UPPSALA UNIVERSITET CAMPUS GOTLAND
Institutionen för informatik och media
Utbildningsprogram: Kandidatprogram i Systemvetenskap (inriktning Programvaruteknik)
Kursansvarig lärare: Millan Lundgren

Datum: 2022-01-18

Rapport om Säkerhetstestning
IT-säkerhet, 7,5 hp

Säkerhetstestning

Testning av en applikation för Småstads Kommun

Fanny Johansson

1 Problemformulering

CodeSec har fått i uppdrag att genomföra kontrollering och testning av en applikation för rapportering av miljöbrott som tillhandahålls av Småstads Kommun. Det som kontrolleras på applikationen är bland annat hur säker den är, hur välskriven den är samt hur säkert datan som finns i applikationen är.

För att utföra kontrolleringen och testningen på applikationen kommer kodgranskning att göras, manuella tester samt mindre penetrationstester. De brister som finns i applikationen dokumenteras och sammanställs i denna rapport.

1.1 Antaganden

Vid driftsättning av en applikation missas ofta en gedigen genomgång av hur säker applikationen är och hur välskriven koden är. I denna applikation kan det visa sig att data som lagras i den inte lagras på ett tillräckligt säkert sätt, vilket kan göra att obehöriga kan komma åt datan. Dessutom kan det finnas kod bakom applikationen som egentligen borde tagits bort vid driftsättning, som kan ge information till hackers som inte ska vara tillgänglig för dem. Därför kommer koden gås igenom noggrant och flertal kontroller kommer göras för att minimera riskerna för detta.

2 Resultat

För att utreda hur hög säkerhet applikationen för Småstads kommun har görs först en klassificering av den data som lagras. Därefter kommer flera typer av säkerhetstester att göras, nämligen manuella tester, penetrationstester samt kodgranskning med verktyg. Detta för att säkerställa att applikationen har en så hög säkerhetsnivå som möjligt vid driftsättning.

2.1 Klassificering av data

Systemet som byggs för Småstads kommun innehåller flera typer av data som är olika känslig. Det finns delvis data som inte är särskilt känslig som endast behöver det mest basala skyddet. Exempel på denna typ av data är all information som lagras kring själva miljöbrotten som rapporteras. Platsen där ett miljöbrott skett, vad för typ av miljöbrott, datum för brottet och allmän information om ärendet är data som inte är känslig. Denna data kan klassas måttlig ur säkerhetsaspekten konfidentialitet i MSB:s (2009) matris för informationsklassning, vilket innebär att informationen inte får göras tillgänglig eller avslöjas för obehöriga och görs detta kan konsekvensen bli en måttlig negativ påverkan på Småstads kommun. Eftersom denna data är en central del av applikationen då det är just miljöbrott den behandlar går det även att klassa denna data utifrån säkerhetsaspekten riktighet, vilket innebär att informationen inte förändras och att den är riktig. Det är nämligen viktigt att datan som lagras kring miljöbrotten stämmer så att ärenden kan följas upp korrekt. Vid förlust av riktighet kan konsekvenserna därför bli betydande för Småstads kommun bedömt utifrån MSB:s (2009) matris.

Systemet lagrar även känslig data, så som inloggningsuppgifter som inkluderar både användarnamn och lösenord. Det är därför av stor vikt att denna data skyddas extra väl, vilket bör göras genom att hascha lösenorden när de sparas i databasen. Detta gör att om någon skulle lyckas ta sig in i databasen över inloggningsuppgifter kan denne inte komma åt lösenorden som lagras i klartext. Dessutom är det säkrast att lagra inloggningsuppgifter i en separat databas, för att

minimera dataläckage vid intrång. Även denna typ av data hamnar i säkerhetsaspekten konfidentialitet, och förloras denna data kan konsekvensen bli allvarlig enligt MSB:s (2009) matris.

Annan känslig data som systemet lagrar är uppgifter kring de anställda och Småstads kommuns avdelningar. Datan innefattar för- och efternamn på de anställda, vilken avdelning de arbetar på, deras roll på företaget samt deras unika anställnings-ID. Det kan därför vara värt att överväga om delar av eller hela denna data ska krypteras när den sparas ned i databasen, så att inga obehöriga kan komma åt denna vid eventuella databasintrång. Enligt Internetstiftelsen (2022) används kryptering för att göra viss information svårläslig för alla som ej bör ha tillgång till denna information, och för att göra informationen läslig igen krävs att den dekrypteras. Detta innebär att ovannämnd information lagras krypterad i databasen, och vid visning för behöriga dekrypteras den automatiskt. Även denna data bör lagras i en separat databas för ökad säkerhet. Utifrån MSB:s (2009) matris klassas denna data utifrån aspekten konfidentialitet vilket betyder att datan inte får göras tillgänglig eller avslöjas för obehöriga, vilket är viktigt eftersom det handlar om anställdas personuppgifter. Med tanke på att det är personuppgifter involverade kan förlust av datan innebära betydande konsekvenser för Småstads kommun (MSB, 2009).

