Fakultet
Ekonomi, Kommunikation och IT

Joakim de Jong, Carl-Henrik Svanemark

Definition av Säkerhetsevaluering

Definition of Security Evaluation

Examensarbete 15 poäng
Dataingenjörsprogrammet

Datum/Termin:	09-06-03
Handledare:	Simone Fischer-Hübner
Examinator:	Martin Blom
Ev. löpnummer:	C2009:04

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

Definition av Säkerhetsevaluering

Joakim de Jong, Carl-Henrik Svanemark

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Joakim de Jong

Carl-Henrik Svanemark

Godkänd, 09-06-03

Handledare: Simone Fischer-Hübner

Examinator: Martin Blom

Sammanfattning

Detta examensarbete har gjorts på uppdrag av Compare Testlab på Sätterstrand, Hammarö. Målet med arbetet var att definera en uppsättning frågor och punkter som ska vara med i ett lättviktstest med inriktning på säkerhet. Efter mycket studier av böcker och webbsidor kom vi fram till att det inte var så lätt som vi trodde att definera ett allmänt säkerhetstest för mjukvaror eller applikationer. Efter samtal med uppdragsgivaren bestämdes det att arbetet skulle begränsas till webbapplikationer.

Internet används mer och mer för varje år som går och det dyker upp nya webbapplikationer i snabbare takt än gamla försvinner. Många företag är beroende av webbapplikationer för att kunna bedriva versamhet. Även andra företag och organisationer, som inte är helt beroende av webbapplikationer, har ofta nytta av dem. Denna rapport behandlar vissa punkter och aspekter som man bör tänka på vid ett sådant test. Dessa punkter inkluderar till exempel autentisering, säker kommunikation och attacker mot webbapplikationer.

Definition of Security Evaluation

This work has been done on behalf of Compare Testlab that is located on Sätterstrand, Hammarö. The purpose of the work was to define a set of questions to be considered in a light-weight test with a focus on security. After a study of literature found in different books and at web pages we came to the conclusion that it is not as easy to define a general security test for software or applications as we thought from the beginning. Hence, after talking to the employer it was decided that the work should be limited to web applications.

Internet is used more and more for every passing year and new web applications appear faster than old applications disappear. Many companies are dependant of web applications to be able to conduct business. Also, other companies or organisations that are not totally dependant of web applications often have good use of them. This report investigates some points and aspects to be considered in such a test. These points include for example authentication, secure communication and attacks on web applications.

Innehåll

Figurer

Tabeller

1 Inledning

Sedan 90-talet har Internet spridit sig allt mer och idag finns det över i stort sett hela världen. Internet har blivit en oerhört viktig infrastuktur som används i många sammanhang, mycket tack vare att teknikerna som används över Internet under åren har utvecklats och blivit flera. Företag använder Internet för att marknadsföra sig, interagera med sina kunder och som ett verktyg i sitt dagliga arbete. Det har gått så långt att Internet är funktionskritiskt för många verksamheter. I takt med att Internet har vuxit och spridit sig har också hotbilden vuxit och förändrats. Internet designades inte med säkerhet i åtanke, vilket till exempel kan ses på bristen av säkerhet i de grundläggande protokollen, och kan därför vara ett verktyg för någon som vill utnyttja systemet. Attackerarna över Internet har utvecklats från att i början vara nyfikna människor, som kanske bara vill se vad som är möjligt att göra, till att vara kriminella människor med intressen i politik och pengar. Kunskapensnivån som krävs för att genomföra en attack över Internet har sjunkit. Med hjälp av verktyg som finns tillgängliga på Internet kan vanliga personer genomföra attacker som man behövde stora kunskaper för att kunna genomföra förr.

Compare Testlab, en del av stiftelsen Compare, jobbar för att kunna erbjuda oberoende tester av open source-applikationer. Vi har fått i uppdrag att hjälpa Testlabbet utveckla en del av den test-suite som behövs för att kunna utvärdera de applikationer som ska testas. Testerna ska kunna ge en bild av en vid samling egenskaper hos den testade mjukvaran. Vårt uppdrag gäller säkerhetsdelen av testerna. Vad säkerhetstestet bör omfatta och varför, är det som vi går igenom i den här uppsatsen. Till en början var det meningen att testet skulle omfatta alla sorters mjukvaror, men vi var tvungna att göra en avgränsning för att kunna genomföra undersökningen. Detta eftersom olika sorters applikationer har mycket olika karaktär, något som påverkar vilka hot applikationen behöver skydda sig emot. Det är med andra ord svårt att komma fram till generella säkerhetsprinciper för alla sorters mjukvaror. Avgränsningen har vi gjort genom att fokusera på webbapplikationer, den applikationstyp som huvudsakligen kommer testas av Compare Testlab.

2 Bakgrund

2.1 Uppdragsgivaren

Compare Testlab är en oberoende aktör som erbjuder infrastruktur, metoder, kompetens, tjänster och koncept inom området mjukvarutestning. Företaget har ett samarbete med Open Sweden och Programverket.org vilka i sin tur har ett tätt samarbete. Open Sweden jobbar för att stimulera användning av öppna program och öppna standarder inom offentlig sektor. Programverket.org jobbar för ett ökat samarbete mellan kommuner, landsting och myndigheter när det gäller öppen mjukvara. Compare Testlab håller på att utveckla ett ramverk för tester av mjukvaror, där allmänheten ska kunna se resultaten av dessa tester. Detta för att lättare kunna avgöra om mjukvaran passar deras behov och uppfyller deras kriterier inom till exempel prestanda, underhåll och säkerhet. Huvudmålgruppen är SKL (Sveriges Kommuner och Landsting).

2.2 Syfte

Syftet med vårt arbete är att definiera ett säkerhetstest som ska ingå i ett existerande testkonceptsramverk (SKL-ramverket som beskrivs nedan). Testet ska vara på en rimlig nivå för att det inte ska vara för dyrt att genomföra, men ändå tillräckligt omfattande för att få en relevant bedömning av säkerheten för mjukvaran. Mjukvarorna som testas kan variera i storlek och komplexitet. Därför behöver testet vara väl genomtänkt för att passa olika sorters mjukvaror.

2.3 SKL-ramverket

SKL-ramverket är under uppbyggnad och är tänkt att innehålla lättviktstester för mjukvaror i samarbete med Open Sweden riktat mot kommuner och landsting. Det kommer även att innehålla en webbportal där testresultaten kan publiceras för lätt åtkomst. För

att testerna ska kunna publiceras på webbportalen kommer det finnas en sida, som endast auktoriserade testare har åtkomst till, där de lägger in sina resultat.

