endava

—

ASP.NET Security

Agenda

1. ON WEB APPLICATION SECURITY
2. COMMON VULNERABILITIES IN SOFTWARE
3. SECURE DATA STORAGE
4. AUTHENTICATION AND AUTHORIZATION
5. RESOURCES

1

On Web Application Security

Why Web Application Security is Important?

Nine out of ten web applications have security vulnerabilities. This is the rather frightening result of a study released in 2020 by Positive Technologies, a provider of various security solutions. Obviously, such results can often be biased towards the business model of those who conduct them, but several other studies from previous years yielded similar outcomes. Here’s a report about one study from as far back as 2009:
https://www.darkreading.com/risk/majority-of-web-apps-have-severe-vulnerabilities

They also found out that about four out of five web application vulnerabilities are part of the code, instead of, say, the server configuration. From this, we can deduce two trends:
- The major security risk for web applications is the code it is made of.
- The problem is industry-wide, and the situation does not seem to get better.

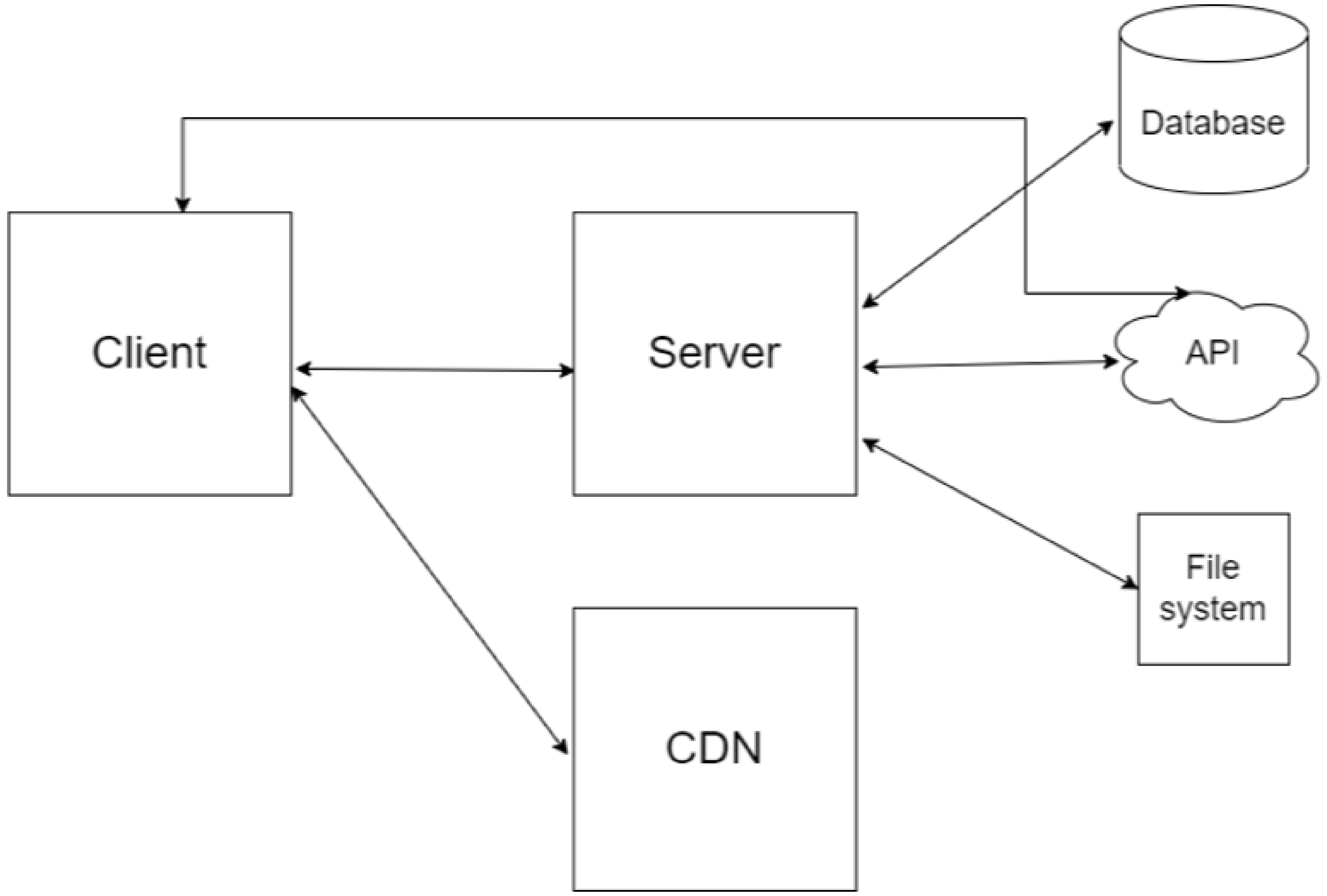
ASP.NET Core: History and Options

ASP.NET has a long history, tied to that of .NET, which was first released as a beta in 2001, and as a final version 1.0 in early 2002. Back then, the software package was called “.NET Framework”, and contained a server web application framework called ASP.NET. (The first three letters were carried over from the previous Microsoft web technology ASP, which was short for “Active Server Pages”.)