Det finns även viss data i applikationen som är svår att klassificera som antingen känslig eller icke-känslig. Denna data är de bilder och prover som tas vid brottsplatsen. Bilderna och proverna kan nämligen innehålla känslig data, men troligtvis gör dem oftast inte det då de mestadels är till för att ge en tydligare bild över vad som inträffat. Det mest rimliga i detta fall är därför att ta det säkra före det osäkra och kryptera även denna data, då det kan vara känslig information som bilderna och proverna innehåller. Däremot är det viktigt att denna data är riktig, och den bedöms därför ur säkerhetsaspekten riktighet, där förlust av datan kan bli betydande. Detta innebär att det är viktigt att datan som anges är korrekt, och konsekvensen vid förlust innebär en betydande negativ påverkan på Småstads kommun.

Uppdelningen av data över olika databaser bör därför se ut som i figur 1:

Figur 1: Databasfördelning

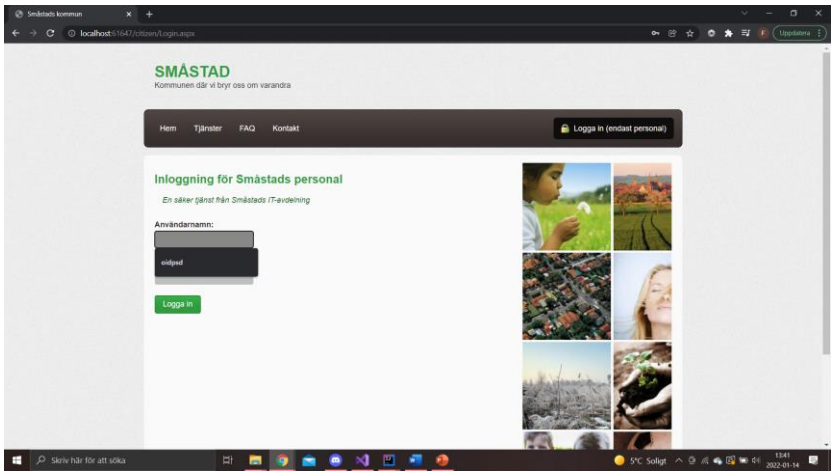
Sammanfattningsvis innehåller applikationen en del känslig data som kan ge allvarliga konskenvenser vid dataförlust eller databasintrång. Därför bedöms datan i applikationen sammantaget ha en allvarlig konsekvensnivå, vilket gör att detta blir klassificeringen för den lagrade datan.

2.2 Manuell kodgranskning

Som applikationen är uppbyggd i dagsläget lagras all data i samma databas. Hänvisat till den checklista (bilaga 1) som är framtagen för säker programmering bryter detta mot punkt 1.1: Olika databaser som hanterar olika typer av data (ex. inloggningsuppgifter, loggar, datahantering etc.). Vid användning av flera olika databaser enligt förslaget i figur 1 minskar risken för dataläckage, eftersom vid intrång i en av databaserna kommer den som gör intrånget endast åt en typ av data istället för all data som den gör om det endast finns en gemensam databas.

Ingen typ av kryptering finns i dagsläget implementerad i applikationen. Detta bryter mot checklistan (bilaga 1) punkt 1.3: Kryptera känslig data. I applikationen lagras känslig data som inte bör lagras i klartext i databasen, då eventuella hackers som tar sig in i databasen enkelt kan se denna data. Därför är det av stor vikt att kryptering implementeras på flera ställen som nämnt i dataklassificeringen.

Punkt 2.3 i checklistan (bilaga 1) beskriver att applikationen bör undvika auto-complete i fritextfält. Auto-complete är inte något som inaktiverats i applikationen, och detta visas i figur 2. Auto-complete kan ge hackers ledtrådar om inloggningsuppgifter och bör därför inaktiveras i applikationen för att säkerställa säker inhämtning av data. Enligt Mozilla (2022) bör auto-complete stängas av vid input av känslig information och detta kan enkelt lösas genom att lägga till attributet autocomplete=”off” på elementen som har hand om inputen.



Figur 2: Granskning kring auto-complete i fritextrutor.

