Password systemer til husbehov

Kort gennemgang af database-funktioner til adgangskode-registering med reversibel og irreversibel kryptering.

Skrevet den **03. Feb 2009** af **trer** I kategorien **Databaser / MS SQL** | ★★★☆☆

Indledning
Fint, nu er du kommet så vidt at du valgt at "betale" for at læse artiklen - så har du sikkert også opdaget, at nævnte artikel ligger i kategorien "DATABASER - MSSQL" (Det kan man f.eks. læse lige over artiklens titel) og derfor kan forventes at handle disse ting?

Hvis du har været opmærksom på dette, så er det perfekt! Og hvis ikke - så er det bare ærgerligt, men jeg takker så ekstra meget for de 5 points :-)

Artiklen her giver en kort vejledning i hvordan man kan opbygge sit eget system til validering af brugeres adgangskoder (irreversibel kryptering) eller lave registering af adgangskoder til andre systemer (reversibel kryptering).

I øvrigt - uanset om man bruger reversibel eller irreversibel kryptering så vil den altid kunne brydes ved at "brute force" angreb. Spørgsmålet er bare hvor længe det vil tage.

Historik & Opdateringer
16 feb 2004 : Smårettelser, mest ord og tegnsætning

Adgangskoder med irreversibel kryptering
Når man laver et loginsystem til et system vil det ofte være fordelagtigt at kryptere adgangskoder med en algoritme der ikke tillader dekryptering. Dvs. når adgangskoden først er omformet fra klar tekst til en forvansket streng, så kan den ikke føres tilbage.

Men hvordan bruger man så adgangskoden når den først er krypteret? Simpelt - den adgangskode brugeren indtaster ved efterfølgende logons krypteres også, og så er det de to krypterede strenge der skal sammenlignes.

Selvfølgelig kan en smart person så blot kryptere f.eks. en ordbog og et (stort) antal tilfældige strenge, sammenholde den med de krypterede adgangskoder - og vupti - samtlige adgangskoder er brudt.

For at undgå den slags tilføjer man et *seed*, en unik værdi, til hver adgangskode før man krypterer den. Dette *seed* forhindrer så, at to adgangskoder får samme krypterede værdi - og dermed kan ovenstående ikke ske.

Til at generere den krypterede adgangskode kan forskellige algoritmer bruges. Det primære for algoritmen er, at den giver en høj sikkerhed for, at to forskellige strenge ikke giver samme krypterede værdi.

I dag ser man typisk algoritmen MD5 brugt - men også paritetskontroller og CRC vil fungere, blot de baseres på tilstrækkelig mange bits.

MD5 vil, uanset antal tegn i input give en 32 tegns streng ud, det gør den specielt anvendelig da den så også skjuler adgangskodens længde effektivt. Smider man f.eks. biblen ind, får man stadig kun 32 tegn ud.

Men MD5 kan dog stadig brydes; Når angriberen på et tidspunkt via "brute force" har genskabt biblen (sker nok først et par år efter dommedag) så er MD5 krypteringen "brudt".

På SQL Server kan man dog, mod bedre vidende, kalde de indbyggede funktioner til adgangskode håndtering. Problemet ved det er ikke sikkerheden, men at Microsoft ikke har dokumenteret disse og ikke garenterer at de fungerer ens mellem forskellige versioner af SQL Server.

Nu advarslen er givet, så er her de "hemmelige" funktioner:

```
pwdencrypt (adgangskode)
```

som krypterer en klartekst-adgangskode til en 46 bytes lang binær værdi, og

```
pwdcompare (adgangskode, krypteret_kode)
```

som sammenligner en klar tekst adgangskode med en krypteret kode og returnerer 1 eller 0 for om den er ok.

Alle brugere, uanset rettigheder, har ret til at afvikle disse! Og inden det undrer nogen - det er altså fordi de bruges når en given bruger logger på systemet eller ønsker at ændre sin adgangskode.

