dSik - Key management

Lukas Peter Jørgensen, 201206057, DA4

16. juni 2014

1 Sikkerhedsmål

CAA

CAA står for:

Confidentiality: Information skal holdes hemmelig for uvedkommende, gælder både under forsendelse, opbevaring og behandling af data

Authenticity: Informationen er autentisk, den er ikke blevet manipuleret af en uautoriseret person.

Availability: Systemer skal være tilgængelig når de skal bruges

Definition af et sikkert system

Det er typisk svært eller umuligt at bevise et system er sikkert. Tit bliver man nødt til at lave mange antagelser om angriberens muligheder for at bevise at et system er sikkert og disse antagelser er typisk forkerte.

Vi bruger udtrykket "sikret system"istedet for "sikkert system"da det blot indikerer at vi har sikret systemet mod nogle bestemte tilfælde. Disse tilfælde defineres ved en *Sikkerhedspolitik* på baggrund af en $Trusselmodel$, og implementerer herefter denne sikkerhedspolitik vha. nogle *Sikkerhedsmekanismer*. På den måde får vi, at et sikret system kan beskrives som:

$Sikretsystem = Sikkerhedspolitik + Trusselsmodel + Sikkerhedsmekanismer$

2 Key Management

Key Management handler om problemet der ligger i at få distribueret, opdateret og sikret nøgler.

Man må anerkende to grundlæggende principper: Jo længere og mere man bruger en hemmelig systemparameter jo større er risikoen for at den bliver fundet.

Og det andet princip:

Ethvert system kryptografi må have en eller flere nøgler der beskyttet udelukkende vha. fysiske, ikke-kryptografiske metoder.

En secret-key løsning for de to parter A og B er:

1. A og B beslutter sig for nøgle K_AAB, som kun skal bruges til nøgletransport.
2. Hvis A vil sende en meddelelse til B, så genererer A først en tilfældig sessionsnøgle k.
3. A sender herefter $E_{K_{AB}}(k), E_k(M)$ til B.
4. B bruger herefter nøglen K_{AB} til at dekryptere sessionsnøglen k og bruger derefter denne til at dekryptere meddelelsen.

Problem: I systemer med mange brugere skal der være en nøgletransport for hver bruger, det er ikke godt.

3 KDC

En centraliseret løsning i form af et *Key Distribution Center* (KDC). A har en nøgle K_A til at kommunikere med KDC'en, når A vil snakke med B bliver KDC'en bedt om at generere en nøgle. Fordel: A kan sætte sin lid til at KDC'en sørger for at A ikke bliver narret til at tro at C er B. Ulemper: skalerer dårligt, single-point-of-failure. Endnu vigtigere: A skal stole på KDC'en. En mere decentraliseret løsning er nødvendig for store netværk som f.eks. internettet.

4 PKI(CA)

En anden løsning er *Certification Authorities* (CA) hvor den delte nøgle K_{AB} mellem A og B erstattes af offentlige og private nøglepar.

Det skal nævnes at CA'er blot er en del af en større struktur kaldet *Public Key Infrastructure* (PKI) der står for at skabe, styre, opbevare, distribuere og tilbagekalde digitale certifikater, men vi fokuserer kun på CA'erne.

Alle brugere bør have en korrekt kopi af pk_{CA} på en eller anden måde (e.g. bundled med browseren). En bruger A kontakter CA'en, identificerer sig selv unikt (typisk vha. en *Registration Authority* (RA), der også er en del af PKI.) og sender sin offentlige nøgle pk_A til CA'en. CA'en skaber så et certifikat, som bl.a. indeholder en streng ID_A der identificerer A unikt, A's offentlige nøgle pk_A og CA'ens signatur af ID'et og nøglen således:

$$Cert_A = S_{sk_{CA}}(ID_A, pk_A)$$

På den måde kan enhver bruger med pk_{CA} nu tjekke at de har fat i en gyldig offentlig nøgle for A og kan derfor kommunikere sikkert med A.

Her skal man have tillid til at CA'en har tjekket en brugers identitet ordentligt, men modsat KDCC har CA'en ikke mulighed for at signere dokumenter eller dekryptere for en da den ikke har fat i den private nøgle.