ASP.NET Core provides many options to create web sites: different server frameworks and approaches, and also different techniques and formats for client-side aspects. For example: MVC, Razor Pages, Web API, Blazor. Many components essentially lead to a large attack surface area, so there are many places where things may go wrong security-wise.

Identifying and Mitigating Threats

In a modern web application, there will also be other components, such as:
- databases
- other external services
- file system resources
- CDNs (Content Delivery Networks) holding libraries and other, mostly static, assets



Here are some of the things that web applications need to provide safeguards against:
- User data sent to the database might contain malicious commands that are then run on the database. A typical attack is called SQL injection.
- User data sent to the server and later sent to a client may contain unwanted content such as malicious JavaScript code. This is commonly referred to as Cross-site Scripting.
- Data exchanged between client and server may be intercepted, or stolen, putting parts of the application at risk. Using a secure transport mechanism can help.
- Insufficient authorization might grant users access to server resources they should not be allowed to see.
- The way authentication works in the web might be abused by crafting HTTP requests, leading to users causing the application to take undesired actions. Cross-Site Request Forgery is a typical attack in this space.
- Unexpected user input—such as too much data, to little data, or incorrect data—leads to an unwanted behavior of a web application, or provokes error messages.
- Error messages may reveal internal information about the server, which could be useful for an attacker. On the other hand, internal errors need to be properly handled and logged.- Assets hosted on a third-party site (think CDNs) may have been manipulated.

