;q=0.8,en-GB;q=0.7,en;q=0.6
```

And the HTTP response would be:

```
HTTP/1.1 200 OK
Date: Fri, 15 Apr 2022 17:23:39 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
CF-Ray: 6fc65ec939e038d5-ATH
Access-Control-Allow-Origin: *
ETag: W/"7f-QErqx0SfvxNYCxnPFiaeEkPvzRc"
Strict-Transport-Security: max-age=15552000; includeSubDomains
Via: 1.1 vegur
CF-Cache-Status: DYNAMIC
Content-Security-Policy: default-src 'self';base-uri 'self';block-all-
mixed-content;font-src 'self' https: data:;form-action 'self';frame-
ancestors 'self';img-src 'self' data:;object-src 'none';script-src
'self';script-src-attr 'none';style-src 'self' https: 'unsafe-
inline';upgrade-insecure-requests
Cross-Origin-Embedder-Policy: require-corp
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Expect-Ct: max-age=0
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Ratelimit-Limit: 10
X-Ratelimit-Remaining: 9
X-Ratelimit-Reset: 0
X-Xss-Protection: 0
Vary: Accept-Encoding
Server: cloudflare
Content-Encoding: br
Content-Length: 117
```

```
....
```

## Recommendation

Since the app is using the Helmet ExpressJS middleware, it is possible to enable the header with the following function wrapper:

```
app.use(helmet.noSniff());
```

## CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

### 5.4.8 [Web] Debug routes in production

**Description** **INFO**

It was identified that the application is deployed with debugging code still enabled or active, which can create unintended entry points or expose sensitive information. More precisely, it was found that a debug route is left enabled at the production deployment. The routes are defined using React router, a standard library for routing in React which allows the navigation among views of various components in the application changing the browser URL and keeping the UI in sync with the URL. Allowing debug routes increases the overall attack surface of the application.

The issue exists in the following location:

**File: axelar-web-app-develop/src/routes.tsx**

```
    <Switch>
      <ProtectedRoute exact path="/" component={AppPage} />
      <ProtectedRoute exact path="/app" component={AppPage} />
      <Route exact path="/landing" component={LandingPage} />
      <ProtectedRoute exact path="/debug" component={DebugPage} />
    </Switch>
```

The "ProtectedRoute" normally limits the access to the specific routes only to authenticated users:

**File: axelar-web-app-develop/src/components/CompositeComponents/ProtectedRoute.tsx**

```
// @ts-ignore
function ProtectedRoute({ component: Component, ...restOfProps }) {
  const isAuthenticated = useRecoilValue(IsLoggedInWithBetaPassword)

  return (
    <Route
      {...restOfProps}
      render={(props) =>
```

```
        isAuthenticated ? <Component {...props} /> : <Redirect to="/landing" />
      }
    />
  )
}

export default ProtectedRoute

```

However, the default value is set to true and all users can access them:

```
File: axelar/axelar-web-app-develop/src/state/ApplicationStatus.ts
export const IsLoggedInWithBetaPassword = atom<boolean>({
  key: "IsLoggedInWithBetaPassword",
  default: true,
})

```

## Impact

Debug entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the application.

In the duration of this assessment, no vulnerability was identified in the exposed routes and the relevant debug interfaces. As a result, the issue is marked as INFORMATIONAL.

## Recommendation

It is advisable to restrict the access to the debug interfaces and routes in the production deployment

| CVSS Score |
|---|
| AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X |

### 5.4.9 [Web] Third-party log service "datadoghq" in use

**Description** **INFO**

The team identified that a third-party logging service is integrated in the application, allowing user's related data to be transmitted to third-party services. Datadog is a monitoring and analytics platform for cloud environments.

**File: axelar-web-app-develop/src/index.tsx**
```
datadogLogs.init({
  clientToken: process.env.REACT_APP_DD_CLIENT_TOKEN as string,
  datacenter: "us",
  site: "datadoghq.com",
  forwardErrorsToLogs: true,
  sampleRate: 1,
  service: `satellite_browser_${process.env.REACT_APP_STAGE}`,
  env: process.env.REACT_APP_STAGE,
  beforeSend: (log) => {
    if (log.http && log.http.status_code === 404) {
      return false
    }
  },
})
```

**File: axelar-web-app-develop/src/hooks/usePostTransactionToBridge.tsx**
```
        const log = {
          sourceChain: sourceChain?.chainName,
          destinationChain: destinationChain?.chainName,
          destinationAddress,
          asset: sourceAsset?.common_key,
          traceId: _traceId,
        }
        datadogLogs.logger.info("LINK_EVENT", log)
```

## Impact

In the examined case, no sensitive data was identified in the transmitted payload, and as a result the issue is marked as INFORMATIONAL.

## Recommendation

It is advisable to ensure that any data that can lead to user identification or tracking are masked before send to a third-party service (e.g., trace identifier). Also, it is recommended to verify that the third-party service is following the best-practices for transferring and storing the data.

Furthermore, it is recommended to evaluate if the selected "datacenter" and "site" parameters are the correct ones and compliant with the data protection laws and regulations for the region of the visitors. For example, if sensitive information is stored for EU users, a server in Europe might be required to be selected https://www.datadoghq.com/about/latest-news/press-releases/eu-region-germany/

## CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

## References & Applicable Documents

| Ref. | Title | Version |
|---|---|---|
| *N/A* | *N/A* | *N/A* |

## Document History

| Revision | Description | Changes Made By | Date |
|---|---|---|---|
| *0.1* | *Initial draft* | *Chaintroopers* | *April 22nd, 2022* |
| *1.0* | *Final version* | *Chaintroopers* | *April 25th, 2022* |