3 Säkerhet

3.1 Definition

Olika grupper eller människor har definierat säkerhet på olika sätt. Men på några punkter brukar man allmänt komma överens.

> *"When we talk about 'computer security', we mean that we are addressing three very important aspects of any computer-related system:* ***confidentiality, integrity,*** *and* ***availability.****"*
> Pfleeger, [16, s. 10]

> *"****Confidentiality*** *ensures that computer-related assets are accessed only by authorized parties. That is, only those who should have access to something will actually get that access."*
> Pfleeger, [16, s. 10]

> *"In information security,* ***integrity*** *means that data cannot be modified without authorization."*
> Wikipedia, [27]

> *"****Availability*** *refers to the ability to use the information or resource desired. Availability is an important aspect of reliability as well as of system design because an unavailable system is at least as bad as no system at all."*
> Bishop, [2]

Vi kommer att begränsa oss genom att inte inkludera säkerhet mot fysiska hot som inbrott, jordbävningar, eldsvådor m.m. Dessutom kommer vi inte inkludera mjukvarans omgivning, till exempel operativsystem, eventuell brandvägg, databas och nätverksuppbyggnad. Dessa begränsningar sätter vi eftersom vårt fokus ligger på mjukvara som ofta kan existera i olika miljöer med olika krav och specifikationer. Det skulle gå att göra ett test för varje separat plattform men det ingår inte i den uppgift vi fått av Compare Testlab och det skulle krävas längre tid än vi har på oss att få fram ett sådant resultat.

3.2 Värde av säkerhetstest

Om en användare, som kan vara till exempel en kommun, förening, enskild person eller ett företag, vill börja använda en applikation som är open-source så är det i vissa fall viktigt att veta hur säker applikationen är. Det skulle till exempel inte vara bra om applikationen hanterar personuppgifter men inte krypterar något eller skyddar informationen från att läcka ut på något sätt. För att få en bild av vad applikationen gör för att skydda informationen så kan man genomföra ett test av säkerheten. Nu är personuppgifter bara ett exempel på vad man skulle kunna testa i en applikation. Om det redan finns ett testresultat för en viss applikation så kan intressenten själv granska resultatet och bedöma hur bra applikationen passar i dess verksamhet.

4 Förstudie

Under förstudien upptäckte vi att det är svårt att definiera vad säkerhet är. Därmed är det även svårt att definiera ett säkerhetstest och vilka kategorier som kan ingå. Från början så tänkte vi att man kunde läsa i böcker och på Internet för att få idéer därifrån. Detta visade sig vara felaktiga antaganden, då nästan all litteratur gäller säkerhet för ett helt system. Ingen av "kategorierna" som vi hittade passade in på att testa endast en mjukvara/applikation, utan var definierade för systemet mjukvaran befinner sig i. Med system

så menar vi omgivningen i form av operativsystem, hårdvara och eventuell brandvägg. Det är dessa tre delar som ur ett säkerhetsperspektiv utgör stommen i systemet runt ett program. Det som vi hittar ganska lätt om säkerhet för enstaka program/mjukvaror är programmeringsfel. Men för att testa programmeringsfel så måste man i stort sett gå igenom hela koden för programmet och det skulle bli för tidskrävande och därmed för dyrt.

På grund av dessa svårigheter satt vi i flera dagar och letade i litteraturen. Det kändes som om vi inte kom någonstans och tiden kändes bortslösad. Därför bestämde vi oss för att sätta oss ner med våra uppdragsgivare för att komma fram till en lösning.

Under mötet med uppdragsgivaren kom vi fram till att det var en lite för stor uppgift att försöka hitta generella frågor för många olika mjukvaror, så det bestämdes att vi skulle utgå från en befintlig mjukvara som används för pilottester. Om tiden sedan räcker till kan man generalisera frågorna lite mer. Mjukvaran vi fick var en webbapplikation som hanterar personer och jobb/projekt som personerna arbetar med.

5 Punkter att testa

5.1 Personlig integritet

Det talas ofta om att man ska skydda datasystem från intrång och andra hot. En viktig del av datasäkerheten som man inte får glömma är personlig integritet. Nu för tiden lagras allt mer data, både hemlig och offentlig, om personer och det blir allt viktigare att se till att inte information om personerna läses eller ändras av obehöriga.

5.1.1 Definition

Hur definieras då personlig integritet?

> *"**Integritet**, rätt att få sin personliga egenart och inre sfär respekterad och att inte utsättas för personligen störande ingrepp (personlig integritet)"*

Nationalencyklopedin, [11]

*"**Integritetskränkning**, Numera för ordet integritetskränkning tankarna till personlighetsskyddet. Varje person har rätt att ha ett område som är skyddat mot intrång."*
Nationalencyklopedin, [11]

En aspekt som många inte tänker på är e-mail [16, kap. 9.6]. E-mail kan bäst liknas med ett vykort som man skickar med vanlig post. På vägen fram till mottagaren kan vem som helst som kommer i kontakt med vykortet läsa vad som står på det. Samma sak är det med e-mailmeddelanden. Eftersom det är många som inte vet om att det som skrivs i mailet kan läsas av andra så bryr de sig inte om att kryptera innehållet. Något som förhindrar att information om specifika individer samlas in genom avlyssning på detta sätt är att det finns en så stor mängd meddelanden att gå igenom. De två platser med störst risk för avlyssning av e-mail är hos eller i närheten av sändaren eller mottagaren. På dessa platser är det lättare att urskilja de meddelanden som man är intresserad av att undersöka.

Personlig integritet handlar inte bara om vad man gör på Internet eller vilka personuppgifter som finns lagrade. Få inser hur mycket information som kan samlas ihop om en person [16]. Några exempel på spår som lämnas är:

- Banker får information om var man är när man tar ut pengar och vid vilken tidpunkt transaktionen skett.

- När man ringer med en mobiltelefon vet operatören vilken mast som används och därmed ungefär var man befinner sig, vid vilken tidpunkt samtalet var och hur länge samtalet varade.

- Strömförbrukningen för ditt hem kan övervakas och slutsatser kan dras om hurvida någon är hemma eller inte.

- En transponder som används som betalning i en vägtull gör att tidpunkt och plats registreras. Den informationen kan i efterhand användas som bevisning om man är misstänkt för fortkörning. Detta genom genom att räkna på tiden mellan två vägtullar och jämföra med avståndet mellan dem och på så sätt få fram medelhastigheten man haft på sträckan mellan tullarna.

