**IT-Sec sårbarhet 1**

CWE-89: SQL Injection

Den presenterade koden representerar en funktion inom en webbapplikation som tillhandahåller kontouppgifter baserat på användarnamn. Funktionen visar en allvarlig SQL-injektionssårbarhet.

En bild som visar text, skärmbild, programvara, Multimedieprogram

Automatiskt genererad beskrivning

**1. Sårbarhetsbeskrivning**

Koden konstruerar en SQL-fråga genom att lägga till det användarnamn som direkt tas från en HTTP-begäran till en SQL-sträng. Eftersom inga säkerhetsåtgärder eller inputvalideringar används kan en illvillig användare skicka in skadlig kod som kan manipulera den ursprungliga SQL-frågan.

**2. Hur sårbarheten kan utnyttjas**

En angripare kan skicka en speciellt utformad sträng som när den läggs till i SQL-strängen kan leda till en annan än den avsedda frågan.

**3. Potentiella risker**

* Databasmanipulation: Genom att skicka in skadlig SQL-kod kan en angripare utföra en rad skadliga handlingar som att radera tabeller, ändra data eller till och med lägga till ny data.
* Dataintrång: Angriparen kan hämta känslig information om andra användare, vilket kan leda till integritetsintrång.
* Förlust av data: Om en angripare väljer att radera data, kan det leda till betydande förluster för företaget eller användarna.
* Systemkompromiss: Beroende på databasens konfiguration och rättigheter kan en angripare potentiellt använda SQL-injektion för att få ytterligare rättigheter eller tillgång till andra delar av systemet.

**4. Rekommenderade åtgärder**

* Använd parameteriserade frågor eller ORM (Object-Relational Mapping) bibliotek för att interagera med databasen istället för att skapa SQL-frågor från direkt användarinmatning.
* Implementera ordentlig inputvalidering för att säkerställa att inkommande data är i det förväntade formatet och inte innehåller skadliga tecken eller kommandon.
* Begränsa databasanvändarens rättigheter så att de inte kan utföra skadliga handlingar som att radera tabeller.

**IT-Sec sårbarhet 2**

CWE-862: Missing Authorization

En bild som visar text, skärmbild, programvara, Teckensnitt

Automatiskt genererad beskrivningEn bild som visar text, skärmbild, programvara, Teckensnitt

Automatiskt genererad beskrivningEn bild som visar text, skärmbild, programvara, Teckensnitt

Automatiskt genererad beskrivning

**1. Sårbarhetsbeskrivning**

Webbapplikationen saknar auktorisering, vilket innebär att alla API:er är öppna och kan nås av alla, oavsett om de har behörighet eller inte. Utan en auktoriseringsmekanism kan vilken användare som helst komma åt alla resurser, vilket exponerar känslig data och kritiska funktioner för potentiella hot.

**2. Hur sårbarheten kan utnyttjas**

* Eftersom det inte finns någon auktoriseringskontroll, kan en obehörig användare:
* Navigera till valfritt API-endpoint direkt genom webbläsaren eller andra verktyg.
* Åtkomst och hämta data de inte borde ha tillgång till.
* Utföra operationer (som att lägga till, ändra eller radera data) som de inte borde ha rättigheter att göra.

**3. Potentiella risker**

* Dataintrång: Obehöriga användare kan komma åt känslig information, vilket kan leda till integritetsintrång.
* Manipulation av data: Utan lämplig auktorisering kan en obehörig användare ändra eller radera viktig data, vilket kan orsaka felaktig information att spridas eller systemfel.
* Överträdelse av dataskyddsregler: Bristen på auktorisering kan leda till att företaget överträder dataskyddslagar, vilket kan resultera i böter eller andra påföljder.
* Missbruk av funktioner: Utan lämpliga auktoriseringskontroller kan funktioner som är avsedda för specifika användare eller roller missbrukas, vilket kan leda till oavsiktlig eller illvillig skada.

**4. Rekommenderade åtgärder**

* Implementera auktorisering: Använd bibliotek eller verktyg som Spring Security för att tillföra robust auktoriseringskontroll på alla API:er.
* Rollbaserad tillgång: Inför rollbaserad auktorisering där olika roller (t.ex. admin, användare) har olika nivåer av åtkomst och rättigheter.
* Token-baserad autentisering: Överväg att använda en token-baserad autentisering, som JWT (JSON Web Token), för att säkra dina API:er och verifiera användarens identitet samt rättigheter.
* API-nycklar: För tredjepartsintegreringar eller externa tjänster, använd API-nycklar för att begränsa och kontrollera tillgången till dina API:er.
* Granskning och loggning: Inför granskning och loggning av alla åtkomstförsök och operationer på systemet för att snabbt kunna identifiera och svara på obehörig åtkomst eller aktivitet

**Sårbarhet 1 åtgärdat**

CWE-89: SQL Injection

**En bild som visar text, skärmbild, Teckensnitt

Automatiskt genererad beskrivning**

I den ursprungliga koden konstruerades SQL-frågan genom att direkt använda username-parametern in i frågan. Detta öppnar upp för SQL-injektionssårbarheter, där en illvillig användare kan infoga skadlig SQL-kod som del av username för att manipulera eller skada databasen.

I den förbättrade koden används accountRepository.findByUserName(username), som använder JPA. Detta verktyg skapar parameteriserade SQL-frågor, vilket skyddar mot SQL-injektion eftersom varje input behandlas som data och inte som en del av SQL-koden.

**Sårbarhet 2 åtgärdat**

CWE-862: Missing Authorization

**En bild som visar text, programvara, Multimedieprogram, Teckensnitt

Automatiskt genererad beskrivning**

* **Auktorisering:** Genom att kräva att vissa endpoints (i detta fall "/account" och allt under den) bara kan nås av autentiserade användare, ser vi till att bara de som har rätt behörighet kan komma åt dessa resurser.
* **Autentisering:** Formulärbaserad inloggning säkerställer att användare måste ange sina inloggningsuppgifter (vanligtvis användarnamn och lösenord) för att få åtkomst till skyddade delar av applikationen.
* **Säkerhet mot obehörig åtkomst:** Eftersom oautentiserade förfrågningar till "/account" och underordnade URL:er omdirigeras till inloggningsformuläret, kan obehöriga användare inte enkelt navigera till dessa URL:er och se eller manipulera data. De skulle först behöva känna till en giltig användares inloggningsuppgifter.
* **Skydd mot direkta URL-åtkomster:** Även om någon vet direkt-URL:en, skulle de inte kunna åtkomst den utan att först logga in.

En bild som visar text, skärmbild, Teckensnitt, nummer

Automatiskt genererad beskrivning