Et par funktioner der wrapper ovenstående, og "vores eget" adgangskode system med irreversibel kryptering er klart:

```
create table dbo.[users] (
  [username] varchar(20) not null primary key nonclustered,
  [password] binary(46) not null
)
go

create procedure dbo.AddUser (@username varchar(20),@password varchar(20))
as
begin
  set nocount on
  -- gem direkte den valgte bruger og adgangskode
  insert into dbo.[users] ([username],[password])
  values (@username, pwdencrypt(@password))
end
go

create function dbo.IsValidLogin (@username varchar(20),@password
varchar(20))
returns tinyint
as
begin
  declare @pwd binary(46)
  -- først finder vi adgangskoden til den pågældende bruger
  select @pwd=[password]
  from dbo.[users]
  where [username]=@username
  -- og så returnerer vi resultatet af sammenligningen
  return pwdcompare(@password,@pwd)
end
```

```
go
```

Hvorom alt er, man bør implementere sin egen MD5 algoritme og lave sine egne udgaver af pwdencrypt() og pwdcompare(). En ide til hvordan man laver sin *seed* værdi og gemmer den i strengen kan ses i næste afsnit.

En kommentar vedrørende funktionen pwdcompare() er nok på sin plads her:

Har man administrative rettigheder på sin SQL Server så har man ret til at tilgå SQL Servers krypterede adgangskoder i tabellen MASTER.dbo.SysLogins! Så smid en ordliste ind i en anden tabel og join de to tabeller via pwdcompare() - og ud kommer alle adgangskoder der er ordbogsord. Nu kan man så tage en hyggesnak med de pågældende brugere...

Reversibel kryptering
Reversibel kryptering er naturligvis kryptering der tillader at den forvanskede tekst kan føres tilbage til klar tekst. Og hvorfor pokker skulle man så lave et sådant system, kunne man spørge? Alt andet lige må det da være mindre sikkert.

Og svaret er "Ja, det er det!". Det eneste er, at irreversibel kryptering bare ikke kan bruges alle steder. Hvis man f.eks. ønsker at lave et system hvor man kan registere sine adgangskoder til andre systemer (f.eks. til administrator kontoen til Windows, root til sin Unix boks, adgangskoden til hardware setup på pc'en og ens firewall / router etc) - så man slipper for at huske dem eller skrive dem på de dersens små gule lapper - så skal man jo kunne få dem tilbage i klar tekst, ellers er ideen i at registere dem ligesom gået fløjten.

Der findes en række forskellige muligheder til sådanne krypteringer, f.eks. kunne man bruge en afart af public key/private key - men det simpleste er "Security through Obscurity"; Hvis ingen ved hvorledes du har valgt at kryptere og dekryptere din tekst, så kan de ikke gøre dig kunsten efter.

Indkodningen kan så være ganske enkel, f.eks. baseret på et *seed*, igen en unik værdi, og en række matematiske funktioner der slutter med en XOR henover klar teksten. Fordelen - og risikoen - ved XOR er, at når man skal dekode kan man blot køre samme XOR igen, og vupti, så har man klar teksten tilbage. En anden ulempe er, at enhver der ser kildeteksten til indkodning og dekodning kan bryde ens system.

Med andre ord - det er altså af allerhøjeste vigtighed at ingen får kendskab til hvorledes ens kodning er bygget op, og at ingen får adgang til ens kildetekst. Det sidste kan Microsoft SQL Server heldigvis hjælpe med.

Alle objekter der skrives i T-SQL, dvs. views, procedurer, triggere og funktioner, kan nemlig krypteres irreversibelt så kildeteksten er skjult for alle - inklusive en selv.

Et system til reversibel kryptering kan være skrevet således:

