OWASP Top Ten Web Application Security Risks Alloantibody Exchange Review

R. George Hauser, 6/1/2023

This document describes our review of the OWASP Top 10 Web Application Security Risks. We have described important aspects of our security plan in the responses. For suggestions with less relevance to our web application, for example because we use Microsoft Azure for a particular security concern, we have omitted a response to avoid redundancy and improve readability. We have traced these answers to specific sections in our code and/or Microsoft's documentation (e.g., ASP.NET, Azure).

A01:2021-Broken Access Control

Broken access control is the top security risk. It allows users who should not access a particular website resource to access it.

- To avoid this scenario, we deny resources by default.
- We implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
- Our policies to control access are defined once and re-used through the application.
- We have not enabled cross-origin requests. This is the default behavior in ASP.NET.
- Users cannot create, update, or delete records.
- We perform front-end and server-side validation of user input. We use models to further define acceptable user inputs.
- We have disabled access to the web root directory. We have reviewed the web root to ensure it contains only the files it is intended to house (e.g., CSS, JS).
- We log login attempts and outcomes.
- We invalidate session identifiers on logout. We have an inactivity timer to log out the user.

A02:2021-Cryptographic Failures

We have reviewed the cryptographic features of our website. Specifically, it has end-to-end encryption in transit and at rest for protection. Encryption in transit uses TSL 1.2. The encryption (e.g., standard algorithms, protocols, and keys) are managed by Azure. We minimized storage of sensitive data. We use HSTS and HTTPS redirection, which re-routes HTTP requestion to HTTPS. We have disable caching for response that contain sensitive data. We have reviewed these features to ensure they work.

A03:2021-Injection

We have reviewed out application to reduce or eliminate the risk of SQL injection attacks. We have a on both front-end and server-side user input validation. We use categorical lists instead of strings where possible. When this is not possible (e.g., names), we use positive input validation. We do not require special characters, and therefore, we remove them from user inputs. We use parameterized queries. And, we limit the number of results returned from the query

A04:2021-Insecure Design

Insecure design refers, not to implementation, but rather the conception of the software.

- We have followed professional advice on how to secure the design of the security. “PleuralSight: Securing ASP.NET Core 6 with OAuth2 and OpenID Connect”.
- We require recent citations from Microsoft for all security related operations.
- We have a list of checks/unit tests we use to evaluate the project’s security.
- We have a policy system in Microsoft Azure AD with roles assigned based on user needs. We employ these policies within the web application to control authorization.

A05:2021-Security Misconfiguration

Security misconfiguration could lead to security vulnerabilities.

- We have followed professional advice on how to setup our authentication, “PleuralSight: Securing ASP.NET Core 6 with OAuth2 and OpenID Connect”.
- We have followed Microsoft’s advice for implementation of authorization.
- We have followed Microsoft’s advice to require authenticated users as a default.
- We have reviewed and removed unused dependencies, features, components, and files. Our web application only uses Microsoft libraries. (We do not have any non-Microsoft libraries.)
- Error handling has a designated page instead of a stack trace.

Need to implement different credentials for development and production.

A06:2021-Vulnerable and Outdated Components

Vulnerable and outdated components represent a security vulnerability. We have reviewed and removed unused dependencies, features, components, and files. Our web application only uses Microsoft libraries. (We do not have any non-Microsoft libraries.) We use Azure, which maintains updated Microsoft products.

A07:2021-Identification and Authentication Failures

Identification and authentication are important to provide appropriate access to a user.

- We use Microsoft Azure AD with its OpenIDConnect/OAuth2 protocol for user authentication. Users may recycle their health system’s Microsoft AD credentials for use with our system. If they receive an AD account from us, we require multifactor authentication.
- We do not send user information in the URL (e.g., session identifiers in the URL).
- We audit user sign-in (e.g., success, failure).
- ASP.NET handles the generation of session identifiers by default.
- We clear cookies as part of sign out.

A08:2021-Software and Data Integrity Failures

“Software and data integrity failures” refer to vulnerabilities introduced through shared code obtained or modified by a nefarious source.

Our web application uses only code from Microsoft. This code is downloaded through Microsoft’s Visual Studio NuGet package manager to ensure authenticity.

The description also mentioned unsigned or unencrypted serialized data. We encrypt our client communications.

A09:2021-Security Logging and Monitoring Failures

Appropriate logging and monitoring are necessary to detect on-going threats.

- We audit user sign-in (e.g., success, failure).
- We audit database access and usage.
- Audit logs are stored in a secure location.
- Audits we use are services of Microsoft Azure.
- The log files are easily downloaded from Azure and viewed in SSMS.
- We have procedures for the timely review of audit logs.

A10:2021-Server-Side Request Forgery

We have reviewed the server-side request forgery attack.

1. With our web application deployed on Azure as software-as-a-service, rather than a network, port scans vulnerabilities are mitigated. This setup also minimizes access to other resources such as local files, simply because they do not exist in our configuration.
2. We sanitize and validate all client-supplied input data.
3. We do not send “raw responses” to our clients. That is, user input from a client is not returned to the client.
4. We employ firewalls with a positive allow list for website updates and database access.