Security-Related APIs

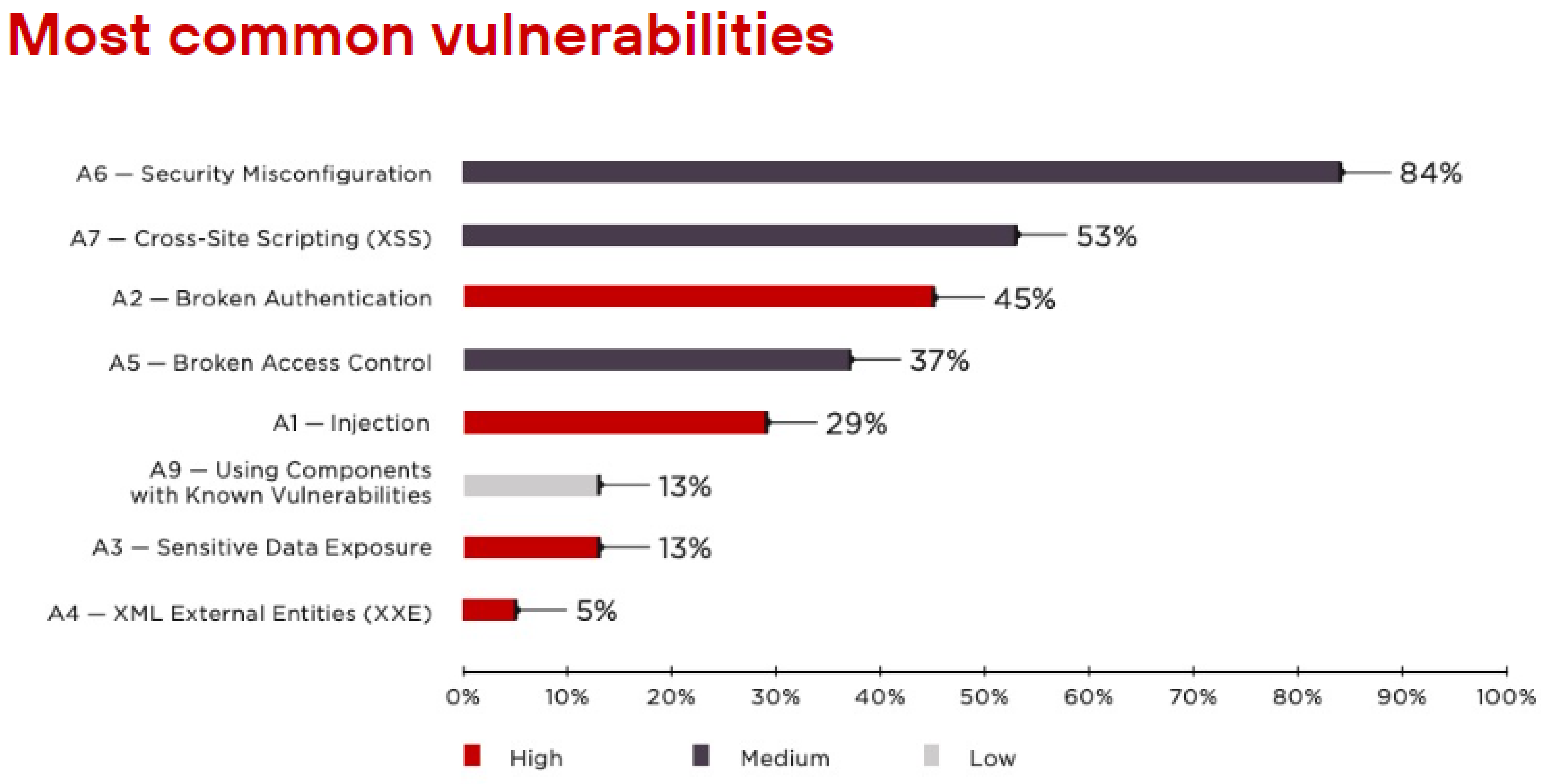
ASP.NET Core and associated technologies comes with several security-related APIs that cannot always be directly mapped to a specific attack.

Here are a few examples:
- ASP.NET Core Identity provides an API for handling authentication and authorization in a web application, including login/logout, profile data, roles, and more.
- Secure communication may be enforced by a variety of options and approaches, including redirecting to HTTPS, securing cookies, and even disabling HTTP.
- Security-related HTTP headers may be explicitly set using web.config, in controllers, and implicitly by defining settings in Program class (in .NET versions prior to 6, the Startup class was the go-to place for this).
- Passwords should not be stored in clear text, but they should be encrypted, or better, hashed, and ASP.NET Core supports relevant formats and algorithms.
- Cloud providers like Microsoft Azure or Amazon AWS have their own APIs for storing secrets such as connection strings or passwords.

2

Common Vulnerabilities in Software

Most Common Vulnerabilities



Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a security vulnerability which enables an attacker to place client side scripts (usually JavaScript) into web pages. When other users load affected pages the attacker's scripts will run, enabling the attacker to steal cookies and session tokens, change the contents of the web page through DOM manipulation or redirect the browser to another page. XSS vulnerabilities generally occur when an application takes user input and outputs it to a page without validating, encoding or escaping it.