> **OBS:** Det vil være direkte stupidt at implementere det direkte som vist herunder - alle konstanter og matematiske funktioner bør som absolut minimum ændres; Ideen er jo netop at metoden er skjult!

```
create view dbo.svr_stats
as
  select getdate() as UPDATED,
     @@io_busy as IO_BUSY,
     @@idle as CPU_IDLE,
```

```
        @@cpu_busy as CPU_BUSY
    go

    create function dbo.PwdEncode(@adgangskode char(20))
    with encryption
    returns binary(24)
    as
    begin
      -- lidt variabler der skal bruges. Bemærk i øvrigt at
      -- @adgangskode er en CHAR - så har vi altid 20 tegn.
      declare @s1 int, @b binary(24)
      declare @x1 int, @x2 int, @x3 int, @x4 int, @x5 int
      -- først laver vi vores rimeligt unikke seed værdi
      select @s1 = (io_busy^cpu_idle^cpu_busy)
      from dbo.svr_stats
      -- Så laver vi 5 XOR nøgler (4 bytes pr nøgle = int)
      set @x1=1373772 ^cast(COS(@s1)*10001 as int)
      set @x2=34388432^cast(TAN(@s1)*10002 as int)
      set @x3=20102393^cast(SIN(@s1)*10004 as int)
      set @x4=23883828^cast(TAN(@s1)*10010 as int)
      set @x5=11999212^cast(TAN(@s1)*20000 as int)
      -- og vi laver en binær streng med seed skjult og XOR værdien af
adgangskoden
      set @b = cast(@x4^cast(substring(@adgangskode, 1,4) as binary(4)) as
binary(4))+
          cast(@x3^cast(substring(@adgangskode, 5,4) as binary(4)) as
binary(4))+
          cast(@x5^cast(substring(@adgangskode, 9,4) as binary(4)) as
binary(4))+
          cast(@x2^cast(substring(@adgangskode,13,4) as binary(4)) as
binary(4))+
          cast(@s1^20112919 as binary(4))+
          cast(@x1^cast(substring(@adgangskode,17,4) as binary(4)) as binary(4))
      -- den sender vi retur
      return @b
    end
    go

    create function dbo.PwdDecode(@adgangskode binary(24))
    with encryption
    returns varchar(20)
    as
    begin
      -- lidt variabler
      declare @s1 int, @b char(20)
      declare @x1 int, @x2 int, @x3 int, @x4 int, @x5 int
      -- først finder vi vores seed fra indkodningen
      set @s1=substring(@adgangskode,17,4)^20112919
      -- så gendanner vi vores nøgler fra indkodningen
      set @x1=1373772 ^cast(COS(@s1)*10001 as int)
      set @x2=34388432^cast(TAN(@s1)*10002 as int)
      set @x3=20102393^cast(SIN(@s1)*10004 as int)
      set @x4=23883828^cast(TAN(@s1)*10010 as int)
      set @x5=11999212^cast(TAN(@s1)*20000 as int)
      -- og så kører vi dekoder vi strengen
```

```
      set @b = cast(cast(@x4^substring(@adgangskode, 1,4) as binary(4)) as
varchar)+
          cast(cast(@x3^substring(@adgangskode, 5,4) as binary(4)) as varchar)+
          cast(cast(@x5^substring(@adgangskode, 9,4) as binary(4)) as varchar)+
          cast(cast(@x2^substring(@adgangskode,13,4) as binary(4)) as varchar)+
          cast(cast(@x1^substring(@adgangskode,21,4) as binary(4)) as varchar)
      -- fjern unyttige mellemrum og send retur
      return rtrim(@b)
    end
    go
```

De to funktioner kan nu kaldes direkte i forbindelse med insert, update og select på de tabeller hvor de forskellige adgangskoder er registeret, og selvom krypteringen ikke er særlig stærk, så vil man kunne få nogle hyggelige timer til at gå med at bryde den.

Men der er endnu en svaghed ved reversibel kryptering; Alle der kan kalde funktionerne kan jo dekode det skjulte til klar tekst. Og er det alle ens administrative adgangskoder en hacker får adgang til - thja, så kan man vist godt starte med at kigge i avisernes jobsektion.

Heldigvis er der to måder man kan beskytte sig lidt videre på:

Den første er ved at sætte rettigheder i databasen. Man kan give udvalgte personer ret til at udføre disse funktioner og nægte retten til alle andre. Der er bare det problem, at alle med administrative privilegier (f.eks. SA og DBO) har ret til at udføre alle procedurer og funktioner - uanset hvad der er sat af rettigheder.

Ikke så godt, og en hacker vil jo typisk gå efter disse administrative privilegier - og så er vi jo nok tilbage ved avisernes jobsektion igen...

Den anden måde er ved at tilføje endnu et *seed* til krypteringsstrengen - det kunne f.eks. være en pinkode brugerne skulle indtaste eller bygge på en MAC adresse (disse kan ses for alle forbindelser i tabellen MASTER.dbo.SysProcesses).

Der er et muligt problem ved at bruge en MAC adresse; Ryger netkortet kan man ikke læse sine data før et andet netkort er blevet kodet om til den valgte MAC adresse, og der kan kun være én maskine på et netværk med en given MAC adresse.

Men leger man lidt videre med pinkode-tanken, så kan man give hver bruger en personlig pinkode som via AND og OR gøres til en "global" pinkode der kan kryptere koderne.

Men det at kræve en brugerindtastning af en kode, det åbner for endnu en interessant mulighed som Zedios anførte da jeg diskuterede sikkerhed med ham; Hvis vi nu blot bruger brugeres indtastning til at danne vores XOR række - så kunne vi jo, meget nemmere lave vores reversible kryptering - og vi slipper for svagheden med at have seed'et stående i den krypterede streng.

