KTH Datavetenskap
och kommunikation

2D1395 Datasäkerhet
Lösningar till tentamen 2006-10-18

1 Förklara innebörden av nedanstående begrepp och ge exempel på mekanismer och tekniker som används för att åstadkomma dem. [6p]

Konfidentialitet (confidentiality)
Integritet (integrity)
Spårbarhet (accountability)

Lösning:

Konfidentialitet Att information inte ska kunna läsas av obehöriga. En teknik för att åstadkomma detta är kryptering.

Integritet Att information inte ska kunna förändras av obehöriga eller genom misstag utan att det upptäcks. Tekniker för att ådtadkomma detta är checksummor som beror av en hemlig nyckel (t.ex. HMAC) eller digitala signaturer.

Spårbarhet Att det ska vara möjligt att säkert veta vem som gjort vad (exempelvis vem som loggat in, läst eller ändrat filer). Detta säkerställs genom loggning av händelser i systemet. Det är viktigt att loggar inte kan förstöras eller förvanskas.

2 En lösenordsfil i exempelvis Unix innehåller bland annat användarnamn och hashade lösenord istället för lösenorden i klartext. Om någon får tag i lösenordsfilen så ska det inte ge tillgång till lösenorden i klartext.

2a Vilka krav bör man ha på den hashfunktion som används? [1p]

Lösning: Det mest centrala är att det givet hashvärdet $h(x)$ ska vara svårt att hitta x eller något annat värde y som uppfyller $h(y) = h(x)$

2b Ibland används *saltning* av lösenorden. Vilken sorts attacker skyddar saltning mot, hur går en sådan attack till (utan salt och med salt), och hur mycket mer tidskrävande blir en sådan attack om man har en lösenordsfil med 1000 användare och använder 8 bytes med salt? [2p]

Lösning: Låt oss till exempel anta att SHA-1 används för hashning av lösenord. Saltning innebär att man för varje användare u väljer en slumpsträng R_u. Om u har lösenord P_u så lagras i lösenordsfilen SHA-1$(P_u||R_u)$ och R_u tillsamans med användarens identitet (olika R_u för olika användare).

Saltning är tänkt att skydda mot ordlistattacker. En ordlistattack innebär att man använder en lista $P_1, \ldots, P_N$ med vanligt förekommande lösenord. Om lösenordsfilen innehåller hashade lösenord men utan saltning går ordlistattacken till så att man bildar listan av hashade ordlisteord

(SHA-1(P_1), . . . , SHA-1(P_N)). Sedan söker man efter varje hashat lösenord i lösenordsfilen bland listan med hashade ordlisteord. Med bra val av datatruktur kan man räkna med att attacken tar tid $O(N + M)$, om det finns M användare på systemet.

Om vi däremot har salt kommer vi att för varje ord P_i i ordlistan och varje saltvärde R_u i lösenordsfilen behöva beräkna SHA-1($P_i||R_u$) och med 1000 användare och 8 bytes salt innebär det att med mycket hög sannolikhet har alla användare olika salt. Det blir alltså $N \cdot M$ beräkningar av SHA-1 nu jämfört med N stycken utan salt. Om ordlistan har betydligt fler än 1000 ord, vilket är rimligt (`/usr/dict/words` har drygt 25000 ord) så kommer saltning att innebära att det tar ungefär M gånger längre tid (dvs ca 1000 gånger längre tid i exemplet).

3 Min prenumeration på ett antivirusprogram gick ut för några månader sedan, men nu har jag förnyat den. Jag har nu kört den uppdaterade versionen och den har rensat bort ett antal spionprogram och virus. Vad bör jag oroa mig för när det gäller säkerheten på datorn efter uppgraderingen? Vilka ytterligare åtgärder kan jag vidta och vad har de för fördelar och nackdelar? [3p]

Lösning: Problemet är att under den tid datorn inte fick virusprogrammet uppdaterat har den uppenbarligen utsatts för attacker. Dessa kan ha lämnat root-kit efter sig som ofta missas eller inte kan detekteras av antivirusprogram, och utomstående kan ha utnyttjat detta till att konfigurera om datorn på sätt som inte lämnar spår efter sig som antivirusprogram kan upptäcka. Saker har helt enkelt gjorts av någon med administratörsrättigheter. Man kan skaffa program som är till för att leta efter root-kit för att försöka hitta och ta bort dem, vilket inte är helt lätt.

Vill man göra en mer noggrann rensning kan man installera om datorn. Detta är dock tidsödande och inte heller helt riskfritt. Installationsskivorna är antagligen gamla och det innebär att när man installerat om startar datorn utan en stor mängd säkerhetsuppdateringar. För exempelvis Windows kommer dessa att hämtas över nätet och systemet patchas. Men under denna process är datorn uppkopplad och tämligen oskyddad.

