Inrichting Centrale bank

Merijn Plagge
0941200

23 mei 2018

Samenvatting

Voor project 4 moesten wij voor het individuele onderdeel een advies geven over de inrichting van de *Centrale Bank*. Aan het eind van week 5 moeten een van de adviezen van de studenten worden gekozen en uitgewerkt de rest van de loop van het project. Wij moesten tijdens de loop van het project letten op vijf onderdelen: *beheren, analyseren, adviseren ontwerpen en realiseren.*

Inhoudsopgave

1 Communicatie

1.1 Server of servers?

Er zijn kort gezegd twee opties om de server(s) op te zetten. Of wel één centrale server, or meerdere losse servers die met elkaar praten. Hieronder zal ik uitleggen welke ik gekozen heb en waarom.

1.1.1 Centrale server

Dit is het makkelijkste systeem. Je hoeft namelijk maar ëën server op te zetten.

1.1.2 Verbonden servers

Het veiligste idee is om meerdere servers te hebben. Als er een van de servers uitvalt, dan zal het banksysteem nog steeds werken.

1.2 protocol

1.2.1 MQTT of MQ?

MQ, want ëën client ëën consumer.
Hoe?

2 Database

Wat geincript? Waar geplaats?

2.1 Gevoelige gegevens

2.2 Analyze en advies beveiliging lokale opslag

Hieronder zijn verschillende manieren te vinden hoe inloggegevens en andere gevoelige informatie op kunnen worden geslagen. Ik heb het hier over de voor en nadelen van deze manieren, en ook geef ik hier advies of het van toepassing is bij de bank.

2.2.1 Plain text

Dit is de meest onveilige manier om wachtwoorden, pincodes, RFID-codes op te slaan. Stel je database raakt op de een of andere manier blootgesteld aan onbevoegden personen, dan kan je makkelijk pincodes en RFID-codes aflezen. Het is wel de makkelijkste manier om inloggegevens op te slaan. Je kan namelijk makkelijk alle gegevens opvragen zonder moeite.

Maar sinds het hier om de beveiliging van een bank gaat is dit niet veilig genoeg. Tijdens het ontwikkelen van de centrale bank zou het wel tijdelijk kunnen worden toegepast om onderdelen te testen, en zou daarna vervangen moeten worden door een geëncrypt systeem.

2.2.2 Encrypt de hele database

Een andere veiligere optie is om de database lokaal te versleutelen. Stel de gebruiker stuurt een pincode naar de server, dan unencrypt de server die pincode in de database, en vergelijkt de pincodes. Dit is al een stuk veiliger dan een plain text wachtwoord, maar het heeft nog steeds een nadeel. Als een hacker de key om de database te ontsluitelen te pakken krijgt, dan kan hij alle wachtwoorden achterhalen. Ook zullen dezelfde wachtwoorden zichtbaar zijn, omdat de geëncripte wachtwoorden er hetzelfde uitzien.

Mijn advies is om ook dit systeem niet te gebruiken, al is het een stuk veiliger dan plain text. Het zou voor een schoolprojectje goed genoeg zijn, sinds zelfs grote bedrijven deze fouten nog maken, maar het is beter om de later beschreven manieren te gebruiken.

2.2.3 Hashing

Naast de hele database te versleutelen kan je ook de individuele pincodes and RFID codes hashen. Het verschil tussen een hash en een ge encrypte string is dat je een hash niet terug kunt zetten naar het origineel met een key. Een hash kan je wel zelf terugrekenen. Het moet dus een langzame hashinging methode zijn omdat je anders als hacker snel wachtwoorden zou kunnen raden. Goede hashing aloritmes hiervoor zouden kunnen zijn: *scrypt, bcrypt of PBKDF2.* Md5 is hier bijvoorbeeld niet voor geschikt omdat hier hashes snel te berekenen zijn. Het moment dat de gebruiker bijvoorbeeld een pincode invoert, zal deze gehashed worden. Zodra deze bij de database aankomt zullen de hashes worden vergeleken. Dit is veiliger dan de hele database te encrypten, omdat de gegevens niet meer geunencrypt hoeven te worden. Je hebt jammer genoeg nog steeds het probleem dat je versleutende data met elkaar kunt vergelijken. Gebruikers met hetzelfde wachtwoord zullen ook dezelfde hash hebben. Je hebt ook zogenaamde rainbow tables. Dit zijn tabellen waar mensen de hashes al voor je hebben uitgerekend. Zo kan je ook snel door de meer zware hashing aloritmes heen gaan, en zijn deze ook niet veilig.

$$\text{pincode} \xrightarrow{hash} \text{hashed pincode}$$

2.2.4 Hashing and Salting

De beveiligingsproblemen die overblijven bij hashen zijn op te lossen, waarna geen beveiligingsproblemen overblijven met de database. Ze zijn op te lossen door midden van *Salting.* Ik raad dan ook aan om dit systeem te gebruiken voor de beveiliging. Dit is een voor iedere gebruiker random gegenereerde string die je opslaat in de database. Het idee is om gevoelige gegevens, als ze worden ingevoerd aan deze string to koppelen, waarna je ze door een hashing algoritme stuurt. Het resultaat sla je op in je database. Als de gebruiker een pincode invoert dan zal eerst de pincode geïncrypt moeten worden. Dit moet in de Arduino gebeuren sinds data tussen de computer en Arduino kan worden afgeluisterd. Hier wordt geen hashing gebruikt omdat je anders weet hoe de server wachtwoorden vergelijkt in de database. Hierna stuurt de Arduino deze string via de computer naar de server. Op de server zal de binnenkomende pincode gedecrypt worden. Hierna zal de salt worden toegevoegd, en de hash worden toegevoegd. Dan wordt hash vergeleken worden met de hash in de database.

Hieronder is visueel weergegeven wat mijn model is om een pincode veilig te vergelijken naar de database te krijgen. De laatste hash zal vergeleken worden met de hash in de database.

$$\text{pin} \xrightarrow{\textbf{Arduino } encrytion} \text{encryptedPin} \xrightarrow{\textbf{server } decryption} \text{pin} \xrightarrow{salting} \text{pin+salt} \xrightarrow{hashing} \text{hash}$$

3 Conclusie

Dit en dat moet gebeuren.