Applikationen innehåller en inloggningsfunktion där handläggare, samordnare och chefer kan logga in och se relevant information kring händelser. Däremot är ingen lösenordspolicy implementerad som säkerställer att de anställda har säkra lösenord. Enligt Microsoft (2021) är en lösenordspolicy viktig eftersom det förebygger att hackare kan gissa sig till vanliga lösenord manuellt eller använda verktyg för detta. Microsoft (2021) nämner att ett starkt lösenord bör innehålla minst 8 tecken och innefatta en kombination av både siffror, bokstäver och symboler.

Vid försök att komma åt en sida som kräver inloggning utan att vara inloggad bör resultatet vara en AccessDenied-sida enligt checklistan (bilaga 1) punkt 4.2. Ett försök gjordes att nå …/manager/crimemanager.aspx, vilket resulterade i att man blir kvar på login-sidan och ingen ny sida dyker upp som är specifikt gjord för AccessDenied.

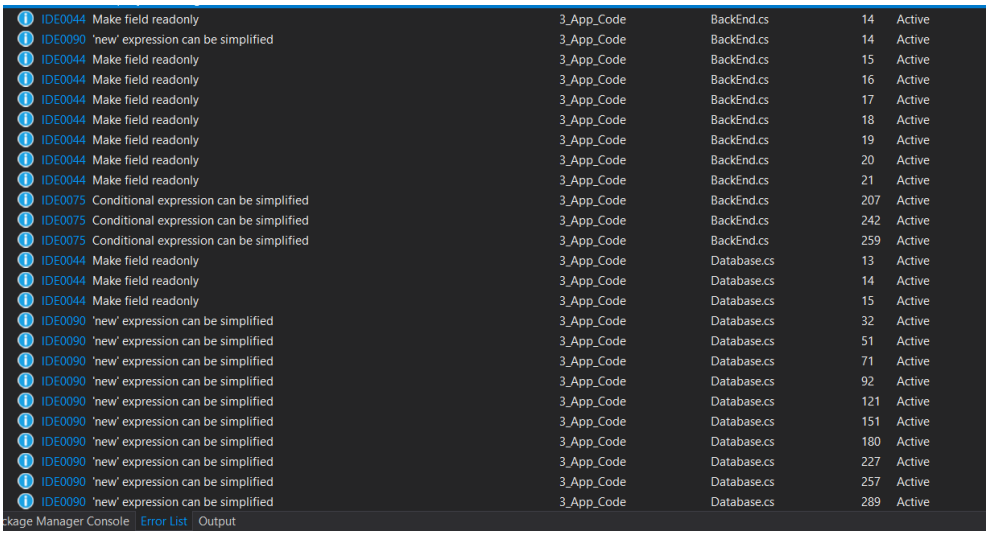
Det finns i dagsläget ingen loggningsfunktionalitet i applikationen, vilket innebär att händelser i systemet inte dokumenteras automatiskt. Detta är enligt checklista (bilaga 1) punkt 7 något som ett säkert system bör innehålla då det blir lättare att hitta eventuella fel samt om någon obehörig tagit sig in i systemet.

Vid granskning av sidan upptäcktes även ett fel vad väljer behörighet på de inloggade vyerna. Loggar man in via samordnarens konto visas vyn …/coordinator/StartCoordinator.aspx, men det är fortfarande möjligt att via URL ta sig åt andra vyer som egentligen kräver annan inloggning. Det går till exempel bra att ändra URL:en till …/manager/StartManager och hamna på just denna vy, vilket inte bör vara möjligt när man inte är inloggad som chef.

2.3 Kodgranskning med verktyg

Enligt Lundgren (2021) är granskning av kod viktig för att kunna upptäcka och åtgärda fel tidigt i utvecklingsprocessen då en kodgranskning ofta kan avslöja direktplaceringen av en bugg i ett program. Lundgren (2021) förklarar att kodgranskning hjälper med att se till att inga fel finns i koden, minimera riskerna att få problem i systemet, bekräftar att ny kod följer uppsatta riktlinjer samt ökar effektiviteten av ny kod.

För att granska koden automatiskt användes FxCop som finns inbyggt i Visual Studio. Detta görs för att ta reda på vilka sårbarheter och felaktigheter som finns i koden men som kanske inte är synliga vid en manuell kodgranskning som gjordes tidigare.