Note: This attack should probably be called "JavaScript injection"; a group of Microsoft security engineers came up with "Cross-site scripting" instead in early 2000, and that name stuck. In case you are wondering: the acronym CSS was already taken by Cascading Style Sheets, so XSS was being used.

https://docs.microsoft.com/en-us/aspnet/core/security/cross-site-scripting?view=aspnetcore-6.0

SQL Injection Attacks

SQL stands for “Structured Query Language” and was invented in the 1970s to provide a language to communicate with a relational database to, among other things, read and write data to it. Even the creators (Donald D. Chamberlin and Raymond F. Boyce) could probably not imagine back then that their brainchild would still be in use almost 50 years later. That there would be web sites where user input will be sent to a web server where it will be put in SQL queries that will then be executed against a database was certainly far from everyone’s imagination back then.

However, SQL suffers from an architectural problem that many query languages have: data and commands are in the same string. If an SQL query is the result of a concatenation of strings, some of them contributed by the user, there might be a risk.

https://docs.microsoft.com/en-us/ef/core/querying/raw-sql

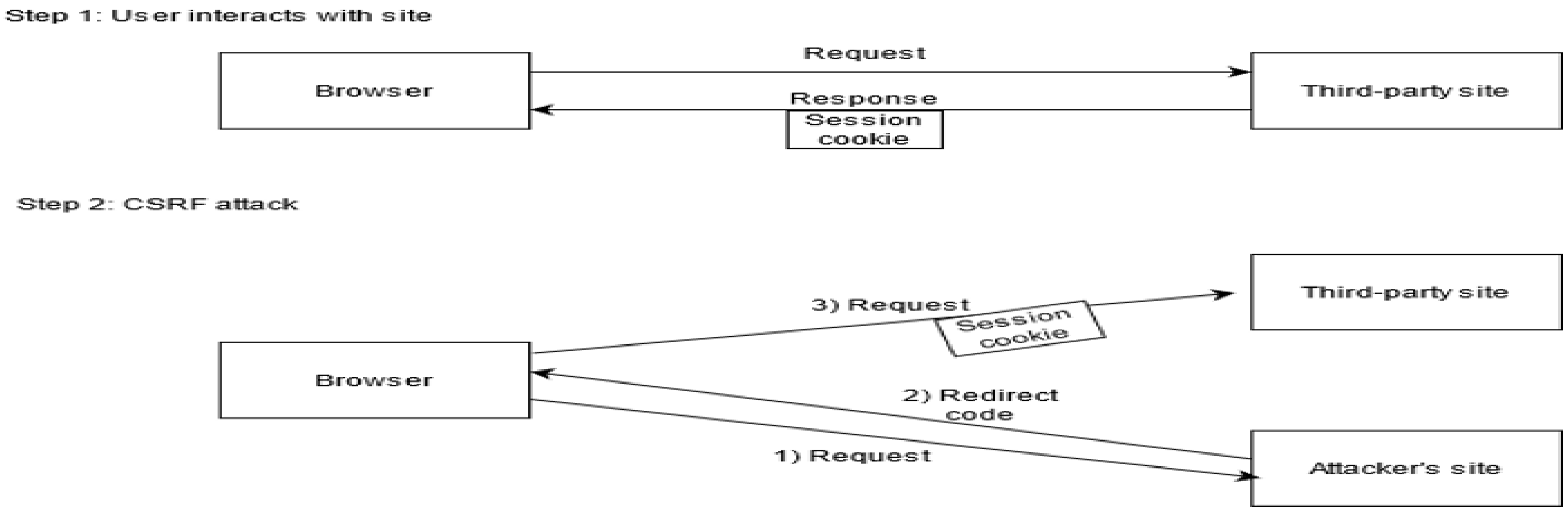
Cross-Site Request Forgery (XSRF/CSRF) Attacks

Cross-site request forgery (also known as XSRF or CSRF) is an attack against web-hosted apps whereby a malicious web app can influence the interaction between a client browser and a web app that trusts that browser. These attacks are possible because web browsers send some types of authentication tokens automatically with every request to a website. This form of exploit is also known as a one-click attack or session riding because the attack takes advantage of the user's previously authenticated session.

