Checklista Säker Programmering

Denna checklista är framtagen via en workshop där studenterna fått lyfta det man tänkt på samt lärarens tillägg. Checklistan är till för uppgiften säker programmering i kursen IT-säkerhet, den är inte helt komplett vad gäller allt man bör tänka på när en webbapplikation utvecklas.

Viktigt att tänka på är vilket operativsystem som ska användas och utifrån det vilket IDE (utvecklingsmiljö) som ska användas (helst ramverk med viss inbyggd säkerhet) och vilket språk som ska användas (har du möjlighet välj det du kan bäst).

I uppgiften Säker programmering är ni tvungna att jobba med Visual Studio och .NET Core MVC vilket betyder att viss säkerhet är inbyggd.

Att göra i uppgiften:

Du ska kommentera i koden på lämpligt ställe (i början av en fil, ovanför en klass, en funktion, metod) genom att skriva ex. ”Checklistan 2.3 Vi stänger av auto-complete för att säkra att ingen information finns kvar i formulär till obehörig användare”. ***ALLA punkter i checklistan ska finnas som kommentar i koden.***

1. **Datalagring**
   Data som ska finnas i en databas (från början eller som fylls på via applikationen) ska klassificeras för att veta om man behöver:
   1.1. Olika databaser som hanterar olika typer av data (ex. inloggningsuppgifter, loggar, datahanteringen etc).
   1.2. Förhindra injection (validering – se datainhämtning)
      1.2.1. Antingen med parameterisering (om språket är SQL)
      1.2.2. Eller med ORM (EntityFramework) och LINQ för automatiserad parameterisering
   1.3. Kryptera känslig data (välj rätt typ av kryptering)

   Dessutom bör man tänka på vem som har tillgång till vad för data (se auktorisering) och om det är skriv och/eller läsrättigheter. Fundera också över hur länge data ska lagras och om det ska tas bort automatiskt (ska då programmeras in – tänk GDPR)

2. Datainhämtning

Designa säker inhämtningen av data genom att

2.1. Använda formulärkontroller där användaren väljer istället för att skriva (så långt det är möjligt)
2.2. Validera input med data annotations (som läggs på dataklasserna s.k poco-klasser) – vilket tvingar användaren att ge oss rätt data
2.3. Undvik auto-complete
2.4. Jobba med Razor View (@variabelnamn = automatisk html encode för output, samt inbyggt skydd mot XSS)
2.5. Skydd mot CSRF
    2.5.1. Formulär innanför form-element med method=”post”
    2.5.2. [ValidateAntiForgeryToken] på action-method (som anropas av formuläret)

Data som ska vidare till lagring ska krypteras (se datalagring), samt följa de lagar som finns (ex. GDPR). Tänk också igenom vad för data som ska hämtas in från användare samt visas upp för användaren.

3. Autentisering

Helst ska man implementera en flerfaktors-lösning (2 eller fler), men tänk på att

3.1. Jobba med IdentityUser och IdentityRole
3.2. Generellt felmeddelande vid inlogggning
3.3. Försök till åtkomst via url går automatiskt till login-sidan
3.4. Fördröjning och/eller captcha vid flera försök, samt lockout vid för många felaktiga försök
3.5. Lösenordet ska vara hashad (kryptering)
3.6. Implementera lösenordspolicy (längd och komplexitet)
3.7. Ta bort session/cookie vid utloggning

Tänk på att det ska finnas en process och korrekt implementerad funktion för att hantera bortglömda lösenord.

4. Auktorisering

Behörighetsnivån är starkt kopplad till datalagring och autentiseringen.

4.1. Begränsa behörighet (inte mer än man behöver tillgång till)
    4.1.1. Jobba med Authorize och Roles, samt AllowAnonymus
4.2. Försök till åtkomst via url när man är inloggad leder till AccessDenied

5. **Felhantering**
   5.1. Validera att parametrar har innehåll
   5.2. Generella felmeddelanden vid fel inmatning i formulär (specifikt inloggning)
   5.3. Viktig kod ska ligga i try-catch
   5.4. Hantera exceptions korrekt
      5.4.1. Vad ska hända – felmeddelande/defaultinställning
      5.4.2. Generellt felmeddelande till användaren
      5.4.3. Logga stacktrace enligt beslut

   Felaktigheter som inte fångas i exceptions och där användaren upptäcker problem ska ha en möjlighet att kommunicera det till rätt instans.

6. **Loggning**
   Diskussioner ska ha förts med ansvariga om vad som ska loggas och hur det ska göras och om det ska loggas automatiskt när inget händer eller när något händer som är ovanligt. Samt om loggarna ska kategoriseras och sparas i olika tabeller.
   6.1. Logga viktiga saker (inloggningar, vad som gör, felmeddelanden)
   6.2. Inkludera tid, ip-adress, vem och vad som gjordes
   6.3. Kryptera loggningarna (dekryptering för att kunna läsa loggarna)
   6.4. Förvara loggningarna i annan databas (och annan server)

   Det ska också finnas en process om vem/vilka som läser loggarna och när (behörighetsnivå). Och hur länge loggarna ska finnas och hur de tas bort.

7. **Inför driftsättning**
   7.1. Kommentera vad som ska bort eller förändras innan driftsättningen

   Självklart ska funktionalitet testas och systemet testköras. Viktig information om behörighet och åtkomstnivåer ska finnas dokumenterad (troligen via hotmodelleringen)