Man skulle även kunna oroa sig för att BIOS påverkats.

Kommentar: Det finns naturligtvis en massa allmänna goda råd som att ta regelbundna backup:er, använda IDS och så vidare, men de har inte så mycket att göra med det beskrivna scenariot.

4 Foo AB har kommit på ett jättefiffigt system för att hantera inloggning på Unix-liknande system. Istället för en lösenordsfil så har de en katalog med en fil per användare där filnamnet har formatet *användarnamn.lösenord* (exempel följer). Detta gör att de slipper implementera hashfunktion och annat jobbigt. Katalogen kan bara listas av administratören (som då förvisso kan se allas lösenord, men så kan det gå med fiffiga lösningar).

Användarnamn består enbart av små bokstäver. Lösenord kan innehålla bokstäver (stora och små), siffror, samt ett 20-tal övriga tecken.

T.ex. kan katalogen se ut så här (men man måste vara administratör för att lista filerna):

```
host# ls /secret/accounts
alice.Am3hc94-"# bob.password532
```

För att autentisera användarnamn och lösenord används följande pythonkod:

```
def goodpass(username, password):
       upass = username + '.' + password
       # No whitespace allowed in username or password
       for c in string.whitespace:
                if c in upass:
```

```
                    return False
    # Verify username and password
    if(len(glob.glob("/secret/accounts/" + upass)) == 1):
            return True
    else:
            return False
```

Det du behöver veta om python är att `glob.glob` fungerar likadant som om du gör '`ls`' med samma argument och att `len(...) == 1` kontrollerar att man får tillbaka exakt ett resultat. Du kan anta att den första for-slingan som kontrollerar att det inte finns whitespace (mellanslag m.m) i strängarna fungerar.

4a Om vi känner till att alice är ett användarnamn på systemet, beskriv en effektiv attack för att lyckas logga in som alice. [1p]

Lösning: *Kommentar: Den här uppgiften är ett exempel på en vanlig form av säkerhetshål, nämligen otillräcklig indatakontroll. Problemet är att användaren bara antas mata in sådant som verkligen kan vara riktiga användarnamn och lösenord. Genom att mata in helt andra saker kan en illasinnad användare kringå säkerheten. Visst kan man angripa detta system med generiska attacker som ordlistattacker och "social engineering" men här finns ett säkerhetshål i lösenordsprogrammet som är så stort att det utan tvekan ger den mest effektiva vägen för en attack.*

I denna deluppgift räcker det att mata in `alice` med `*` som lösenord eftersom `glob.glob` då får `alice.*` som argument och det kommer att matcha exakt en fil, nämligen den som motsvarar `alice` och hennes lösenord.

4b Vi misstänker att bob använder samma lösenord på ett annat system. Beskriv en effektiv attack för att ta reda på hans lösenord. Det finns inget kommando när man väl är inloggad för att visa lösenord, så det räcker inte med att man lyckas att logga in som bob. [1p]

Lösning: Lösenordskontrollen låter oss bestämma ett tecken i taget i lösenordet för `bob`. Genom att testa i tur och ordning lösenorden `a*`, `b*`, `c*`,... kommer vi att kunna bestämma första tecknet (vi måste förstås prova med alla tecken som är tillåtna i lösenordet) eftersom exakt ett av mönstren svarar mot lösenordet. När vi vet förstabokstaven, exempelvis "`q`, kan vi fixera den och prova fram nästa bokstav: `qa*`, `qb*`, `qc*`,..., och så vidare tills vi har hela lösenordet.

4c Vi känner inte till några användarnamn på systemet. Beskriv en rimligt effektiv attack för att lyckas logga in. [1p]

Lösning: I detta fall blir attacken inte lika snabb som i b-uppgiften, men fortfarande rimlig. Vi testar `a*`,`b*`,... som användarnamn och `*` som lösenord. Om det finns någon användare som är ensam om förstabokstaven i användarnamnet kommer vi att lyckas logga in. Lyckas inte det provar vi alla kombinationer av två första tecken i användarnamnet, osv. Har vi riktig otur, så kommar kortaste prefix som unikt identifierar en användare att vara långt, och då tar detta lång tid, men antagligen räcker ett par tre tecken.

4d Vi är intresserade av hurvida en viss fil, `/etc/uucp/passwd`, finns. Som användare har man inte listrättighet till katalogen `/etc/uucp/`. Beskriv en attack som tar reda på om den filen finns på systemet. [1p]

Lösning: Ange `../.` som användarnamn och `/etc/uucp/` som lösenord. Konkateneringen ger `../../etc/uucp/` vilket skulle vara sökvägen till den filen om vi utgår från `/secret/accounts/`.