  Informationen som samlas in av olika webbapplikationer kan vara nödvändig att spara av olika anledningar. Till exempel kanske en bank vill kunna erbjuda en tjänst där man kan se sina egna transaktioner bakåt i tiden. Däremot så kan sådan information vara till skada om den läcker ut till fel personer. Därför kan uppgifter som knyter informationen till en person att anonymiseras. Detta kan göras med pseudonymer. Pseudonymer tar bort den direkta kopplingen mellan datat och identiteten men tillåter att kopplingen återskapas vid behov, men bara om man vet vad pseudonymen representerar.[18, 1]

5.1.2 Personuppgiftslagen

Personuppgiftslagen (PuL) bygger på EU-direktiv och trädde i kraft 1998. Den innehåller regler för hur personuppgifter får behandlas och är till för att människors personliga integritet inte ska kränkas vid behandling av personuppgifter[5]. Riksdagens hemsida har en lista på författningar och lagar där beteckningen "behandling" definieras:

> *"Varje åtgärd eller serie av åtgärder som vidtas i fråga om personuppgifter, vare sig det sker på automatisk väg eller inte, t.ex. insamling, registrering, organisering, lagring, bearbetning eller ändring, återvinning, inhämtande, användning, utlämnande genom översändande, spridning eller annat tillhandahållande av uppgifter, sammanställning eller samkörning, blockering, utplåning eller förstöring."*
> Svensk författningssamling, [8]

Personuppgifter är all slags information som direkt eller indirekt kan knytas till en fysisk person som är i livet. Om man till exempel vill publicera ett foto på vänner, arbetskamrater, klasskamrater eller liknande kan man behöva fråga personen/personerna i fråga om samtycke att bilden läggs ut på Internet. Med samtycke menar man att personen i fråga med eller utan tillfrågan godtar behandlingen av personuppgiften[5].

5.1.3 Sammanfattning

I vissa webbapplikationer/webbtjänster lagras personlig information om kunder och/eller användare. Anledningen till att informationen sparas kan till exempel vara för att underlätta när man handlar på Internet. För att man ska slippa skriva in adressuppgifter varje gång så räcker det att logga in så finns adressen redan ifylld i beställningen. Detta gör att vissa krav ställs på webbapplikationen att inte läcka ut personuppgifter utan personens godkännande. Därför kan detta vara en ganska viktig punkt att testa.

5.2 Autentisering

Om man har ett system eller en applikation som har flera olika användare är det högst troligt att man har ett behov av någon form av autentisering. Detta för att kunna skilja på användarna och avgöra vad den aktuella användaren får göra. Man brukar dela upp autentisering i tre olika kategorier eller faktorer:

- Något användaren vet (lösenord)
- Något användaren har (smart card, kreditkort, kod-dosa)
- Något användaren är (fingeravtryck, iris, ansikte)

I vissa fall används dessa faktorer tillsammans för att få en högre säkerhet.

5.2.1 Lösenord

Den enklaste och absolut vanligaste metoden för autentisering är kombinationen av användarnamn och lösenord som grundar sig i att användaren vet det hemliga svaret på frågan "Vad är lösenordet?"[25].

5.2.1.1 Krav på val av lösenord

För att kombinationen av användarnamn och lösenord ska vara effektiv är det viktigt att ha lösenord som är svåra att gissa eller knäcka. Samtidigt måste användarna komma ihåg sina lösenord för att kunna logga in. Optimalt för lösenordssäkerheten skulle vara om applikationen genererar ett slumpat lösenord som består av en kombination av bokstäver, siffror och specialtecken i godtycklig ordning och längd. Nackdelen med detta skulle vara att användarna skulle ha svårt att komma ihåg lösenordet och troligtvis skriva det på en lapp vilket i sig skulle vara ett hot mot lösenordssäkerheten. Detta är en svår avvägning som man bör tänka på när man skapar lösenorden. En undersökning som genomfördes i storbrittanien visar att 12% av användare har ordet `"password"` som lösenord. Detta är ett väldigt osäkert lösenord eftersom det är så vanligt och för att det är lätt att gissa[25].

Det finns olika sätt för att knäcka lösenord. Det lättaste sättet är att gissa sig fram till lösenordet. Ett vanligt sätt att gissa lösenord är genom en så kallad "dictionary attack" där man använder sig av en lista med ord (en ordbok, eng. dictionary)[2].

När ett lösenord ska väljas rekommenderas följande [25]

- Lösenordet bör vara minst 8 tecken långt
- Specialtecken bör användas för att motverka enkla "dictionary attacks". Specialtecken kan vara till exempel: !, @, &, +, #, § med flera.
- Försök undvika vanliga mönster som stor förstabokstav och specialtecken endast i slutet, då detta kan vara lätt att förutse när man ska knäcka lösenordet.

- Skapa gärna meningar och använd första bokstaven i varje ord i lösenordet. Till exempel: “Jag vill ha ett bra lösenord som få kan knäcka” kan bli “jvheblsfkk” som inte är ett uppenbart ord. Detta exempel kan utökas genom att man byter ut några av bokstäverna mot specialtecken eller siffror. Då skulle det kunna bli till exempel: “jvh3B1$fkk”.

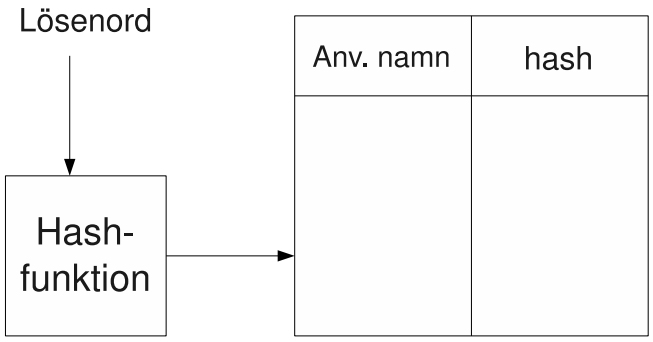
5.2.1.2 Kryptering av lösenord

För att kunna använda sig av användarnamn-lösenord metoden måste lösenordet sparas någonstans så att det kan jämföras med det lösenordet som skrivs in vid inloggning. Det enklaste är att bara lägga in lösenordet som klartext i en lista, fil eller databas tillsammans med användarnamnet. Då blir det lätt att leta upp användarnamnet och jämföra med motsvarande lösenord. Ett problem med att lägga lösenordet som klartext är att vem som helst som har tillgång till databasen, filen eller listan kan läsa det och sedan använda det för att logga in och möjligtvis göra skada. För att motverka detta bör man inte spara lösenordet i klartext utan kryptera det på något sätt.

