Secure Passwords

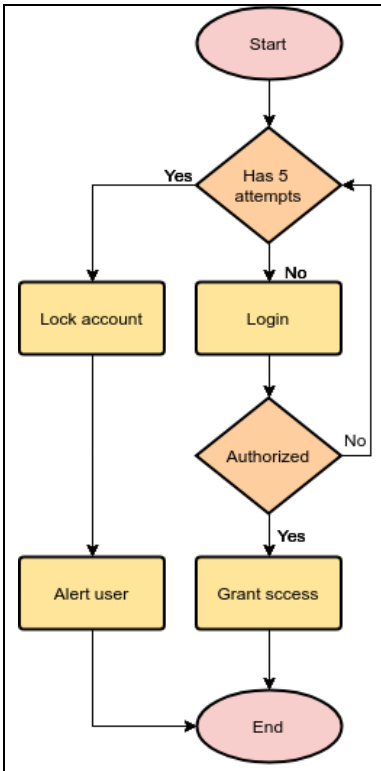
Kryptering

Sikker lagring af adgangskoder

Hashing-algoritmer repræsenterer en kategori af en-vejs funktioner, der omdanner en vilkårlig mængde inputdata til et unikt "fingeraftryk" med en bestemt fast længde. Disse funktioner er uigenkaldelige, hvilket betyder, at det ikke er muligt at gendanne det originale input fra dets hash-værdi. Mindre ændringer i inputdataene resulterer i markante ændringer i den resulterende hash, en egenskab der er kendt som "avalanche-effekten." Denne egenskab gør hashing særlig velegnet til beskyttelse af følsomme data såsom adgangskoder, idet hashede værdier opbevares i stedet for de faktiske adgangskoder, hvilket minimerer risikoen ved eventuelle sikkerhedsbrud.

I en kontekst af autentificering og sikker lagring af brugeroplysninger kan en applikation udvikles til at opbevare login-information over tid i en sikker database. Dette kan indebære anvendelse af både hashing og eventuelle yderligere sikkerhedsmekanismer, såsom salting.

Overordnet set kan login-processen repræsenteres gennem et rutediagram, hvor forskellige trin i autentificeringsprocessen kan kortlægges og udvides i overensstemmelse med specifikke krav og protokoller, der er angivet til et givnet system. Denne metode til repræsentation fremhæver processens strukturerede natur og muliggør en systematisk tilgang til design og implementering.



Den generelle procedure for kontoregistrering og autentificering indenfor et hash-baseret sikkerhedssystem kan artikuleres som følger:

1) En ny bruger initierer oprettelsen af en konto, som involverer definition af legitimationsoplysninger såsom brugernavn og adgangskode.
2) Ved registreringen bliver brugerens adgangskode behandlet gennem en hashing-algoritme og den resulterende hash-værdi lagres i databasen. Det er væsentligt at understrege, at adgangskoden på intet tidspunkt lagres i klartekst på harddisken, hvilket fremmer systemets sikkerhed.
3) Når en bruger forsøger at autentificere sig, genereres en hash-værdi for den indtastede adgangskode og sammenlignes med den tilsvarende hash-værdi, der er lagret i databasen, korresponderende til den pågældende brugers konto.
4) I tilfælde af overensstemmelse mellem de sammenlignede hash-værdier, bliver brugeren autentificeret og får dermed adgang til systemet. Hvis der ikke er overensstemmelse, bliver brugeren informeret om, at de indtastede legitimationsoplysninger er ugyldige.
5) Trin 3 og 4 repræsenterer en kontinuerlig proces, der itereres hver gang en bruger eller en enhed forsøger at logge ind på en konto, hvilket sikrer vedvarende integritet og fortrolighed i autentificeringsprocessen.

Gemme Adgangskode:

1. Generer et langt, tilfældigt salt.
2. Kombiner salt og adgangskode, og hash ved hjælp af en standard algoritme.
3. Lagre salt og hash i brugerens databasepost.

Validering af Adgangskode:

1. Indhent brugerens salt og hash fra databasen.
2. Tilføj salt til indtastet adgangskode og hash med samme funktion.
3. Sammenlign hash-værdier; hvis de matcher, er adgangskoden korrekt, ellers forkert.

Yderligere vejledning kan findes på link til video.

Udvidelse implementering af Password-Based Key Derivation ved Brug af Rfc2898DeriveBytes

Mål: I denne opgave vil du udvide din eksisterende applikation til at anvende en mere sikker metode til adgangskodehåndtering ved at benytte Password-Based Key Derivation Function 2 (PBKDF2). Dette vil blive implementeret ved hjælp af Rfc2898DeriveBytes-klassen i .NET.

Delopgaver:

1. **Forståelse af PBKDF2:** Start med at forstå, hvordan PBKDF2 fungerer, og hvordan den bruger en pseudo-tilfældig talgenerator baseret på HMACSHA1.
2. **Generering af salt og nøgle:** Anvend Rfc2898DeriveBytes til at tage et kodeord, et salt og et antal iterationer for at generere nøgler. Du skal bruge GetBytes-metoden til dette formål.
3. **Konvertering til Base64:** Implementer funktionalitet til at konvertere password, salt og nøglebytes til en base64-streng. Dette vil hjælpe når disse værdier lageres i en database.
4. **Læsning fra database:** Skriv kode til at vende tilbage fra base64-streng til byte, når du loader værdier fra databasen.
5. **Validering:** Sørg for, at din implementering korrekt validerer brugeradgangskoder mod de gemte hash-værdier i databasen.
6. **Test og Dokumentation:** Test din implementering for at sikre, at den fungerer som forventet, og dokumenter din kode grundigt.
7. **(Valgfrit) Se en Referencevideo:** For yderligere forståelse, kan du overveje at se denne videoguide.