Knäcka lösenord genom ordboksattacker

Programmering 1, 2017-03-31

I följande övning ska vi försöka knäcka lösenord krypterade (egentligen hashade) med algoritmen DES (Data Encryption Standard). DES var länge standardalgoritmen för kryptering av lösenord på Unix-baserade operativsystem, men har numera ersatts av algoritmer som är bättre rustade mot kryptoattacker. Trots brister i DES visar det sig att algoritmen fortfarande används i många system. Den lösenordlista som vi ska bearbeta läckte efter ett intrång hos en större blogg-tjänst så sent som 2010 (DES blev mer eller mindre dödförklarad mot slutet av 1990-talet).[1]

Funktionen som används för att DES-kryptera lösenord på Unix-baserade system heter crypt(). Den tar två strängar som argument, *key* och *salt*, och returnerar en krypterad sträng, en *lösenordshash*, som är 13 tecken lång. Argumentet *key* är lösenordet vi vill kryptera. För DES är lösenordet begränsat till maximalt åtta tecken. *Salt* är en sträng med två slumpvist valda tecken ur uppsättningen [a-zA-Z0-9.\] vilket ger $64^2 = 4096$ kombinationer. Saltet möjliggör att ett och samma lösenord kan resultera i ett stort antal olika hash-värden (4096 för DES) vilket försvårar vissa typer av kryptoattacker. Märkligt nog hålls saltet inte hemligt, utan återfinns som lösenordshashens två första tecken.

Att knäcka en lösenordshash innebär att man gissar ett möjligt lösenord som sedan krypteras med det aktuella hash-värdets salt. Om det resulterande hash-värdet stämmer överens med lösenords-hashen har vi lyckats knäcka lösenordet. I Ruby finns crypt() tillgänglig som medlemsfunktion till klassen String:

```
encrypted_pass = "bTpZW/VN29vVc"      # DES-encrypted password hash
guessed_pass = "p4ssw0rd"             # the password we'd like to try
salt = encrypted_pass[0..1]           # extract salt from hash

# password found if hashes match
if encrypted_pass == guessed_pass.crypt(salt)
   puts "Password found: #{encrypted_pass} => #{guessed_pass}"
end
```

En enkel men effektiv kryptoattack på lösenordhashar bygger på att man prövar en lista med möjliga lösenord. Metoden som benämns *dictionary attack* eller *ordboks/ordlisteattack* fungerar bra på svagare lösenord, exempelvis vanligt förekommande ord och teckenkombinationer. Hur framgångsrik vår attack blir (success rate) beror på flera faktorer, t ex hur omfattande ordlistor vi har tillgång till, tillgänglig datorkraft och tid, men mest på lösenordens komplexitet.

Ett alternativ till ordboksattacker är att generera samtliga möjliga teckenkombinationer för ett lösenord i en så kallad *brute force-attack* , dvs ”aaaaaaaa”, ”aaaaaaab”, ”aaaaaaac” … osv. Metoden är tidskrävande, men med större datorresurser kommer den knäcka samtliga lösenord.

En ordboksattack kan också utökas till en så kallad *hybrid-attack* genom att transformera orden med en uppsättning regler. Vanliga regler kan innebära att man skapar ett prefix ett suffix till orden med siffror eller specialtecken, ex: ”mary” → ”mary97!”. Man kan också substituera tecken, typiskt vokaler med siffror, ex ”password” → ”p4ssw0rd”.

---

1 Den som vill fördjupa sig i kryptering/hashning av lösenord, ”best practices”, etc, kan exempelvis läsa vidare på https://crackstation.net/hashing-security.htm.

Förslag till arbetsordning och funktionalitet för vår cracker

1. Börja med att verifiera ovanstående exempelkod, dvs att ni kan extrahera salt från ett hash och använda det för att kryptera en given sträng.
2. Studera de föreslagna strategierna nedan och bestäm er för en (eller utarbeta en egen). Är någon strategi enklare att implementera? Vilken verkar vara mest effektiv?
3. Hämta zip-arkivet med lösenordshashar från Classroom. Det finns listor med 100, 1000, 10000 och 700000 hash-värden. Börja arbeta med en lagom stor lista. Kommer stora listor orsaka problem med prestanda eller minne? Bör man bearbeta listan i portioner?
4. Hämta några olika ordlistor från http://www.justpain.com/ut_maps/wordlists/
   Studera gärna listorna i en editor. Hur många ord innehåller listorna? Hur långa är orden? Används stora och små bokstäver?
5. Bygg grundfunktionaliteten för cracker-programmet. Arbeta stegvis enligt given strategi och försök strukturera programmet m.h.a separata funktioner (subrutiner) som har en avgränsad funktionalitet.
6. Förfina programmet genom att spara knäckta lösenord i ett dictionary (datatypen Hash) med lösenordshash som nyckel och lösenordet i klartext som värde. Bygg funktionalitet för att spara/ladda informationen via en textfil vid programmets början/slut.
7. Använd gärna ARGV, dvs kommando-rads-argumenten till programmet för att läsa in filnamn för lösenordshashar och ordlistor.

Strategi 1

```
1. För varje lösenordshash
   2. Extrahera saltet
   3. Generera hash-värden för varje ord i ordlistan med aktuellt salt-värde
       4. Jämför varje genererat hash med aktuell lösenordshash
```

Strategi 2

```
1. Sortera samtliga lösenordshashar
2. För varje ord i ordlistan
   3. Generera samtliga möjliga hash-värden med samtliga (4096) salt-kombinationer
   4. För varje genererat hash-värde
       5. Sök efter matchande lösenordshash
```

Strategi 3

```
1. Sortera samtliga lösenordshashar
2. Skapa en lista över förekommande salt-värden
3. Fortsätt vid steg 2 i strategi 2 men generera endast
   hash-värden baserat på förekommande salt-värden
```