Example:
In 2005, security researcher Samy Kamkar found a security vulnerability in the then popular social network, MySpace. He managed to inject JavaScript code into his profile page, a classical Cross-Site Scripting (XSS) attack as explained previously (Common Vulnerabilities in Software: Cross-Site Scripting (XSS)). The JavaScript code, however, did something really interesting: when executed, it issued an HTTP request on the victim's behalf, adding them to Kamkar's friends list. This started a chain reaction, and less than 20 hours later, Kamkar had over one million friends on MySpace.

Kamkar himself provided a detailed reconstruction of the events at https://samy.pl/myspace/, and a thorough technical description of the attack at https://samy.pl/myspace/tech.html. However I recommend that you read this chapter first, so that you know all the required technical details about these kind of attacks.

https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-6.0#antiforgery-in-aspnet-core



3

Secure Data Storage

Storing Secrets

Never store passwords or other sensitive data in source code. Production secrets shouldn't be used for development or test. Secrets shouldn't be deployed with the app. Instead, production secrets should be accessed through a controlled means like environment variables or Azure Key Vault.

Safe storage of app secrets:
- Environment variables
- The Secret Manager
- The appsettings.json file
- Storing secrets in the cloud (https://docs.microsoft.com/en-us/aspnet/core/security/key-vault-configuration?view=aspnetcore-6.0)

https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-6.0&tabs=windows

Handling Passwords

Storing in Plaintext

Are you serious that we need to talk about this???

Implementing Password Hashing (Encrypted Passwords)

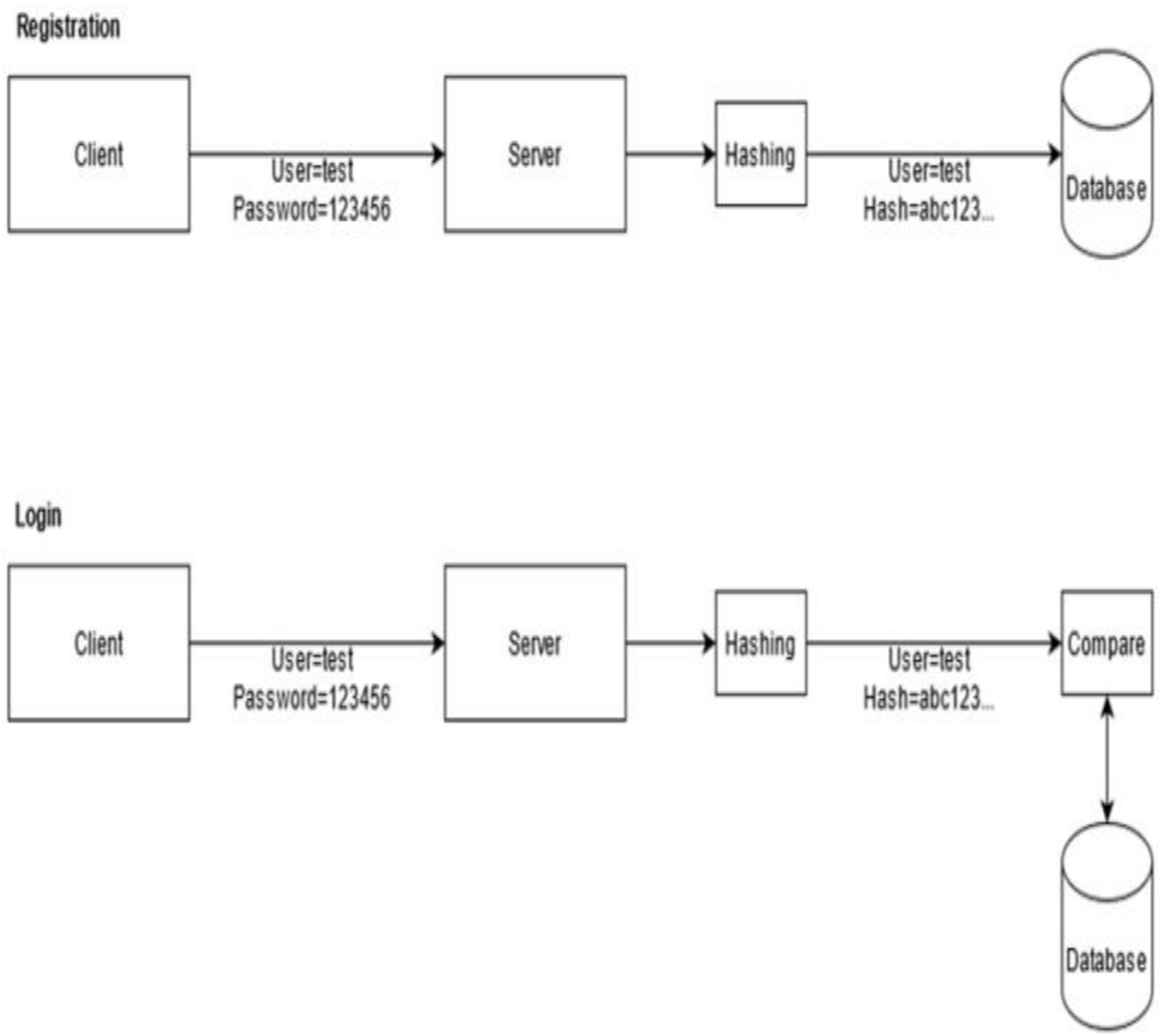
Insecure Hashes - The MD5 and SHA1 hash algorithms have been shown to be manipulable such that two passwords could create the same fingerprint.