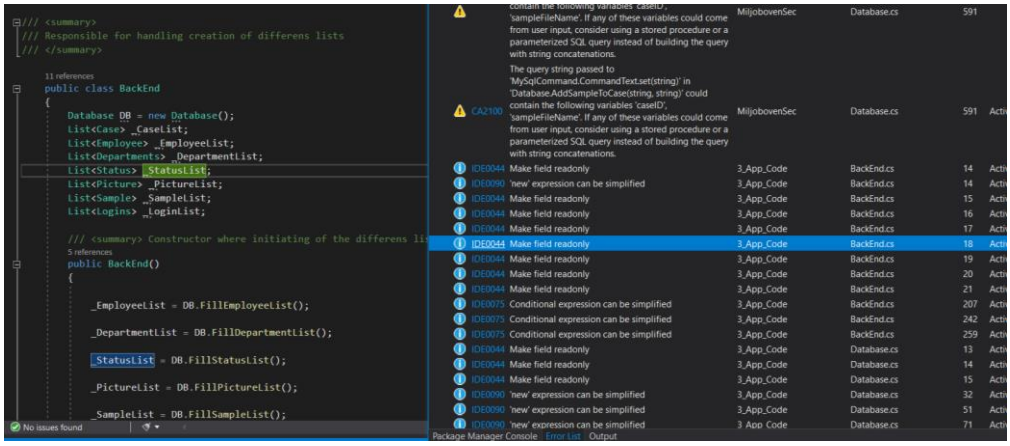


IDE0044 Make field readonly	3_App_Code	BackEnd.cs	14	Active
IDE0090 'new' expression can be simplified	3_App_Code	BackEnd.cs	14	Active
IDE0044 Make field readonly	3_App_Code	BackEnd.cs	15	Active
IDE0044 Make field readonly	3_App_Code	BackEnd.cs	16	Active
IDE0044 Make field readonly	3_App_Code	BackEnd.cs	17	Active
IDE0044 Make field readonly	3_App_Code	BackEnd.cs	18	Active
IDE0044 Make field readonly	3_App_Code	BackEnd.cs	19	Active
IDE0044 Make field readonly	3_App_Code	BackEnd.cs	20	Active
IDE0044 Make field readonly	3_App_Code	BackEnd.cs	21	Active
IDE0075 Conditional expression can be simplified	3_App_Code	BackEnd.cs	207	Active
IDE0075 Conditional expression can be simplified	3_App_Code	BackEnd.cs	242	Active
IDE0075 Conditional expression can be simplified	3_App_Code	BackEnd.cs	259	Active
IDE0044 Make field readonly	3_App_Code	Database.cs	13	Active
IDE0044 Make field readonly	3_App_Code	Database.cs	14	Active
IDE0044 Make field readonly	3_App_Code	Database.cs	15	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	32	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	51	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	71	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	92	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	121	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	151	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	180	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	227	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	257	Active
IDE0090 'new' expression can be simplified	3_App_Code	Database.cs	289	Active

ckage Manager Console | Error List | Output

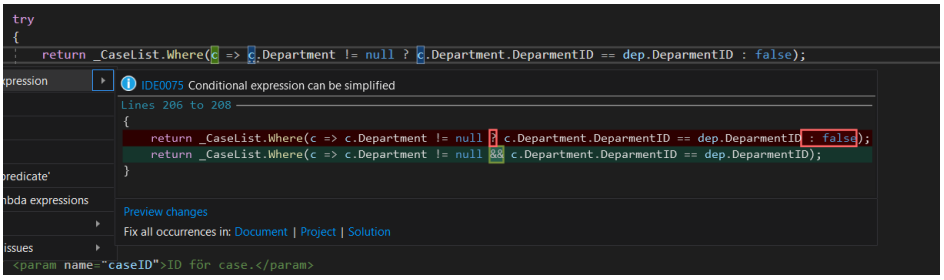
Figur 3: Meddelanden i Visual Studio från FxCop

FxCop visar att det finns flera fält i koden som bör göras om till ”readonly”, det vill säga att de inte bör vara möjligt att ändra dessa utan bara läsa det som står i dem. Detta felmeddelande förekommer på flera ställen både i filen BackEnd.cs och Database.cs. Vilka fält den föreslår visas i figur 4.



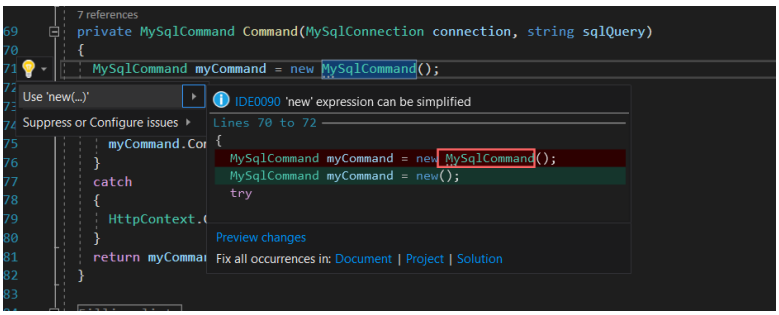
Figur 4: Fält som bör göras readonly

Därefter visar FxCop meddelanden om att vissa villkorliga uttryck kan simplifieras. Exempel på ett sådant uttryck kan ses i figur 5, där även Visual Studios föreslagna lösning visas.



Figur 5: Exempel i Visual Studio

FxCop visar även meddelande om att flera new-uttryck kan förenklas i koden. Detta gäller på flera ställen i Database.cs-filen, och ett exempel visas i figur 6. Visual Studio föreslår att instansieringen new MySqlCommand() kan ersättas med enbart new(), vilket förenklar koden på flera ställen och gör att död kod kan tas bort ur Database.cs-filen.



Figur 6: Exempel i Visual Studio.

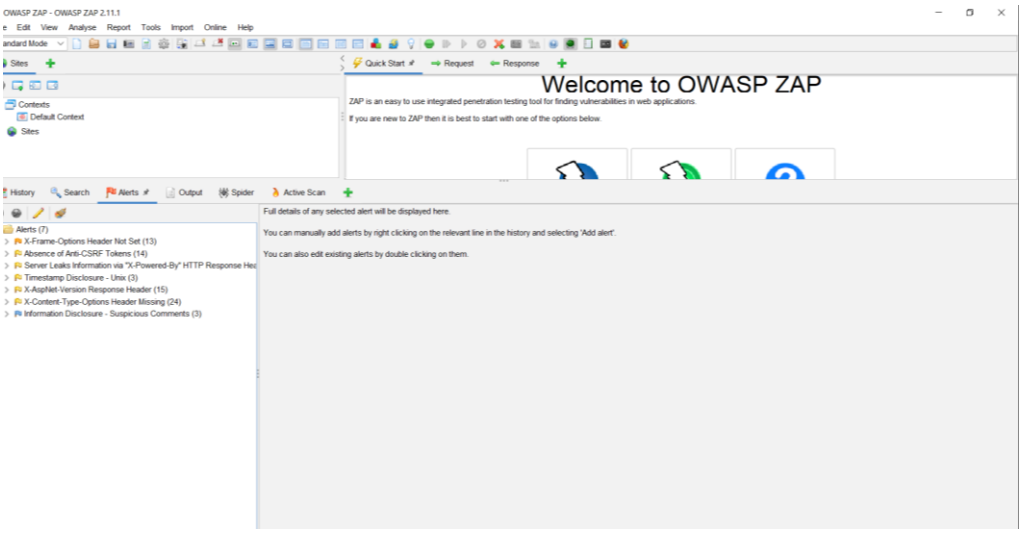
Förutom de meddelanden som FxCop visar finns det även varningar som visas i figur 7. Dessa varningar ger felkod CA2100 vilket enligt Microsoft (2016) innebär att SQL-queries för säkerhetsbrister bör granskas. Microsoft (2016) beskriver att en metod ställer in egenskapen System.Data.IDbCommand.CommandText genom att använda en sträng som är byggd från ett string-argument till metoden, och lösningen på detta är att använda sig av en parametriserad query istället. Denna varning visas på 16 olika ställen i koden, där samtliga delar felkoden CA2100 och innefattar samma beskrivning.

Figur 7: Varningar i Visual Studio.

De varningar och meddelanden som visades efter granskning med FxCop var sådant som inte hittades vid den manuella kodgranskningen. Varningarna som visades i figur 7 är troligen inte synliga för blotta ögat vid kodning, men visar på att det är bra att även granska koden med hjälp av verktyg. De meddelanden som visades hade kunnat upptäckas med blotta ögat, men det kan vara svårt att hitta denna typ av fel genom att läsa direkt i koden.

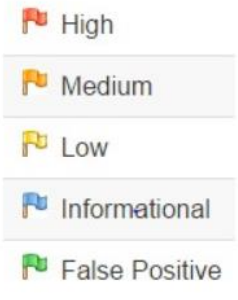
2.4 Penetrationstestning med ZAP

För att utföra penetrationstester automatiskt på applikationen användes verktyget ZAP. De fel som ZAP hittade listas i figur 8.



Figur 8: Kodgranskning med ZAP.

Enligt ZAP (2022) delas riskerna som hittas i applikationen upp i olika riskkategorier. Dessa kategorier kan ses i figur 9. Utifrån de risker som hittades i applikationen genom penetrationstester med ZAP klassas därmed en risk som medium, fem som låg risk samt en som informativ.



Figur 9: ZAP:s riskkategorier.

Det första ZAP reagerade på var ”X-Frame-Options Header Not Set”. Beskrivningen till detta är ”X-Frame-Options header is not included in the HTTP

response to protect against 'ClickJacking' attacks”. Som lösning på detta beskriver ZAP att de flesta moderna webbläsarna har stöd för X-Frame-Options http header, men det kan vara en idé att försäkra att detta är inställt på samtliga webbsidor som returneras av Småstads Kommuns applikation.

