BLOG | Kim Tiedemann

EMNER *Offentlig it, Sikkerhedshuller, Webapplikationer*

OMG - de sender mit 5f4dcc3b5aa765d61d8327deb882cf99 i klar tekst

Af Kim Bjørn Tiedemann 28. juli 2014 kl. 10:06

Jeg havde egentligt ikke forestillet mig, at mit tredje blogindlæg også skulle handle om sikkerhed, men jeg stødte på en sjov implementering, som jeg må dele.

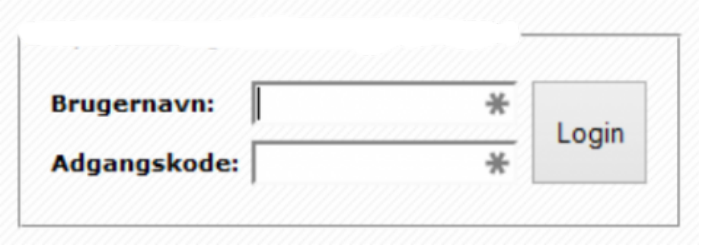
Dette er endnu et offentligt system, som er meget brugt og som tilbyder personificeret indhold og følsomme oplysninger til autentificerede brugere.

Vi må ikke sende adgangskoden over en ukrypteret kommunikationslinje

Det er tydeligt, at der er nogen, der har sagt "vi må ikke sende adgangskoder over en ukrypteret linje", men i stedet for at lave https på sitet, så har man i stedet forsøgt at opnå sikkerhed på anden vis.

Hashing client-side

Brugeren bliver altså præsenteret for en login formular over ganske almindelig http. Brugeren kan indtaste brugernavn og adgangskode:



Html form'en postes tilbage til http://.../Login.asp, og under normale omstændigheder vil adgangskoden derfor også postes tilbage i klar tekst.

Men på sitets javascript findes en lille funktion, som kaldes på html form'ens onSubmit.

```
function krypterKode() {
    document.form.md5kode.value =
          hex_md5(document.form.adgangskode.value);
    document.form.adgangskode.value = '';
}
```

Bemærk at funktionen kaldes kryptér kode :-)

Inden form'en submittes tilbage til login siden, så "krypteres" adgangskoden med MD5, hvorefter MD5 værdien puttes ind i et hidden felt og adgangskode feltet nulstilles.

Hvad er der galt med denne metode?

Der er mange ting galt med implementeringen.

For det første har man ikke sikret noget som helst. Eve (den onde hacker) har slet ikke brug for Alice's (den rare og uvidende bruger) adgangskode i klar tekst. MD5 hash værdien er nok til at logge ind, og den sendes i klar tekst uden brug af https.

For det andet er det dårlig praksis at bruge client side hashing i forbindelse med adgangskoder, da der bør være et server-side hash check også. Hvis hashing kun sker client-side og brugerdatabasen for sitet kompromiteres, så er det altså muligt for hackeren, at logge ind som alle brugere, da hackeren jo principielt har alle adgangskoder! Det svarer til at gemme adgangskoden i klar tekst i databasen.

For det tredje så kan vi med ret stor sandsynlighed antage, at sitet gemmer MD5 hashværdien i brugerdatabasen. Skulle denne database blive kompromiteret på et tidspunkt, er det en smal sag ved brug af lookuptabeller eller brute-force at få brugernes adgangskoder ud i klar tekst. Desværre ved vi, at brugere genbruger samme brugernavn og adgangskode på tværs af sites (bare spørg din omgangskreds), og derfor vil de kompromiterede informationer med stor sandsynlighed kunne misbruges på andre sites. Sitet burde benytte en mere moderne hash algoritme som SHA-2 (http://en.wikipedia.org/wiki/Sha-2) samt salt'e (http://en.wikipedia.org/wiki/Salt_(cryptography)) adgangskoden pr bruger. Hvis man vil gå all-in, så kunne de bruge en langsom kryptografisk algoritme som PBKDF2 (http://en.wikipedia.org/wiki/PBKDF2) eller bcrypt (http://en.wikipedia.org/wiki/Bcrypt) .

For lige at opsummere:

- Https everywhere

- Benyt server-side hashing
- Benyt moderne hash algorimer og husk rigeligt salt
- Benyt langsomme kryptografiske algoritmer som PBKDF2 eller bcrypt, så brute-force bliver umulig i praksis.

Hvis man nogensinde mangler argumenter for, hvorfor adgangskoder som minimum skal hashes og saltes, så er her en god FAQ (http://plaintextoffenders.com/faq/devs) .

Ps 5f4dcc3b5aa765d61d8327deb882cf99 er MD5 hash for strengen "password"

Om Kim Bjørn Tiedemann
Kim er udviklingschef hos Schultz og arbejder også som løsningsarkitekt og scrum coach. Han har arbejdet med agil udvikling de seneste 5 år, teknologi er hans store passion, og i sin fritid koder han på en Windows 8-app. Han er uddannet i datalogi fra AAU.
Følg @kimtiede 96 følgere