5 En brandvägg kan användas för att förhindra intrång. Det finns även ett antal tekniker som används för att upptäcka en pågående attack eller för att man i ett senare skede ska kunna anlysera misstänkta aktiviteter.

5a Bland annat använder man intrångsdetekteringssystem (IDS). Beskriv ett par lämpliga datakällor för ett IDS att jobba med. [1p]

Lösning: Huvudakliga källor är systemloggar (och ev checksummor eller liknande för känsliga filer) och nätverkstrafik.

5b Vilka två huvudsakliga tekniker arbetar IDS med för att analysera insamlade data? [1p]

Lösning: Mönsterigenkänning som innebär att systemet försöker upptäcka kända attacker genom att de ofta lämnar vissa typer av spår.

Statistiska metoder som innebär att systemet har en statistisk modell av vad som är ”normalt” och larmar om det sker avvikelser.

5c Ofta vill man ha två (eller flera) IDS uppe parallelt. Varför? [1p]

Lösning: Flera tänkbara orsaker. Olika IDS kan kontrollera olika saker (t.ex. HIDS och NIDS). I ett stort system behövs lastdelning. En angripare kan hacka ett IDS, vilket skulle kunna upptäckas av ett annat IDS.

5d ”Honeypot” är ett kompletterande verktyg. Vad är en honeypot och varför har man en? [1p]

Lösning: Det är ett system som inte innehåller något känsligt eller viktigt, utan vars enda uppgift är att bli angripet. Det ger administratön möjlighet att studera en pågående attack utan att behöva vara orolig för skador. Det kan avleda angripare från andra, känsliga men bättre skyddade, system.

6 I det här problemet använder vi asymmetrisk kryptering och digitala signaturer som vi tänker oss fungerar ungefär som i GnuPG. Vi antar att man inte använder samma nycklar för kryptering och signering. Antag att A har Bs publika nycklar och vice versa.

Nedan har vi hoppat över den del av GnuPG som har att göra med att generara en sessionsnyckel som används för symmetrisk kryptering av meddelandet (man fick poäng vare sig man använde en sessionsnyckel eller ej).

6a A vill skicka ett krypterat och signerat meddelande till B. Beskriv As protokoll för att göra detta (dvs vilka moment som ingår och hur nycklar används). Beskriv på motsvarande sätt vad B ska göra för att kunna kontrollera integritet och läsa klartexten. [1p]

Lösning: A har nyckelpar (E_A, D_A) för kryptering respektive dekryptering samt (S_A, V_A) för signering respektive verifiering av signaturer. De publika nycklarna är E_A och V_A. Motsvarande för B är (E_B, D_B) och (S_B, V_B).

A gör följande för att skicka meddelande m. Kryptera m med E_B för att erhålla kryptotext c. Signera c med S_A för att få signatur σ. Skicka c och σ till B.

B genomför verifiering av att σ är en signatur som A gjort på c, med hjälp av As publika nyckel V_A. Därefter dekrypterar B kryptotexten c med D_B och erhåller m.

Kommentar: Man kan också tänka sig att A signerar m och sedan krypterar (m, σ). Då måste B dekryptera innan signaturen kontrolleras. Att signera m och sedan bara kryptera m medan σ skickas i klartext är en dålig ide, eftersom σ kan läcka information om m.

6b Elaka E har fått tag på ett antal meddelanden som B fått och som är krypterade och signerade genom att stjäla ett USB-minne där B förvarade dem. Dessa meddelanden är från ett antal olika parter som alla kommunicerat på samma sätt som A i uppgiften ovan. Vilka möjligheter har E att undersöka vem som skickat meddelanden till B? [1p]

Lösning: Svaret på den här frågan är beroende av svaret i föregående fråga. Om signaturerna krypterats så är det svårt att ta reda på vem som gjort dem. Är det okrypterade signaturer av det krypterade meddelandet är det antagligen lätt. Man kan skaffa fram de publika nyklarna för ett (stort) antal personer och kontrollera om någon av dessa nycklar fungerar vid verifiering av signaturena.

6c Om A och B inte har varandras nycklar kan de börja med att e-posta dem till varandra för att sedan kommunicera krypterat via e-post. Förklara principerna för en attack som låter E läsa de krypterade breven utan att A och B upptäcker det. Vilka mekanismer och tekniker finns för att undvika detta problem? [2p]

Lösning: Man kan råka ut för en mannen-i-mitten-attack (”Man in the middle”). För beskrivning hänvisas till kursboken (men för att få poäng måste attacken förstås beskrivas i lösningen).

För att undvika denna attack kan man antingen utnyttja PKI, vilket innebär att A och B kan utbyta certifikat som visar vilka publika nycklar de har. Eller så kan man utbyta nycklarna (eller ”fingerprints” för dem) via en annan kanal, exempelvis telefon, där det förhoppningsvis är lättare att verifiera identiteten på den man talar med (man känner kanske igen rösten).