Unsalted Hashes - The primary conceptual weakness of hashes is that an attacker could generate hashes based on a huge list of common passwords and create a reverse-lookup dictionary, or “rainbow table”. To avoid this, a “salt” is a random bit of data added to the data being hashed.

No Future-Proofing - The best password protection algorithms allow you to specify a number of “cycles” to run the password through. Combined with a unique salt for each password, this makes it expensive and slow to try to brute force a password hash. As computing power increases and becomes less expensive over time, you can simply add some zeros to the number of cycles to run each password through and keep up. BCrypt and PBKDF2 are both popular implementations which use this concept.

The Right Path
- Use a strong hash algorithm
- Add a unique, random salt to each hash
- Use a “future-proof” algorithm

Figure 8.4 Password hashing in action



4

Authentication and Authorization

Authentication vs Authorization

Authentication is a process in which a user provides credentials that are then compared to those stored in an operating system, database, app or resource. If they match, users authenticate successfully, and can then perform actions that they're authorized for, during an authorization process. The authorization refers to the process that determines what a user is allowed to do.

Another way to think of authentication is to consider it as a way to enter a space, such as a server, database, app or resource, while authorization is which actions the user can perform to which objects inside that space (server, database, or app).

Authentication

Authentication is the process of determining a user's identity. Authentication may create one or more identities for the current user.

In ASP.NET Core, authentication is handled by the authentication service, IAuthenticationService, which is used by authentication middleware. The authentication service uses registered authentication handlers to complete authentication-related actions. Examples of authentication-related actions include:

- Authenticating a user.
- Responding when an unauthenticated user tries to access a restricted resource.

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-6.0

Authorization

Authorization is the process of determining whether a user has access to a resource.

Authorization refers to the process that determines what a user is able to do. For example, an administrative user is allowed to create a document library, add documents, edit documents, and delete them. A non-administrative user working with the library is only authorized to read the documents.