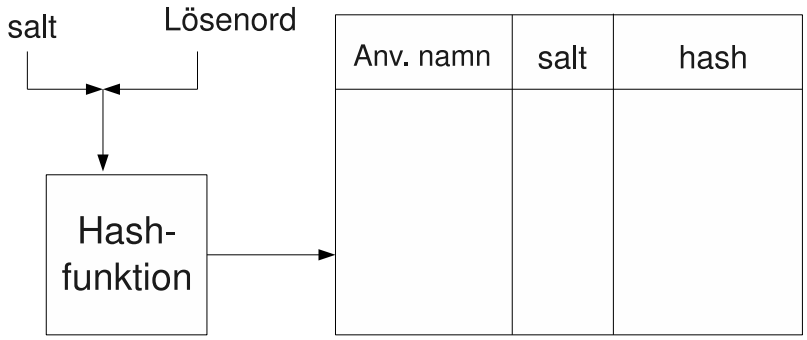
Det finns olika sätt att kryptera lösenordet på. Ett vanligt sätt är att lägga in en hash av lösenordet istället för klartext. Eftersom hashning är envägskryptering (läs mer om kryptering i kap. 5.4) så kan inte lösenordet återskapas från hashen och förblir därmed hemligt. En nackdel med detta skulle vara att om användaren glömmer bort sitt lösenord så kan ingen återskapa det, vilket innebär att man måste lägga in ett nytt lösenord för att ersätta det glömda.

I figur 5.1 illustreras hur lösenordet vid skapande går till en hashfunktion vars resulterande hashvärde läggs i anslutning till användarnamnet för att sedan kunna jämföras vid autentisering.

För att ytterligare ökar säkerheten på lösenordet kan man använda sig av så kallad salt. Du kan läsa mer om salt i kap. 5.4.2 på sida 25. I figur 5.2 illustreras hashningen av lösenordet med hjälp av salt



Figur 5.1: Hashning av lösenord



Figur 5.2: Hashning av lösenord med salt

5.2.2 Smart Cards

Smart card, eller smarta kort ingår i kategorin "Något som användaren har" och ger skydd genom att endast den eller de användare som har rätt kort kommer åt resurserna. De är små, lätta att ta med sig och lätta att använda vilket är viktigt för att användarna ska acceptera korten. Tidigare har smart cards varit utsatta för vissa attacker men nu för tiden anses säkerheten i korten vara relativt hög [21]. En del banker använder sig nu för tiden av smart cards så att kunderna ska kunna logga in för att göra sina bankaffärer över Internet eller göra vanliga inköp i affärer. Oftast används korten tillsammans med en PIN-kod för att inte vem som helst ska kunna ta kortet och använda det.

5.2.3 Biometri

Biometri är ett samlingsnamn som går under faktorn "Något som användaren är", alltså vilka fysiska attribut en person har. Dessa attribut är oftast tillräckligt unika för att kunna särskilja personer och därför kunna användas för autentisering. Igenkänning av människor med hjälp av dess fysiska attribut som till exempel utseende, röst och lukt har funnits lika länge som mänskligheten själv. Även imitation har funnits länge, både för legitima och ickelegitima orsaker, för att efterlikna någon person[2].

Fingeravtryck

Fingeravtryck avläses med någon sorts sensor som skapar en "elektrisk bild" som genom en algoritm omvandlas till ett templat för att senare kunna jämföras med andra templat. Det som läsaren tittar på är upphöjningar och nersänkningar som bildar olika kombinationer och formationer i avtrycket. Det finns flera olika sätt att avläsa fingeravtrycken på en person. Tre av sätten är optiskt, ultraljud och kapacitiva sensorer.

De optiska sensorerna (kamerorna) är relativt stora och otympliga, vilket gör dem olämpliga för till exempel hemmabruk. Dessuton finns det en risk att ett avtryck från en tidigare avläsning finns kvar på avläsningsytan. Då kan en illvillig person återanvända det tidigare avtrycket för att autentisera sig som den användaren. Därför bör ytan torkas av efter användning [2].

När det gäller ultraljudssensorer används principerna för sonografi, på liknande sätt som vid ultraljudsundersökning på till exempel gravida kvinnor. Mycket högfrekvent ljud skickas ut med hjälp av piezoelektriska material. Ljudvågorna tränger genom hudens yttersta lager (överhuden) och reflekteras mot läderhuden. De reflekterade ljudvågorna fångas upp av piezoelektriska material och då kan en elektrisk bild skapas som senare omvandlas till ett templat[29].

Kapacitiva sensorer läser av fingeravtrycket genom att varje pixel i avläsaren tillsammans med läderhuden bildar en kondensator. För att mäta skillnaderna mellan åsar och

dalar jämförs kapacitansen för de olika pixlarna. Sedan bildas en elektrisk bild av pixlarna som översätts till ett templat[29].

Röst

Röstigenkänning (speaker/voice recognition) får inte förväxlas med taligenkänning (speech recognition/verbal information verification). En viktig detalj som skiljer dem åt är att vid röstigenkänning analyseras karakteristiska egenskaper hos en persons röst, oftast för identifikation, medan taligenkänning tolkar innebörden i det som sägs[2, 36].

Taligenkänning går alltså att använda oberoende av vem som talar in i mikrofonen. Det enda som är intressant är innehållet i orden eller meningarna som sägs. Detta kan användas som substitut mot lösenord för personer som till exempel inte kan använda ett vanligt tangentbord. Istället för att skriva in lösenordet så kan de tala in det.

Röstigenkänning är till för att identifiera och/eller verifiera vem den som talar är. Systemet som ska autentisera användaren får först "lära sig" frasen eller ordet som ska vara kontrollvärdet. Sedan när användaren vill komma åt systemet så får han eller hon säga den tidigare överenskomna frasen[2].

Ögon

Mönstret som finns i ögats iris är unikt för varje person [2]. Detta kan därför användas för att autetisera personer på ett säkert sätt. För att kunna avläsa det detaljrika mönstret på iris används en kamera med hög upplösning. Bilden analyseras och ett templat skapas för att kunna jämföras med andra avläsningar Irisscanning kan användas av alla som har ett öga, vilket gör att tekniken har en bred grupp med möjliga användare [30].