7 Hur går en *buffer overrun*-attack mot stacken till? Ett motmedel är att använda en ”kanariefågel” (canary). Vad innebär det? Datorsystem erbjuder ofta olika former av minnesskydd. Ge exempel på ett sådant som skyddar mot denna attack. [3p]

Lösning: Målet med attacken är att få den angripna datorn att exekvera kod som man själv valt, exempelvis för att öppna ett skal (kan vara användbart om programmet kör med mer rättigheter än man själv har). Säkerhetshålet består exempelvis i att en funktion i programmet läser indata som lagras i en buffer på stacken utan att någon kontroll sker av att indata ryms i buffern. Om man matar in för mycket indata kommer det att spilla utanför buffern och skriva över intilliggande minne på stacken. Man ser till att det som spiller över dels ersätter återhoppsadressen (som ofta lagras på stacken) med en adress som pekar på den del av stacken man skriver över. Vidare ser man till att indata som hamnar där den ”nya” returadressen pekar består av maskinkod som utför det man själv vill, t.ex. ett systemanrop för att starta en kommandotolk.

En kanariefågel är en enkel teknik som kan hjälpa till att detektera buffer overrun. Man lagrar ett speciellt värde på stacken ovanför returadressen men under arean där funktionens lokala variabler (t.ex. buffern) ligger. En overrun kommer att skriva över minnescellen som innehåller ”kanariefågeln”. När funktionen avslutas kotrollerar den att kanariefågeln har rätt värde innan den gör återhoppet. Om den inte har rätt värde avbryts exekveringen. För att detta ska fungera så måste det vara omöjligt för en angripare att gissa kanariefågelvärdet. Det kan alltså inte vara en konstant, utan bör vara ett (pseudo)slumpvärde.

Minnsskydd som kan hjälpa är t.ex. om minnet är segmenterat eller sidindelat i kombination med möjligheten att markera att vissa segments eller sidors innehåll inte får exekveras. Om programmets stack, heap (och ev globala data) ligger på sidor eller segment vars innehåll inte kan får exekveras som kod så hindras attacken ovan. Detta hindrar dock inte att returadressen skrivs över, men det gör det svårare för angriparen att stoppa in godtycklig kod att hoppa till.

8 Java är ett språk som stöder kodbaserad åtkomstkontroll (CBAC), vilket innebär att olika klasser kan exekvera med olika rättigheter. Låt oss anta att vi har två klasser, `A` och `B`. Klassen `A` har metoden `log(String m)` som lägger till en rad med strängen `m` till filen `events.log` (exempelvis genom att först läsa in hela filen `events.log`, sedan öppna den för skrivning och skriva tillbaks allt det inlästa och lägga till strängen `m` och till sist stänga den).

Dessutom har `A` och `B` metoder som följer:

```
import java.io.*;

class A
{
    void log(String m) throws Exception
    {   /* gör det som beskrivs i texten ovan */    }

    void blog(String m) throws Exception
    { B b = new B(); b.alog(m); }

    void alog(String m) throws Exception
    { A a = new A(); a.log(m); }
}

class B
{
    void alog(String m) throws Exception
    { A a = new A(); a.log(m);  }
}
```

Med hjälp av en policyfil har `A` läs- och skrivrättigheter till `events.log`. Rättigheterna för `B` tillåter inte skrivning av filer alls.

En användare skriver ett program som har fulla rättigheter:

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    B b = new B();
    try {  a.log("1");  }  catch (Exception e){ };
    try {  b.alog("2"); }  catch (Exception e){ };
    try {  a.alog("3"); }  catch (Exception e){ };
    try {  a.blog("4"); }  catch (Exception e){ };
  }
}
```

Användaren som startar programmet har rätt att läsa och skriva `events.log` som från början är tom. Vad kommer det att stå i `events.log` efter att programmet körts en gång (motivera genom att förklara vilka för frågan relevanta kontroller som sker vid körningen)? [3p]

Lösning: En *stackwalk* används för att avgöra om processen har rätt att göra en operation. Om all kod på stacken har rätt att utföra operationen så är den tillåten. Annars kastas ett undantag (som i exemplet fångas i `main`). I det här fallet gäller det att `Test` och `A` har skrivrättigheter men inte `B`, så i de fall det finns metoder från `B` aktiva på stacken så kommer skrivning att nekas. Detta innebär att skrivning av ”1” och ”3” lyckas men skrivning av ”2” och ”4” nekas. Logfilen kommer alltså att innehålla

```
1
3
```

Kommentar: Om man svarat något annat än att 1 och 3 skrivs ut har man fått 0 poäng (det fanns ingen som svarat så att utskriften var fel trots att förklaringen av mekanismen var korrekt).