**Week-02 A2-Broken Authentication, A5-Broken Access Control**

**Agenda:**

1. Forklar hvad forskellen mellem A2 og A5 er.
2. A2-svagheder:
   1. Brute force attack, og vis et praktisk eksempel (tilføj en timer).
   2. Credential stuffing og forklar om:
      1. CAPTCHA.
      2. Multifactor godkendelse (f.eks. nem-id eller SMS-tjeneste)
      3. Ip-blacklisting
   3. Svage adgangskoder, og vis praktisk eksempel.
3. A5-svagheder:
   1. Uautoriseret adgang til ressourcer og vis praktisk eksempel (@rolesallowed)
   2. Undgå adgangskontrol, vis praktisk eksempel (JDBCRealm)

**Explain about OWASP-A2 and explain a number of the problems that would leave your application vulnerable to A2-problems. For each of the problems explained, explain how this problem could have been prevented.**

A2 er en måde hvorpå en hacker kan få adgang til brugere og informationer fra en applikation, som de ikke burde.

Der er muligvis en række svagheder, hvis en applikation:

* + tillader automatiserede angreb, såsom credential stuffing, hvor en angriber har en liste over gyldige brugernavne og adgangskode fra en anden platform.
  + tillader brute force, eller andre automatiske angreb.
    - **Løsning**: Begræns eller i stigende grad forsinke mislykkede login forsøg. Log alle fejl og alarmer administrator, når et angreb detekteres.
    - **Løsning:** Sørg for at sende samme fejl meddelelser med alle API kald, både ved registrering, loginforsøg og gendannelse af loginoplysninger, for at hærde kontoen mod angreb. (f.eks. meddelelse: ”e-mail eller adgangskode er forkert”).
  + tillader standard, svage eller velkendte adgangskoder f.eks. ”1234” eller ”admin”.
    - **Løsning**: Implementer en kontrol mod svage adgangskoder, f.eks. ved at sammenligne nye eller ændrede adgangskoder mod en liste af velkendte svage adgangskoder.
  + bruger svage eller ineffektive gendannelse- og glemt- adgangskode processer, såsom videns baserede svar, som ikke kan gøres sikker.
    - **Løsning**: Juster adgangskodens længde, kompleksiteten og rotationspolitikken (tilfældigt spørgsmål), jævnfør NIST 800-63 B’s retningslinjer i afsnit 5.1.1 ”memorized secrets”.
    - **Løsning**: I stedet for videns baserede svar, tilsendes et link, til at lave en ny adgangskode.
  + bruger almindelig tekst, krypteret eller svagt hashede adgangskoder (se A3).
    - **Løsning**: Brug en langsom, stærk, saltet og fleksibel (udvikler vælger antal iterationer) hashing algoritme, såsom Argon2, Bcrypt, Scrypt og PBKDF2.
  + har manglende eller ineffektiv godkendelse af flere faktorer.
    - **Løsning**: Implementer multi-faktor godkendelse, hvor det er muligt (f.eks. ved login på nem-id, hvor man både skal logge ind og derefter indtaste en kode forbundet til login). Dette er også med til at forhindre credential stuffing, brute force og stolen credential reuse attacks.
  + viser sessions id i URL.
    - **Løsning**: De skal ikke gemmes i url, men skal i stedet gemmes sikkert og gøres ugyldige efter log ud. Sessions id skal have en timeout.
  + ikke roterer sessions id efter login.
    - **Løsning**: Brug en sikker indbygget sessionsmanager (på serveren), der generer en ny tilfældig sessions id, med høj entropi (udtryk for den samlede uorden).
  + ikke ugyldiggøre korrekte sessions id’er. Bruger sessions eller tokens (især SSO-tokens) er stadig gyldigt, efter log ud eller inaktivitet.
    - **Løsning**: De skal gøres ugyldige efter log ud. Sessions id skal have en timeout.

**OBS.** Aldrig deploy med default brugere, specielt ikke til en admin-bruger.

**Explain the idea behind the term Credential Stuffing**

Credential stuffing er en type af hacking, hvor en hacker har en stjålen liste af bruger informationer, typisk en liste af brugernavne/emails og deres tilknyttede adgangskoder.

Hackeren kører automatisk alle kombinationerne op mod serveren for at teste om nogle af kombinationerne virker til den pågældende platform.