Ett annat sätt att använda ögat för autentisering är att scanna av näthinnan [2, 30, 34]. Mönstret för blodkärlen som förser näthinnan med blod är så komplext att varje individ har ett unikt mönster. Till och med enäggstvillingar har skilda mönster i näthinnan. För att scanna näthinnan använder man en mycket svag infraröd laser och lyser in i ögat på

näthinnan. Reflektionerna som blir från blodkärlen kan skiljas från resten av ögat och det skapas en elektrisk bild som sedan används för att verifiera användaren. Denna metod är inte lika utbredd som irisscanningsmetoden men anses vara en av de säkraste biometrimetoderna hittills. Förespråkare för näthinnescanning påstår att dess felmarginal är en på miljonen[34].

5.2.4 Sammanfattning

Nu för tiden använder allt fler Internet för att utnyttja tjänster som erbjuds där. Många sidor på Internet använder sig av personlig information eller företagsinformation. För att inte sådan information ska läsas eller ändras av obehöriga använder man sig ofta av inloggningsfunktioner, så att endast auktoriserade användare kommer åt informationen. Lättaste lösningen är att använda sig av lösenord. Om man behöver extra hög säkerhet kan man använda sig av till exempel biometri eller smart cards. En nackdel med dessa är att man måste ha någon hårdvara för att kunna använda dem. Fingeravtrycksläsare är den enda biometriska autentiseringsmetoden som är någorlunda utspridd och användbar, mycket tack vare dess relativt enkla uppbygnad och kompakta format. Till exempel finns det flera bärbara datorer med inbyggd fingeravtrycksläsare.

5.3 Kommunikation mellan klient och server

5.3.1 Internets uppbyggnad

Internet bygger till stor del på klient/server-arkitekturen, det vill säga att det finns servrar som tillhandahåller material och tjänster, samt klienter som begär material eller tjänster av dessa servrar. I alla fall där en tjänst skall utföras eller information skall utbytas krävs kommunikation mellan parterna. När man pratar om Internet på en djupare nivå brukar man dela upp det i olika lager. Antingen enligt OSI-modellen, som oftast används i akademiska sammanhang, eller enligt TCP/IP-modellen, som anses vara mer praktiskt tillämpningsbar.

TCP-modellen anges ofta med 4 lager, men i vårt fall använder vi 5 lager, enligt Kurose och Ross[9]. TCP/IP-modellens lager är:

- Applikationslagret
- Transportlagret
- Nätverkslagret
- Länklagret
- Fysiska lagret

Det fysiska lagret ansvarar för hur kommunikationen i mediet ska kodas, till exempel vilka frekvenser eller vilka spänningar som används i en kabel för att förstås av sändare/mottagare i ändarna av mediet. Övriga fysiska aspekter såsom kabel- och kontaktutformning specifieras också i det här lagret.[9]

Länklagret ansvarar för kommunikation mellan två noder som är direkt kopplade till varandra, till exempel två switchar. Lagret ansvarar för accesskontroll för att minska antalet kollisioner i ett delat media. Det kan också ansvara för flödeskontroll och felhantering.[9]

Nätverkslagret ansvarar för adressering i ett nät och även hanteringen av trafikproblem genom val av väg genom nätet. Detta görs med hjälp av routing-protokoll för vägval, och med hjälp av IP - Internet Protocol, för adressering. IP garanterar inte att information kommer fram till mottagaren, utan ger bara möjligheten för kommunikation över ett nätverk. På nätverkslagret finns även en teknik som kallas IPsec, en teknik för säker överföring mellan två änd-system.[9]

Transportlagret ansvarar för leverans av data mellan två processer på hosts i ett nätverk och är därför på en högre nivå än nätverkslagret som bara skickar data mellan två hosts. Transportlagret ansvarar även för flödeskontroller för att undvika överbelastning i nätverket, felkontroll av data och även kontroller av att data anländer i samma ordning

som den skickades. Detta görs vanligtvis med protokollet TCP - Transmission Control Protocol. TCP fungerar genom segmentering av data som sedan numreras och varje paket får en hash-summa för att möjliggöra felkontroller. Numreringen används för att kontrollera ankomst av data och för att kunna leverera paketen i rätt ordning. TCP kontrollerar också antalet tappade paket för att kunna skicka om förlorad data och även för att kontrollera om sändningshastigheten behöver sänkas för att förhindra överbelastning. TCP använder sig av portnummer för att identifiera vilken process som skall ha datan. Om fullständig leverans och leveransordning inte är viktigt kan man istället använda protokollet UDP - User Datagram Protocol. UDP tillhandahåller en minimal service jämfört med TCP. UDP lägger bara till mottagarportnummer och sändarportnummer, en checksumma så att mottagaren kan se om bitfel har introducerats, samt längden för paketet, innan det skickas till nätverkslagret.[9, 37, 38]

Applikationslagret innehåller de protokoll som applikationer, till exempel en webbserver eller en browser, använder. Exempel på protokoll kan vara HTTP, SSL, FTP, DHCP eller IRC. De olika protokollen har olika funktioner. Till exempel har HTTP och FTP som uppgift att överföra filer, medans DHCP förenklar tilldelning av IP-adresser och IRC tillhandahåller funktioner för chatt. SSL erbjuder en service som tillåter en klient att verifiera servern, samt funktioner för att kryptera kommunikationen mellan dem [9].

Av de fem lager som här har beskrivits kan man se att bara två lager har något protokoll som erbjuder kommunikationssäkerhet. Detta är IPsec på nätverkslagret och SSL på applikationslagret. De här protokollen fungerar på olika sätt och ger skydd på olika nivåer och det kan därför vara passande att jämföra dem med varandra.

5.3.1.1 IPsec

IPsec är som sagt ett nätverkslagerprotokoll, och det erbjuder tjänster för säker överföring mellan änd-punkter i ett nätverk. Eftersom det är ett protokoll på låg nivå så kan det hantera alla sorters trafik från högre lager. IPsec har två tjänster som den erbjuder och

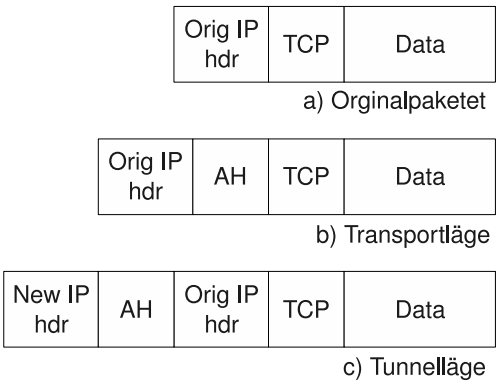
dessa är[9]:

AH - Authentication Header

ESP - Encapsulated Security Package

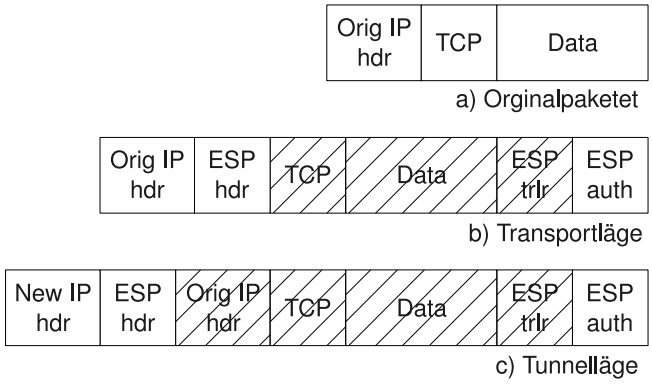
AH erbjuder autentisering och integritet av paketets innehåll medans ESP erbjuder både autentisering, integritet och konfidentialitet. Det här görs genom att lägga till headers till paketet, och i ESPs fall så krypteras även datan. Båda dessa tjänster kan köras i två lägen, Transport eller Tunnel[9].

Om du kör AH i Transportläget så erbjuds autentisering direkt mellan två änd-noder i ett nätverk, så kallad end-to-end. Om du istället använder Tunnelläget så erbjuds autentisering mellan en slutnod och en punkt någonstans innan den slutnod du vill prata med. Till exempel ett företags brandvägg. Detta kallas för end-to-intermediate. Skillnaderna mellan hur IP-paketen ser ut med AH i de olika lägena kan ses i figur 5.3.



Figur 5.3: IPsec AH för IPv4

Transportläget för ESP kan tillhandahålla kryptering samt autentisering mellan två änd-noder. Om någon eller båda av noderna inte stödjer ESP så kan Tunnelläget användas istället, antingen mellan en änd-nod och en gateway eller mellan två gateways. Detta har fler fördelar såsom att klienter inte behöver hantera belastningen av krypteringen, samt att färre nycklar behövs.

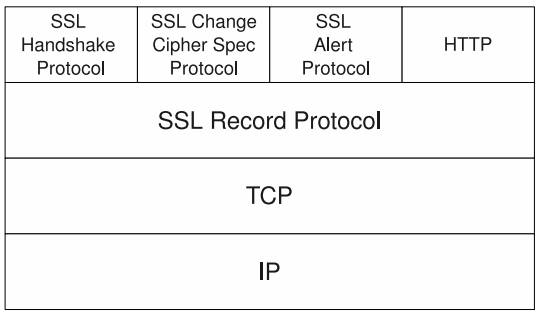


Figur 5.4: IPsec ESP för IPv4

När man vill sätta upp en IPsec-koppling så behöver man gå igenom två olika faser. I fas 1 så sker autentisering av noderna och skapande av krypteringsnycklar för att skydda fas 2. I fas 2 så förhandlar parterna om vilken krypterings och autentisieringsalgoritm som skall användas. Efter dessa två faser så är fortsatt trafik krypterad.

5.3.1.2 TLS och SSL

SSL uppfanns 1994 av Netscape, och utvecklades senare till version två och även version 3 år 1996. SSL v3 är den version av SSL som används mest idag. SSL utvecklades för att öka säkerheten i kommunikationen över TCP-protokollet. SSL kan sägas ligga mellan applikationslagret och transportlagret. I figur 5.5 visas de protokoll som ingår i SSL och deras relation till omgivande protokoll.



Figur 5.5: SSL Protokoll-stack

SSL Record Protokollet tillhandahåller konfidentialitet och integritet för meddelandet. Vad det gör först är att fragmentera ner meddelandet som skall skickas i mindre bitar. Sedan komprimeras fragmentet eventuellt, detta är valfritt. En MAC - Message Authentication Code beräknas på meddelandet och läggs sedan till själva meddelandet. Det är detta som ger en funktion för att verifiera integriteten av meddelandet. Efter detta så krypteras meddelandet med någon av flera möjliga krypteringsalgoritmer.

Till sist läggs det till ett antal SSL RP-headers. Dessa headers är:

Innehållstyp - Berättar vilka sorters högre protokoll som använder datan i paketet

SSL-huvudtyp - Anger vilken huvudversion av SSL som används, till exempel 3 för v3.1 eller v3

SSL-undertyp - Anger undertyp av SSL som används, till exempel 0 för v3 eller 1 för v3.1

Till sist finns även ett fält som beskriver längden av datan[19, 16].

Change Cipher Spec Protokollet är ett mycket enkelt protokoll, som bara består av ett enda meddelande. Meddelandets uppgift är att meddela att tillståndet skall uppdateras för den aktuella kopplingen[19].

Alert Protokollet används för att skicka varningsmeddelanden mellan de olika parterna. Meddelandena kan vara av två olika typer, antingen av Fatal typ, eller av Varningstyp. Om ett meddelande av fatal typ tas emot så avslutas kopplingen. Meddelandet innehåller också en kod som avslöjar den specifika anledningen till meddelandet. Det finns ett flertal anledningar som ger upphov till att ett meddelande skickas, till exempel, att certifikat saknas, MAC för ett meddelande är fel eller att en förhandling om parametrar inte kunde genomföras[19].

Handshake protokollet är det SSL-protokoll som är mest komplext. Det består av ett antal meddelanden som används för att förhandla vilka krypteringsalgoritmer, MAC-

algoritmer och krypteringsnycklar som skall användas, samt meddelanden för att kunna autentisera varandra[19].

5.3.2 Sammanfattning

Alla webbapplikationer kommunicerar med användare över någon form av nätverk. Det fysiska nätverket kan se olika ut och kan vara både trådlöst och trådbundet och kommunikationen kan möjligtvis avlyssnas. För att förstå vilka säkerhetsaspekter man står inför när information skickas över nätverket är det viktigt att ha en förståelse för hur nätverket fungerar. Det finns inga garantier för att information skickas skyddat per automatik, utan istället är det upp till den som gör applikaionen att implementera teknologier som kan ge ett fullgott skydd. Vilket skydd man skall välja beror på vilka krav man behöver uppfylla och vilka begränsningar man har. Om man inte implementerar något slags skydd så kan det resultera i att obehöriga kan läsa av trafiken när den skickas mellan klient och server och sedan utnyttja den på sätt som kan skada.

5.4 Algoritmer för kryptering

Det finns ett antal algoritmer att välja på när man ska kryptera något data. Här kommer en beskrivning på några av dessa.

5.4.1 Hash

Ett hashvärde genereras genom att ett meddelande av variabel längd skickas till en hashfunktion som returnerar ett värde med fast längd. Hashvärden representeras ofta som hexadecimala tal och kan vara användbara i flera olika fall. Ett användningsområde är att använda hashvärdet som ett slags fingeravtryck av en mängd data som kan vara en fil, text eller vilken data som helst för att försäkra sig om att inget har ändrats på något sätt. Detta möjliggörs tack vare hashalgoritmers egenskap att en stor del av hashvärdet ändras även om bara lite av ursprungliga meddelandet ändras.

Hashvärdena i tabellerna 5.1 och 5.2 är genererade med hjälp av en hemsida där man kan skriva in den text som man vill hasha. Adressen dit är `http://www.fileformat.info/tool/hash.htm`

Tabell 5.1: Jämförelse av två hashvärden

Sträng	Hashvärde
The quick brown fox jumps over the lazy dog	`2fd4e1c67a2d28fced849ee1bb76e7391b93eb12`
the quick brown fox jumps over the lazy dog	`16312751ef9307c3fd1afbcb993cdc80464ba0f1`

I tabell 5.1 ses en jämförelse av hashvärden från två strängar som nästan är identiska. Det enda som skiljer strängarna åt är att den ena strängen börjar med stor bokstav. Där ser man tydligt att hashvärdet ändras avsevärt jämfört med hur mycket som ändrats i meddelandet.

I tabell 5.2 kan man se skillnaden på genererat hashvärde för några olika hashfunktioner. Dessa hashfunktioner beskrivs närmare nedan.

Tabell 5.2: Olika hashvärden för ett meddelande

Meddelande	The quick brown fox jumps over the lazy dog
SHA-1	2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
SHA-256	d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592
SHA-384	ca737f1014a48f4c0b6dd43cb177b0afd9e5169367544c4- 94011e3317dbf9a509cb1e5dc1e85a941bbee3d7f2afbc9b1
SHA-512	07e547d9586f6a73f73fbac0435ed76951218fb7d0c8d788a309d785436bbb64- 2e93a252a954f23912547d1e8a3b5ed6e1bfd7097821233fa0538f3db854fee6
MD4	1bee69a46ba811185c194762abaeae90
MD5	9e107d9d372bb6826bd81d3542a419d6

För att hashfunktionen ska vara användbar ställs vissa krav[19]

- Funktionen ska kunna appliceras på data av obestämd storlek
- Funktionen genererar ett värde med fast längd
- För varje givet hashvärde ska det vara orimligt att beräkna meddelandet som genererat det värdet Algoritmer med denna egenskap brukar kallas för envägsalgoritmer.

- Det ska vara relativt lätt att beräkna hashvärdet så att både mjukvaru- och hårdvaruimplementering möjliggörs

- Givet ett meddelande x ska det vara orimligt att beräkna ett meddelande y så att deras hashvärden är identiska. Detta kallas ibland svag kollisionsundvikelse (eng. *weak collision resistance*)

- Det ska vara orimligt att hitta två meddelanden så att deras hashvärden är identiska. Detta kallas ibland stark kollisionsundvikelse (eng. *strong collision resistance*)

Med orimligt menas här att det ska krävas så många beräkningar att man statistiskt sett skulle behöva beräkna under en väldigt lång tid för att kunna hitta svaret. Till exempel dygnet runt i femhundra år.

5.4.1.1 SHA

SHA (Secure Hash Algotithm) utvecklades av NIST (**N**ational **I**nstitute of **S**tandards and **T**echnology). 1995 kom en reviderad version som allmänt kallas SHA-1 och är specifierad i RFC 3174. Hashvärdet som kommer ut ur SHA-1 är 160 bitar långt. 2002 reviderades algoritmen igen och man definierade SHA-256, SHA-384 och SHA-512 som alla bygger på SHA-1 och ger hashvärden på respektive 256, 384 och 512 bitar. Dessa algoritmer ingår i familjen SHA-2 [35].

På grund av brister i SHA-1 agoritmen strävar NIST för att SHA-1 ska fasas ut och att man ska börja använda en av de mer komplicerade algoritmerna. Än så länge rekommenderas att man använder någon av algoritmerna i SHA-2 familjen men NIST har tänkt i förväg genom att påbörja utvecklingen av nya säkrare algoritmer. 2007 Utlystes en tävling som gick ut på att utveckla en ny kryptografisk hashalgoritm som kommer kallas SHA-3. Enligt planerna ska SHA-3 vara färdig att användas i slutet av år 2012. Fram tills dess kommer alla kandidater granskas av både experter och allmänheten för att man ska kunna välja en vinnande algoritm [12].

5.4.1.2 MD4/MD5

MD5 (Message-Digest Algorithm 5) tar emot data, eller meddelande, av godtycklig längd och ger tillbaka ett hasvärde med längden 128 bitar. Teoretiskt sett skulle det krävas 2^{128} försök för att hitta en kollision, det vill säga ett meddelande som resulterar i samma hashvärde som det tidigare. Även om man skulle kunna beräkna en hash varje nanosekund (10^{-9}s) så skulle det krävas tusentals miljarders miljarder ($\approx 10^{22}$) år. På grund av detta kan det kännas ganska säkert att använda MD5 men med hjälp av kryptografisk analys så har man upptäckt vissa egenskaper i algoritmen för MD5 som gör att det i praktiken går lättare att hitta kollisioner [13].

5.4.1.3 RIPEMD-160

RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) är en snabb kryptografisk hashalgoritm som har utvecklats ifrån MD4. Som namnet antyder så är längden på hashvärdet som ges 160 bitar. Den tar in ett värde med godtycklig storlek som delas upp för bearbetning. Algoritmen består i stort sett av två parallella versioner av MD4, med några förbättringar, som slås ihop i slutet av algoritmen [3].

5.4.1.4 Attacker mot hash

Dictionary attack

Vanligaste attacken mot till exempel hashade lösenord är att göra en "Dictionary attack" (ordboksattack). Den sortens attack går ut på att man har en lista med ord som hashas ett och ett och jämförs med hashvärdet för lösenordet. Om hashvärdena är lika så har man hittat lösenordet. Denna attack är vanlig eftersom många använder vanliga ord som lösenord för att lättare komma ihåg det.

Brute force attack

En annan attack är "Brute force attack" där attackeraren provar alla tänkbara kombinationer av bokstäver, siffror och symboler. Alla dessa kombinationer hashas och jämförs med hashen för lösenordet. Detta är en mycket tidsödande attack då det är väldigt många kombinationer att gå igenom. Antalet kombinationer ökar exponentiellt med antalet tecken i lösenordet. För att räkna ut antalet kombinationer som finns för varje teckenantal i lösenordet kan man använda formeln $x = n^m$ där x är antalet kombinationer, n är antalet tecken i teckenuppsättningen som ska undersökas och m är antalet tecken i lösenordet.