Nästa punkt som ZAP ger utslag på var ”Absence of Anti-CSRF Tokens” med följande beskrivning: ” No Anti-CSRF tokens were found in a HTML submission form”. Kortfattat handlar detta om en attack som innebär att ett offer tvingas skicka en http-förfrågan till en måldestination utan deras vetskap eller avsikt för att utföra denna handling, vilket är en risk på grund av att inga Anti-CSRF Tokens finns i applikationen. ZAP rekommenderar att Anti-CSRF Tokens implementeras i applikationen för att förhindra denna form av attack.

Därefter reagerade ZAP på ”Server Leaks Information via ’X-Powered-By’ http Response Header Field(s)” vilket beskrivs som att applikationens server läcker information via en eller flera ”’X-Powered-By’ HTTP Response Headers”. Som lösning på detta föreslår verktyget att webbservern och applikationsservern är konfigurerade för att undertrycka ”X-Powered-By” headers.

Nästa punkt är ”Timestamp Disclosure – Unix”. Beskrivningen ZAP ger om detta är att en tidsstämpel avslöjades av applikationen eller webbservern, vilket enligt ZAP kan lösas genom att en manuell bekräftelse görs om att tidsstämpeldatan inte är känslig och att data inte kan aggregeras för att avslöja exploateringsbara mönster.

”X-AspNet-Version Response Header” är nästa punkt som ZAP ger utslag på. Detta beskrivs som att servern läcker information via "X-AspNet-Version"/"X-AspNetMvc-Version’ HTTP response header field(s).” Vidare beskriver ZAP att detta kan resultera i att en attackerare kan använda denna information för att exploatera kända sårbarheter i applikationen. ZAP föreslår som lösning på detta att servern bör konfigureras så att den inte returnerar nämnda headers.

Vidare reagerar ZAP på “X-Content-Type-Options Header Missing” vilket beskrivs som följande: ”The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'” och gör att äldre versioner av webbläsarna Internet Explorer och Chrome kan utföra så kallad ”MIME-sniffing” på svarskroppen, vilket kan orsaka att denna svarskropp tolkas och visas som en annan innehållstyp än den deklarerade innehållstypen. ZAP föreslår att för att åtgärda detta bör en översyn göras på om applikationen ställer in Content-Type headern på rätt sätt samt att X-Content-Type-Options headern ställs in på nosniff för alla webbsidor.

Den sista punkten som ZAP ger utslag på är “Information Disclosure – Suspicious Comments”, vilket är beskrivet som att koden till applikationen verkar innehålla misstänkta kommentarer som kan hjälpa en angripare. ZAP föreslår att samtliga kommentarer bör tas bort som innehåller information som kan hjälpa en attackerare, samt att relaterade problem till detta bör åtgärdas.

Efter att applikationen penetrationstestats med hjälp av ZAP står det klart att felen som hittades genom manuell kodgranskning, automatiskt kodgranskning samt penetrationstester inte är detsamma för samtliga. ZAP har hittat fel i applikationen som inte kunnat hittats med hjälp av kodgranskning, varken manuell eller automatisk, och kodgranskningen har upptäckt fel som ZAP inte givit utslag på. Detta tyder på att samtliga delar är viktiga att utföra, eftersom det hjälper en med att hitta så många fel och buggar som möjligt i applikationen.

3 Rekommendationer

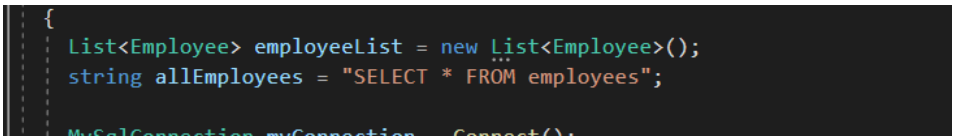
I dagsläget är applikationen uppbyggd med en gemensam databas som innehåller all data. Detta gör att om en hackare skulle ta sig in i databasen hade denne kommit åt all data som lagras av Småstads kommun. Genom att dela upp datan i olika databaser enligt figur 1 förhindras detta, då all data inte lagras på samma plats. Därför bör en ny databasfördelning göras innan applikationen driftsätts. Eftersom detta bedöms ha en allvarlig konsekvensnivå utifrån MSB:s (2009) matris är detta den absolut högsta prioriteringen för applikationen.