**Demonstrate how you, in practice, have, or could, remove some of the vulnerabilities listed above**

**Eksempel på at undgå svage adgangskoder:**

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

**Eksempel på at undgå brute force angreb:**

Nedenstående er et eksempel på en måde at forsinke en bruger, der har tastet forkerte oplysninger ved login.

Disse funktioner bruges i klassen User, i forbindelse med den nye attribut:

**Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse**

Herefter bruges de ovenstående funktioner, i **verifyPassword** til at sætte et nyt timestamp på User, hvis det er under 30 sekunder siden de sidst fejlede i at logge ind eller hvis adgangskoden er skrevet forkert.

**Et billede, der indeholder skærmbillede, fugl

Automatisk genereret beskrivelse**

I userFacaden opdateres brugeren med sit nye timestamp i databasen, hvis verificering af brugeren fejler.

**Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse**

**Explain about OWASP-A5, and explain a number of the problems that would leave your application vulnerable to A5-problems. For each of the problems explained, explain how this problem could have been prevented**

A5 er en måde, hvorpå en bruger der er logget ind, skaffer sig rettigheder til en given ressource, som man ikke burde.

Der er muligvis en række svagheder, hvis en applikation:

* + URL til en admin siden (f.eks. www.webside/admin) er tilgængelig selvom man ikke er admin.
    - **Løsning:** Sikre ved at lave et tjek på brugerens rettigheder før de får adgang til ressourcen.
  + eller en bruger formår at gå udenom adgangskontrol, ved at ændre i url’en, den interne applikations tilstand eller HTML siden, eller blot ved brug af API-angrebs værktøjer.
    - **Løsning**: benægt adgang som standard, med undtagelse af offentlige ressourcer.
  + Tillader at en primær nøgle (f.eks. brugernavn eller e-mail) bliver tilknyttet en anden bruger, som gør det muligt at se og ændre en anden brugers konto.
    - **Løsning**: Model access control, skal håndhæve ejerskab over brugerens information, fremfor at kunne oprette, ændre eller slette enhver anden brugers informationer.
  + forhøjer privilegier. At kunne fungere som en bruger, uden at være logget ind eller fungere som en administrator, når man er logget ind som bruger.
    - **Løsning**: Logging.
  + Metadatamanipulation, såsom at afspille eller manipulere med et JWT, en cookie eller et skjult felt, til at hæve privilegier eller misbruge JWT ugyldighed.
    - **Løsning**: JWT tokens skal være ugyldige hos serveren efter log ud.
  + CORS fejlkonfiguration tillader uautoriseret api adgang.
    - **Løsning**:
  + tvinger browsing til godkendte sider, som en ikke godkendt bruger, eller til privilegerede sider som en standard bruger. Adgang til API med manglende adgangskontroller, til POST, PUT og DELETE.
    - **Løsning**:

**Demonstrate how you, in practice, have removed some of the vulnerabilities listed above**

En måde at forhindre uautoriseret adgang, er at f.eks. at sætte @rolesallowed på de forskellige API. Det er en annotation der kommer fra security biblioteket.

**Et billede, der indeholder fugl

Automatisk genereret beskrivelse**

Nedenstående ses et eksempel på at forhindre brugere med andre roller end lige ”admin” i at tilføje en movie til databasen.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

**Demonstrate at least one practical example of a A5 Vulnerability, and show how you have solved the problem (removed the vulnerability)**

Start med at logge ind som dig selv (i det her tilfælde: ”*a@b.dk*” og adgangskode: ”*test* ”)

Herfra er det muligt at få a’s personlige data. Ved at undersøge siden, således:

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

På ovenstående billede, kan man se værdien for brugerens e-mail, og ved at ændre denne til en anden e-mail, er det muligt at få en anden brugers personlige data (se nedenstående), selvom man stadig er logget ind som sig selv.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

En måde at beskytte en ressource mod uautoriseret adgang, er at implementere Java EE/Tomcats build-in security mekanisme, som består af en JDBCRealm, i kombination med Tomcats build-in basic http authentication support. Denne sikkerheds mekanisme implementeres ved at tilføje en Realm Declaration i context.xml filen, samt tilføje en JDBC driver til Tomcat, eftersom det er denne der håndtere sikkerheden.

Realm betyder at tomcat sikre sig mod at brugere med uautoriseret adgang til en ressource, ikke kan få adgang.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse