SECURITY SPECIFICATION AND IMPLEMENTATION

Cognitive Assistant for Networks (CAN) Release 6.0

JANUARY 23, 2023

AVANSEUS TECHNOLOGIES PVT. LTD.

Revision history

Version	Date	Change description	Created by	Updated by	Reviewed by
V 1.0	Jan, 2023	Initial Release	Naveen/Umesh	Raksha	Chiranjib

Table of Contents

1. Objectives

This document focuses on specifying web application security requirements and its configuration to implement these requirements. These configurations allow the application to behave as expected, even when there is an attack on the application. The idea is to provide a set of security controls engineered into the web application to protect its assets from potentially malicious agents. The document covers handling of OWASP top 10 vulnerabilities as well as handling other major vulnerabilities.

The topics covered in this document are as follows:

1. URL specification
2. Data validation
3. Session management
4. Cross site scripting
5. SQL injection
6. Information leakage
7. Default pages
8. HTTP methods
9. Human confirmation
10. Cross site request forgery
11. HTTP strict transport security
12. Cookie configuration
13. SSL protocol – Transport
14. Cipher suite
15. Hashing
16. Key exchange protocol
17. SSL/TLS channel configuration
18. Hiding server details
19. Anti-clickjacking
20. Prevent directory listing
21. Limit request body size
22. Content security policy

Each section is divided into 2 sub-divisions:

a. **Requirements**: List of sub-requirements are mentioned here.
b. **Implementation**: Methodology or technique used to configure the system to prevent that vulnerability.

2. URL Specification

Requirements:

- URLs shall not contain sensitive information. Sensitive information includes passwords, IP addresses etc.
- URLs shall not show data in clear text.
- Transfer of sensitive data shall always be performed via HTTP POST and not via HTTP GET.
- Systems should ensure that no sensitive information is transferred during a redirection.

Implementation:

- All HTTP requests would be performed via HTTP POST apart from landing screens which do not need query string parameters to load the screen.

3. Data Validation

Requirements:

User input must be rejected where one or more of the following are true:

- Data is not formatted as expected.
- Incorrect syntax.
- Contains parameters or characters with invalid values.
- Contains a numeric value that would cause calculation in the application to divide a number by zero.
- Contains parameters when the source cannot be validated by the user's session.

Implementation:

- Every data submitted in the UI, goes through a client side validation keeping in view that all the above mentioned issues are checked.
- However, when client side validation is bypassed, a standard server side validation framework is put in place to validate all the above mentioned issues on the user submitted data.

4. Session Management

Requirements:

The web application shall have a strong and consistent framework for session ID. management. Creation and deletion must be protected throughout their life cycle based on following measures:

- The session ID shall not be included in the URL.
- The session ID shall be built with high complexity so that it cannot be easily guessed.
- Session ID based on source IP or personal information shall not be used.
- Numerically incremental session IDs shall not be used.

- Active sessions shall be controlled to avoid multiple instances of the application with the same session ID.
- Session IDs shall expire after predefined inactivity time and when the user logs out.
- Session IDs shall not be reused.
- Users shall not be allowed to choose or change session IDs.

Implementation:

- Session ID generation is done by Apache tomcat web server using Pseudo random technique in Java.
- Session ID will not be included in the URL when following is configured:
  - ➢ Go to tomcat folder.
  - ➢ Open conf folder under tomcat (tomcat>>conf)
  - ➢ Open web.xml inside the conf folder (tomcat>>conf>>web.xml)
  - ➢ Under the session-config section add the below lines:

```
<session-config>
        <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

- Session ID name change configuration is as follows:
  - ➢ Go to tomcat folder.
  - ➢ Open conf folder under tomcat (tomcat>>conf)
  - ➢ Open web.xml inside the conf folder (tomcat>>conf>>web.xml)
  - ➢ Under the session-config section add the below lines:

```
<session-config>
        <cookie-config>
              <name>id</name>
        </cookie-config>
</session-config>
```

5. Cross Site Scripting

Requirements:

- The web application shall validate all headers, cookies, query strings, form fields and hidden fields in order to prevent cross-site scripting (XSS).

Implementation:

- A standard XSS filter is in place on the server side which would remove all HTML & Script tags from the request data to make sure that Stored XSS is prevented.
- To sanitize the data, OWASP's XSS library is used.
- As far as developer work is concerned, measures need to be taken by following Secure coding practices document to prevent XSS.

6. SQL Injection

Requirements:

- Insertion of SQL into input data from the client to the application shall be controlled in order to avoid SQL injection attacks.

Implementation:

- Application is not susceptible to SQL injection as SQL relational database is not used for storing data.
- MongoDB is being used and by default Mongo Query injection is not possible unless the “$where” clause is used. It is made sure that the “$where” clause is never used in the application.

7. Information Leakage

Requirements:

The web application shall not disclose any kind of information that can lead to information leakage, and shall ensure that:

- There is no visible or user downloadable files containing information about the application (Eg: Admin manuals).
- Code presented to the user does not contain sensitive information about the application (Eg: References to databases, passwords, user IDs, application structure, programmer comments etc.,)
- Information sent by the application is limited to minimum.
- Identification of the web server type and version shall be removed.

Implementation:

- Web applications don't have any downloadable files containing information about applications.
- Code presented to the user does not contain sensitive information & if the user changes the code from UI, then the Java security manager makes sure that sensitive information access, remote connection is blocked.
- Web Server version and details will not be shown in the UI and also in the response headers too.

8. Default Pages

Requirements:

- Default pages from the web server and from the web application shall be disabled and no unnecessary details shall be shown. A generic error page shall be used instead.

Implementation:

- All default error pages in the web server are modified to show generic error messages.

- Web application error page will be a customized page.

9. HTTP Methods

Requirements:

- All unnecessary HTTP methods (Eg: PUT, DELETE, TRACE, OPTIONS) and WEBDAV methods (Eg: MOVE, PROPFND) shall be disabled on the web application servers.

Implementation:

- Only allowed HTTP methods are GET and POST. Rest all are disabled in the web server.

Method 1: Write this inside the server section of the nginx.conf:

```
if ($request_method !~ ^(GET|POST)$ ) {
    return 403;
 }
```

Method 2 (**Preferred Method**): Write this inside the server section of the nginx.conf

```
location / {
limit_except GET POST { deny all; }
}
```

10. Human Confirmation

Requirements:

- Operations requiring human confirmation (Eg: Password change) shall implement controls to prevent automatic operations that attackers could perform.

Implementation:

- Login happens via Two-factor authentication method. User enters their username and password, then he will be prompted to put an OTP to login.
- Forgot password feature goes through a flow where the user will be sent an auto generated password to his Email id. He needs to login with that and would be mandatorily asked to change his password.
- Change/Reset password feature goes through a flow where the user needs to confirm his old password and enter new password twice.

11. Cross Site Request Forgery (CSRF or XSRF)

Requirements:

- The web application shall have provisions against Cross-Site Request Forgery attacks.

Implementation:

- Web application is built with a strong Anti-CSRF framework using the Double submit cookie design. This is a stateless methodology to prevent CSRF attacks.

12. HTTP Strict Transport Security (HSTS)

Requirements:

- The web application shall implement HTTP Strict Transport Security (HSTS) in critical sections or when transmitting critical data. In other scenarios the use of SSL plus user authentication is sufficient.

Implementation:

- HSTS: HTTP Strict Transport Security (HSTS) is a web server directive that informs user agents and web browsers how to handle its connection through a response header sent at the very beginning and back to the browser. It forces all the connections to happen over https instead of http.
- HTTP Strict Transport Security (HSTS) is a web security policy mechanism that helps to protect websites against protocol downgrade attacks and cookie hijacking.
  - ➢ Your website must have an SSL Certificate.
  - ➢ Redirect ALL HTTP links to HTTPS with a 301 Permanent Redirect. This configuration can be made in the nginx.conf file under /etc/nginx . Add the following lines under the server section of the nginx.conf file.

    ```
    if ($host = <DOMAIN_NAME>) {
         return 301 https://$host$request_uri;
      }
    ```

    DOMAIN NAME should be replaced as applicable.
  - ➢ All subdomains must be covered in your SSL Certificate.
  - ➢ Serve an HSTS header on the base domain for HTTPS requests.
  - ➢ Max-age must be at least 10886400 seconds or 18 Weeks.
- Configuration in Nginx, write this inside the server section of the nginx.conf:

  ```
  add_header Strict-Transport-Security "max-age=31536000; includeSubdomains; preload";
  ```

13. Cookie Configuration

Requirements:

- Httponly flag:

  Purpose: An 'httpOnly cookie' cannot be accessed by the client side API such as JavaScript's. This restriction eliminates the threat of cross site scripting (XSS) attack.
- Secure flag:

  Purpose: A 'secure cookie' can only be transmitted over secure channel i,e https. This makes the cookie less likely to be exposed to the attacker.
- Path attribute:

Purpose: The ‘path’ attribute signifies the URL or the path to which the cookie is valid. The default path is set to ‘/’.

- Domain attribute:

  Purpose: The ‘domain’ attribute specifies the domain to which the cookies are valid. If the attribute is not specified, then the host name of the originating server is used as a default value.

Implementation:

- *Session cookie configuration in Tomcat*
  - ➢ Open web.xml inside the conf folder (tomcat>conf>web.xml)
  - ➢ Under the session-config section add the below lines:

```
<cookie-config>
        <http-only>true</http-only>
        <secure>true</secure>
        <path></application_name/></path>
        <domain>
                <ip_address_of_the_server/domain_name>
        </domain>
</cookie-config>
```

- *Other cookies configuration in Nginx web server*

  Method 1:

```
server {
proxy_cookie_domain  localhost <ip_address_of_the_server/domain_name>;
proxy_cookie_path / "/; secure; HttpOnly; SameSite=strict";
}
```

  Method 2 (**Preferred Method**):

```
add_header Set-Cookie
"Path=/;Domain=avanseuscanvm.com;HttpOnly;Secure;SameSite=strict";
```

14. SSL Protocol - Transport

Requirements:

- Generally, use TLS-1.2 or higher.
- Disable TLS-1.0, 1.1, SSLv3 and lower.

Implementation

- Open the file nginx.conf
- Edit the SSLProtocol as below

```
ssl_protocols TLSv1.2;
```

15. Cipher Suite

Requirements:

- Use compatible cipher suites with TLS-1.2.
- Avoid using CBC suites which can prevent attackers from using BEAST attacks.

Implementation

- Open the file nginx.conf
- Edit the SSLCipherSuite as below

  ssl_prefer_server_ciphers on;
  ssl_ciphers "ALL:!RSA:!CAMELLIA:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!RC4:!SHA1:!SHA256:!SHA384";

16. Hashing

Requirements:

- Use SHA-2 algorithms.
- Disable all other hashing algorithms (Eg: SHA-1, MD5 etc.,).

Implementation

- Open the file nginx.conf
- Edit the SSLCipherSuite as below

  ssl_prefer_server_ciphers on;

  ssl_ciphers "ALL:!RSA:!CAMELLIA:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!RC4:!SHA1:!SHA256:!SHA384";

17. Key Exchange Protocol

Requirements:

- Generally, use ephemeral DH key exchanges (DH & ECDH).
- Disable other key exchanges.

Implementation:

- Open the file nginx.conf
- Edit the SSLCipherSuite as below

  ssl_prefer_server_cipherexcluding local interfaces such as the docker bridges on;

  ssl_ciphers "ALL:!RSA:!CAMELLIA:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!RC4:!SHA1:!SHA256:!SHA384";

18. SSL/TLS Channel Configuration

Requirements:

- Disable compression
- Enable secure renegotiation
- Disable client initiated renegotiation
- Enable session resumption

Implementation:

- Disable Compression: By default, in the Nginx server compression is disabled.
- Enable secure renegotiation: By default, in the Nginx server secure renegotiation is enabled.
- Disable client initiated renegotiation: By default, in the Nginx server client initiated renegotiation is disabled.
- Enable session resumption: Nginx server maintains a cache to implement session resumption. Following configuration is needed in Nginx server nginx.conf file:

```
ssl_session_cache shared:SSL:50m;
ssl_session_timeout 5m;
```

19. Hiding Server Details

Requirements:

- Server Details must be hidden from the Request and Response header.

Implementation:

- Configuration: Nginx Level
  - ➢ Go to /etc/nginx/ folder.
  - ➢ Open nginx.conf file using vi editor.
  - ➢ Add the below under the server {} section

```
server_tokens off;
```

20. Anti-Clickjacking

Requirements:

- The X-Frame-Options HTTP response **header** can be used to indicate whether or not a browser should be allowed to render our application pages in a <frame>, <iframe>, <embed> or <object> to avoid Clickjacking.

Implementation:

- Configuration: Nginx Level
  - ➢ Go to nginx.conf and add the below lines

```
add_header X-Frame-Options "SAMEORIGIN";
```

add_header X-XSS-Protection "1; mode=block";

21. Prevent Directory Listing

The directory listing feature on Nginx is controlled by the ngx_http_index_module. Directory listing is disabled by default on the Nginx configuration file.

Requirements:

- To Stop the listing of Directory in the browser.

Implementation:

- Configuration: Nginx Level
  - ➢ Go to nginx.conf under /etc/httpd/conf/ and change the section under location /{ } section:
    autoindex off;

22. Limit Request Body Size

Requirements:

- To Limit the size of the request body.

Implementation:

- Configuration: Nginx Level
  - ➢ Go to nginx.conf and add the below line

    client_max_body_size <size>;

    Note: <size> in Megha Bytes Eg : 5M

    client_max_body_size 5M;

23. Content Security Policy

Requirements:

- The Content-Security-Policy allows you to reduce the risk of XSS attacks by allowing you to define where resources can be loaded from, preventing browsers from loading data from any other locations. This makes it harder for an attacker to inject malicious code into your site.

  Setting Content Security Policy Header:
  Header set Content-Security-Policy "script-src ‘self’ <source> <source>;”

  Where,
  ***script-src*:** Defines valid sources for JavaScript files.
  ‘self’: It means sources that have the same scheme (protocol), same host and same port as the file the content policy is defined in.
  ***<source>*:** whitelisted/pre-defined sources from where we want the resources to be loaded.

How to use different directives, what do they do?

1. **default-src:** The default policy for loading JavaScript, images, CSS, fonts, AJAX requests, etc.
2. **script-src:** Defines valid sources for JavaScript files.
3. **style-src:** defines valid sources for CSS files.
4. **Img-src:** defines valid sources for images.
5. **font-src**: Define from where the protected resource can load fonts.

There are other directives as well if the default-src is set to 'self' all the directives will automatically set to 'self'

Implementation:

- Configuration: Nginx Level
  - ➢ Go to nginx.conf file and add the below line

```
add_header Content-Security-Policy "default-src 'self';script-src 'self' 'unsafe-inline'
'unsafe-eval' https://maps.googleapis.com/ http://ajax.googleapis.com/;img-src 'self'
data: https://khms0.googleapis.com/ https://khms1.googleapis.com/
https://maps.gstatic.com/ https://www.gstatic.com/ https://maps.googleapis.com/
https://lh3.ggpht.com/ https://cbks0.googleapis.com/;font-src 'self'
https://fonts.gstatic.com/;style-src 'self' 'unsafe-inline'
https://fonts.googleapis.com/;connect-src 'self'";
```

Note: the whole header setting should be written in a single line.

The above setting contains the list of all the whitelisted links/resources allowed by the application. All the other links will be blocked.