Penetrationstesterna som genomförts i applikationen visar bland annat på att det finns risk att information läcks genom så kallade ”X-AspNet-Version Response Header”. Detta kan innebära att en hackare kan få fram information som kan leda till att sårbarheter i applikationen hittas, och därmed använda denna information för att ta sig in i systemet och databasen. Detta kan resultera i stora konsekvenser för Småstads kommun och bör därför ses över och en omkonfigurering ska göras för att åtgärda felet. Nästa prioritering blir därför att åtgärda samtliga fel som visades utifrån penetrationstesterna i högsta möjliga grad.

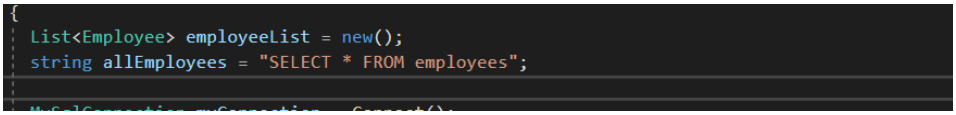
I dagsläget finns ingen lösenordspolicy implementerade i applikationen vilket kan innebära att användarkonton skapas med svaga lösenord. Detta kan resultera i att hackare kan gissa sig fram till lösenord till olika användarkonton och därmed ta sig in där de inte är behöriga. Att implementera en stark lösenordspolicy ska vara en stor prioritet för Småstads kommun då det kan förhindra intrång i systemet av obehöriga. Detta kan medföra betydande konsekvenser för Småstads kommun, och därför är detta nästa prioritering att åtgärda.

Nästa del att prioritera är att åtgärda att det går att komma åt inloggade vyer via URL oavsett vilken roll man loggar in som. Detta kan medföra att till exempel handläggare kommer åt chefssidorna, vilket de inte bör vara behöriga till att göra. En AccessDenied-sida bör även finnas där man hamnar om man försöker ta sig åt en sida via URL utan behörighet.

För att minska mängden död kod i applikationen bör även alla meddelanden som visades genom kodgranskningen med FxCop granskas och åtgärdas. FxCop visade bland annat att vissa uttryck kan förenklas, från koden i figur 10 till koden i figur 11 samt från koden i figur 12 till koden i figur 13. Eftersom dessa exempel endast var meddelanden och inte varningar eller fel enligt FxCop, blir prioriteringen lägst av rekommendationerna.

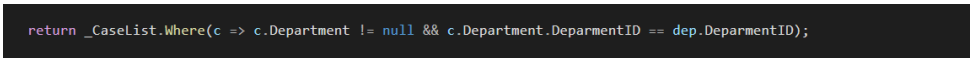


Figur 10: Kod före simplifiering



Figur 11: Kod efter simplifiering

Figur 12: Kod före simplifiering



Figur 13: Kod efter simplifiering

4 Referenslista

Internetstiftelsen (2022), *Kryptering.* [Digital] Tillgänglig: https://internetstiftelsen.se/guide/digitalt-sjalvforsvar-en-introduktion/kryptering/ [Hämtad: 2022-01-14]

Lundgren, M. (2021) Föreläsning: Granskning [2021-09-21]

Microsoft (2016) *CA2100: Review SQL queries for security vulnerabilities* [Digital] Tillgänglig: https://docs.microsoft.com/sv-se/previous-versions/visualstudio/visual-studio-2015/code-quality/ca2100-review-sql-queries-for-security-vulnerabilities?view=vs-2015&redirectedfrom=MSDN [Hämtad: 2022-01-16]