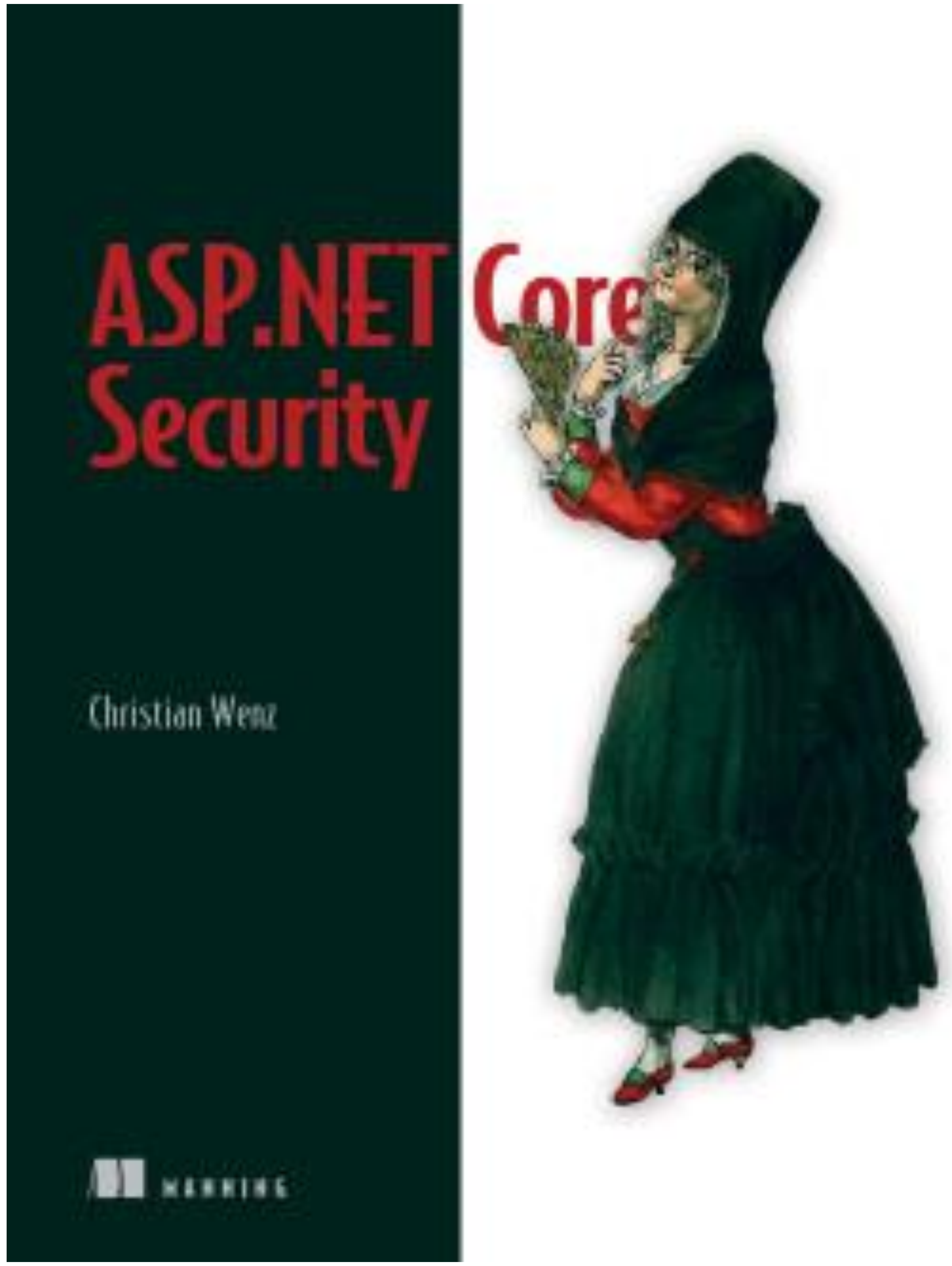
Authorization is orthogonal and independent from authentication. However, authorization requires an authentication mechanism.

https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction?view=aspnetcore-6.0

6

Resources

ASP.NET Core Security



Online preview: https://livebook.manning.com/book/asp-net-core-security/welcome/v-11
Code samples: https://www.manning.com/downloads/2371

Microsoft Docs

https://docs.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-6.0

endava