Hvis en bruger opdager at sin private nøgle er blevet stjålet eller gået tabt, så kan han kontakte sin CA og få tilbagekaldt sit certifikat. Så bliver den sat på en offentlig *Certificate Revocation List* (CRL). Certifikater kan også udløbe efter noget tid.

Certificate chains

Hvad nu hvis to brugere har to forskellige CA'er? Da har de ikke den offentlige nøgle til hinandens CA'er. Til dette problem bruger man certificate chains.

Hvis to brugere A og B med CA'er CA_1 og CA_2, så vil A modtage B's certifikat $Cert_{CA_2}(B, pk_B)$. Men A har ikke CA_2's offentlige nøgle, så det man gør er at sørge for at CA'er sørger for at validere hinandens offentlige nøgler, således A kan anmode CA_1 om CA_2's certifikat og få $Cert_{CA_1}(CA_2, pk_{CA_2})$. Lidt mere generelt kan man have en kæde af CA'er der validerer hinandens certifikater for at finde det endelige certifikat.

Problemer: Man skal stole på alle CA'er i kæden for at kunne stole på B's certifikat. Desuden skal man stadig kende mindst en CA's certifikat fra starten. Sidste problem kan dog løses ved at have det pre-installeret i sit software, man skal dog stole på softwaren så.

X.509

X.509 er et certifikat-standard der definerer en række minimumskrav til et certifikat. Pga. manglende fleksibilitet i de tidligere versioner af X.509 er der blevet lavet en række tilføjelser som ikke alle har implementeret, så der er flere applikationer der påstår at understøtte X.509 men alligevel ikke kan snakke sammen.

Tyveri af nøgler

Hvis man ikke er et stort firma, der investerer i noget sikker hardware til at passe på ens nøgler, gemmer man dem typisk blot på en lokal disk. Men så skal man have den nøgle beskyttet fra tyveri. Dette gøres ved at bruge passwords som man hasher så den for en bestemt længde:

$$E_{H(pw)}(sk)$$

Men da antallet af passwords er meget mindre end 2^n, hvor n er antallet af bits i hashfunktionens output, er det meget lettere at bruteforce. Derfor bruger man typisk en meget langsom hashfunktion (f.eks. at bruge den samme hashfunktion tusindvis af gange.), dette er ikke et problem hvis man kender kodeordet og bare skal udregne det en gang. Men hvis man prøver at bruteforce tager det så lang tid at det er et stort problem.

5 Passwords

Når man evaluerer passwords er der fire spørgsmål man skal stille sig selv:

- Hvordan blev passwordet valgt?
- Hvordan blev passwordet transmitteret til systemet der skal verificere det?
- Hvordan opbevarer du passwordet?
- Hvordan opbevarer systemet passwordet?

Passphrases

E.g. "Syv store trold stop på en klint-> "Sstspek". Nemmere at huske, lige så svære at gætte.

6 Angreb på passwords

Password Crackers

Dictionary-attacks og bruteforce. Passwords er typisk hurtige at gætte med en god algoritme, dette kan modvirkes ved at låse kontoen eller blokere en ip efter f.eks. 3 forsøg.

Social Engineering

At snyde folk igennem falske emails, eller at udgive sig for at være en tekniker der skal reparere nogle systemer.

Phishing/Smishing

(Phishing = email, Smishing = sms)

Vha. falske meddelelser lokker man ofre ind på hjemmesider der ligner en banks hjemmeside så man ikke indser at det er en falsk hjemmeside.

Indbrud

Hvis en angriber bryder ind på en server vil han have adgang til en database, med enten passwordet eller en hashed version af passworded. Hvis du bruger samme password mange steder, kan det betyde at flere systemer bliver kompromitteret.

7 Biometrics

Fordele: Du har den altid med dig, det er svært at forfalske.

Ulemper: Ikke mulighed for anonymitet. Ingen mulighed for at ændre din biometri, så hvis dine digitale data for fingeraftryk bliver stjålet, er det for evigt. Det er let at stjæle f.eks. fingeraftryk som tit bliver brugt som ID. F.eks. laptop, fingeraftryk overalt og fingeraftryk som sikkerhed.