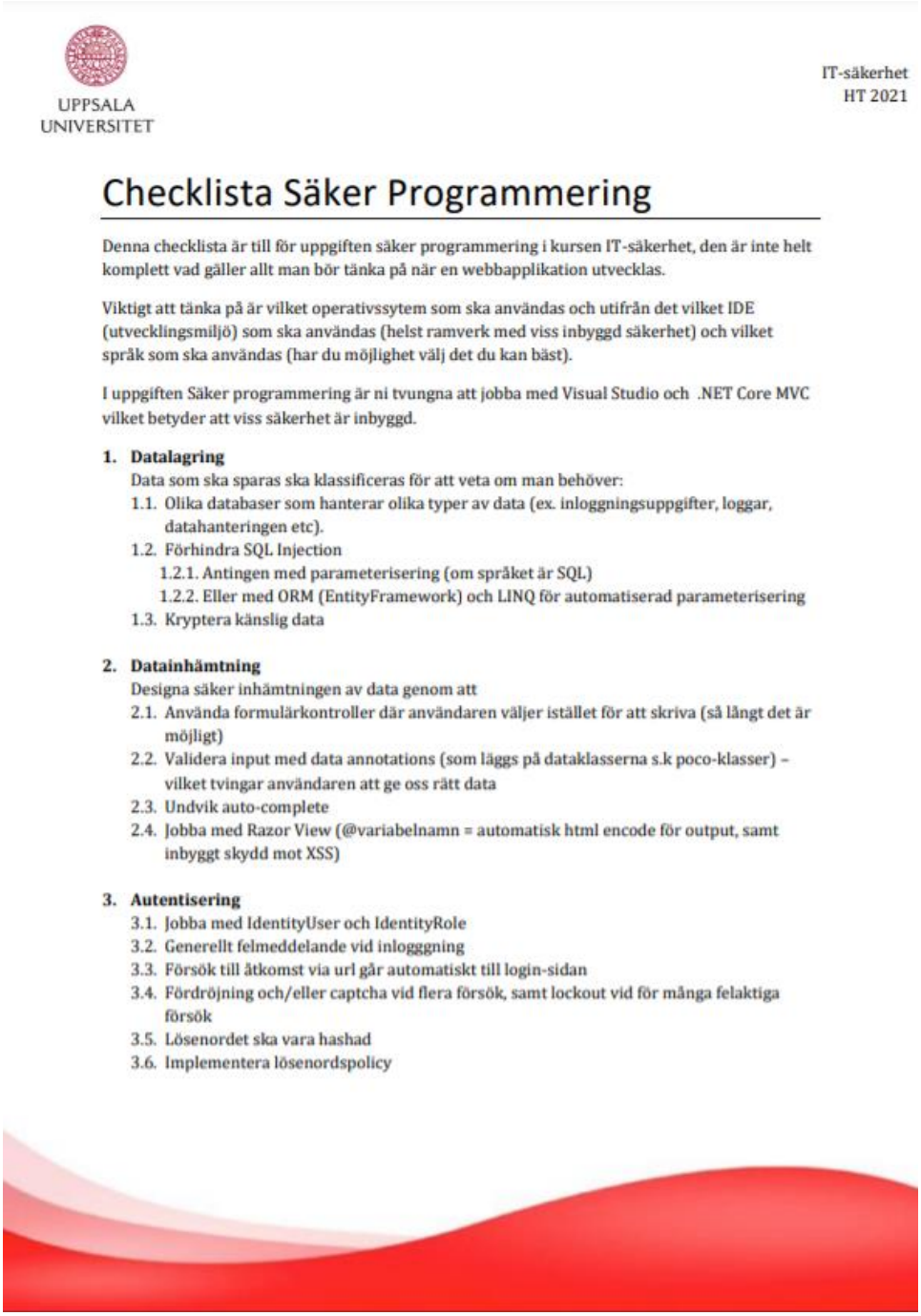
Microsoft (2021) *Password Policy.* [Digital] Tillgänglig: https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/password-policy [Hämtad: 2022-01-14]

Mozilla (2022) *How to turn off form autocompletion.* [Digital] Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/Security/Securing_your_site/Turning_off_form_autocompletion [Hämtad: 2022-01-14]

MSB (2009) *Modell för klassificering av information.* [Digital] Tillgänglig: https://www.msb.se/RibData/Filer/pdf/25602.pdf [Hämtad: 2022-01-14]

ZAP (2022) *Getting Started.* [Digital] Tillgänglig: https://www.zaproxy.org/getting-started/ [Hämtad: 2022-01-16]

5 Bilaga 1 - checklista



UPPSALA UNIVERSITET

IT-säkerhet
HT 2021

Checklista Säker Programmering

Denna checklista 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 operativssytem 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.

1. **Datalagring**
   Data som ska sparas 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 SQL Injection
      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

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)

3. **Autentisering**
   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
   3.6. Implementera lösenordspolicy

UPPSALA
UNIVERSITET

IT-säkerhet
HT 2021

4. **Auktorisering**
   - 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. **Skydd mod CSRF**
   - 5.1. Formulär innanför form-element med method="post"
   - 5.2. asp-antiforgery attribut i form-elementet
   - 5.3. [ValidateAntiForgeryToken] på action-method
   - 5.4. Ta bort token cookie vid utloggning

6. **Felhantering**
   - 6.1. Validera att parametrar har innehåll
   - 6.2. Viktig kod ska ligga i try-catch
   - 6.3. Hantera exceptions korrekt
     - 6.3.1. Vad ska hända – felmeddelande/defaultinställning
     - 6.3.2. Generellt felmeddelande till användaren
     - 6.3.3. Logga stacktrace enligt beslut

7. **Loggning**
   - 7.1. Logga viktiga saker (inloggningar, vad som gör, felmeddelanden)
   - 7.2. Inkludera tid, ip-adress, vem och vad som gjordes
   - 7.3. Kryptera loggningarna
   - 7.4. Förvara loggningarna i annan databas (och annan server)

8. **Inför driftsättning**
   - 8.1. Kommentera vad som ska bort eller förändras innan driftsättningen