Og hvorfor ikke slå to fluer med et smæk - benyt MD5 til at lave en XOR-streng udfra pinkoden - og vupti, vores reversible kryptering er færdig. En anden mulighed, for de dovne, er at benytte SQL Servers indbyggede funktion pwdencrypt() som vist ovenfor - men hvis Microsoft senere ændrer implementeringen af pwdencrypt() kan man ikke længere dekode sine data. Lidt trist.

Imod Brute force
Jeg har ovenfor nævnt at brute force kan benyttes til at dekryptere med. Brute force består simpelthen i, at angriberen afprøver samtlige kombinationer af bogstaver og tal der findes indtil den rigtige kombination er valgt.

Alle systemer er derfor sårbare overfor brute force, og forsvaret er at gøre tidsforbruget på angrebet uacceptabel lang.

For det første er det vigtigt at man ikke annoncerer detaljer om krav til adgangskodernes opbygning. Hvis en angriber *ved* at alle adgangskoder skal indeholde en blanding af store og små bogstaver samt cifre og være på minimum 8 tegn - så behøver angriberen jo ikke at prøve nogen kombinationer der ikke overholder kravet!

Angriberen kan ovenikøbet blot starte med at prøve kombinationer der starter med ét stort bogstav, er efterfulgt af 6 bogstaver og slutter med ét ciffer. Så er det blot fat i ordbogen, vælg de 7 bogstavers ord, tilføj cifre og prøv med dem.

Mange adgangskoder vil så ikke blive brudt, men mange, om ikke de fleste, brugere vil kun have en "minimumskravs-adgangskode" - og én brudt adgangskode er nok.

Men i funktioner som de ovenstående kan man lave nogle små ting, der reelt gør ethvert forsøg på brute force ubrugeligt. Den første og mest simple er, at man tilføjer

```
WAITFOR DELAY '00:00:01'
```

i funktionerne IsValidLogin() og PwdDecode().

Nu ville alle valideringer tage 1 sekund. Dermed kan en angriber kun forsøge 86400 gange indenfor et døgn per session. 86400 lyder måske af meget, men for at brute force skal have en change skal vi have over 500.000 forsøg i sekundet hvis vi skal nå at holde vores ferier...

Men angriberen kan jo blot have flere sessioner samtidigt, og så er forsinkelsen ovenfor jo underordnet. Okay, så implementerer man det lidt anderledes - og får samtidig nogle oplysninger til at spore angriberen.

Man laver en logningsfunktion i de to ovenstående funktioner. Loggen består af en tabel hvori man indsætter dato, tid, brugernavn og informationerne fra Master.dbo.SysProcesses for hvert forsøg.

I SysProcesses finder man bl.a. information om den brugte applikation og MAC adressen på brugerens netkort. MAC adressen er normalt unik på verdensplan, så nu begynder det at ligne noget mht at finde den maskine angriberen har overtaget på dit netværk.

Fint, det var loggen - det tager lidt tid, selvfølgelig, at få data i den og det sløver dermed et brute force angreb ned, men vi kan gøre mere.

På loggen lægges en trigger der samler forsøgene op, grupperer dem ud fra tidsstemplet og tæller antal forsøg mod en enkelt konto. Når så der har været f.eks. 3 forsøg indenfor et minut kalder triggeren ovenstående WAITFOR, men nu med 10 - 20 sekunders ventetid.

Dermed kan der, uanset antal sessioner, kun gennemføres under 20.000 forsøg på en enkelt brugerkonto indenfor et døgn - og brugeren oplever ikke engang en forringelse af hastigheden ved et normalt successfuldt login. Og naturligvis, et brute force angreb er ikke længere praktisk gennemførligt.

Implementeringen af dette vil jeg dog ikke begynde på her - men lade stå som en øvelse til de interesserede :-)

God fornøjelse
Troels

Kommentar af hspoulsen d. 07. Jul 2004 | 1

Kommentar af karsten_larsen d. 27. Aug 2007 | 2

Fin artikel som både konkretisere og perspektivere emnet

Kommentar af thomasjepsen d. 31. Mar 2004 | 3

Kommentar af orca d. 20. Jul 2004 | 4

Kommentar af repsak d. 16. Feb 2004 | 5

Ganske interessant artikkel, selvom jeg sprang algoritmerne over ;-)
Mange gode pointer og argumenter

Kommentar af schoesler d. 10. Apr 2004 | 6

Ganske interessant