Tabell 5.3: Antalet kombinationer för teckenuppsättningarna [a-z] och [a-zA-Z]

m	a-z (gemener)	a-zA-Z (gemener+versaler)
1	$26^1 =$ 26	$52^1 =$ 52
2	$26^2 =$ 676	$52^2 =$ 2 704
3	$26^3 =$ 17 576	$52^3 =$ 140 608
4	$26^4 =$ 456 976	$52^4 =$ 7 311 616
5	$26^5 =$ 11 881 376	$52^5 =$ 380 204 032
6	$26^6 =$ 308 915 776	$52^6 =$ 19 770 609 664
7	$26^7 =$ 8 031 810 176	$52^7 =$ 1 028 071 702 528
8	$26^8 =$ 208 827 064 576	$52^8 =$ 53 459 728 531 456
9	$26^9 =$ 5 429 503 678 976	$52^9 =$ 2 779 905 883 635 712

I tabell 5.3 listas antalet kombinationer för olika ordlängder om man skulle begränsa attacken till att bara undersöka det engelska alfabetet. Där kan man tydligt se att antalet möjliga kombinationer ökar dramatiskt när man ökar ordlängden.

Rainbow Tables

En tredje attack är att använda så kallade "Rainbow Tables" [33]. Då har man i förväg skapat listor eller tabeller med alla möjliga kombinationer och alla de orden associeras till dess hashvärde. På detta sätt räcker det att man letar efter hashvärdet som matchar lösenordets hashvärde så kan man med hjälp av associationen se vilket ord som motsvarar

lösenordet. Fördelen med denna attack är att man inte behöver räkna ut hascharna för alla kombinationer varje gång. Dessutom finns det ofta färdiga tabeller att ladda ner eller beställa från någon som har genererat en tabell. En nackdel med rainbow tables är att filerna som innehåller tabellerna snabbt blir mycket stora.

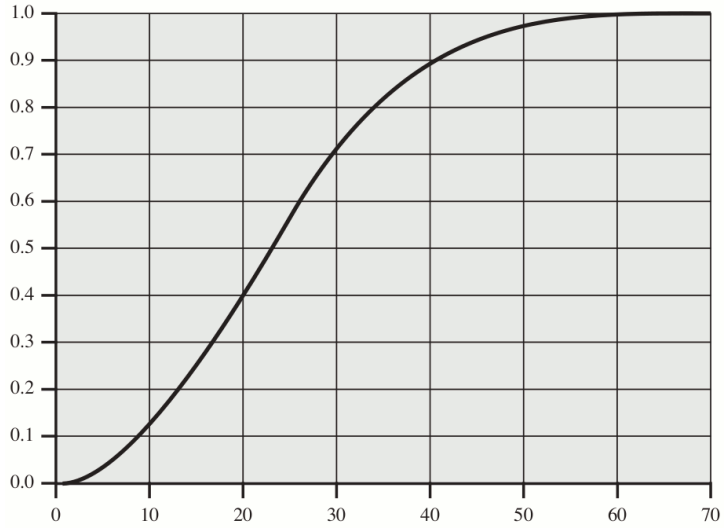
Vi har hittat en sida `http://tbhost.eu/` där man kan ladda ner rainbow tables för flera olika teckenuppsättningar. Till exempel finns tabeller för SHA-1 med teckenuppsättningen [a-z], alltså gemenerna a-z och mellanslag, och med lösenordslängder på 1-9 tecken. Dessa tabeller är uppdelade i fyra olika delar där varje del är på 11.88 GB och 44 filer vardera. Då har de ändå använt ett speciellt filformat som gör att storleken minskar med 50%.

Birthday attack

Birthday attack (Födelsedagsattack) har fått sitt namn av födelsedagsparadoxen som bygger på att sannolikheten, för att något par i en slumpmässigt utvald grupp människor har samma födelsedag, är över 50%[19]. Många tror att det skulle behövas 183 personer, alltså hälften av årets dagar, i gruppen för att sannolikheten skulle vara 50%. Men sannolikhetsläran ger bevis för att det endast behövs en grupp på 23 personer för att sannolikheten ska vara 50,73%. En anledning till att detta resultat är så förvånande är för att man ofta utgår ifrån en person. Då blir sannolikheten låg att den personen har samma födelsedag som någon annan person i gruppen. Om man däremot tittar på gruppen (23 personer) i helhet så finns det enligt kombinatoriken $(23*(23-1))/2 = 253$ olika par [19], vilket medför att sannolikheten är högre än förväntat. I figur 5.6 visas ett diagram över sannolikheten i birthday-paradoxen. Där ser man att kurvan stiger ganska fort och sannolikheten närmar sig 100% redan vid 60 personer.

5.4.2 Salt

Ett sätt att skydda sig mot framför allt dictionary attacks och rainbow tables är att använda sig av ett saltvärde, som är ett slumpat värde som man lägger tillsammans med



Figur 5.6: Birthday-paradoxen

den data (meddelandet) som ska hashas. Oftast används salt i kombination med lösenord för att försvåra för illasinnade personer att knäcka lösenordet. Salt skyddar mot attacker eftersom det gör att användandet av värden som har räknats ut i förväg är ogenomförbart.

Saltvärdet är "publikt", alltså inte undangömt eller krypterat på något sätt utan helt synligt och oftast lagrat i anslutning till lösenordet. Detta gör att det är relativt lätt att få tag på saltvärdet. För att öka effektiviteten med salt och försvåra för någon som fått tag på saltvärdet att knäcka lösenordet, kan man kombinera saltvärdet med lösenordet på ett icke standardiserat sätt. Till exempel om man har en funktion "hash(meddelande)" som räknar ut hashvärdet för ett specifikt meddelande och "." innebär sammanslagning av strängar så att hash('hej' . 'san') $\Leftrightarrow$ hash('hejsan') så kan man kombinera på olika sätt och till och med använda specialtecken mellan värdena.

- hash(salt . lösen)
- hash(lösen . salt)
- hash(salt . '-' . lösen)
- hash('-' . salt . '-' . lösen . '-')

- ...

I listan ovan visas exempel på olika kombinationer mellan lösenord, salt och specialtecken. Genom att kombinera värdena på olika sätt så försvårar man ytterligare för attackeraren att knäcka lösenordet. Om attackeraren inte vet hur salt och lösenord kombineras finns det i stort sett bara en sorts attack kvar, brute force. Eftersom man då har lagt till ett saltvärde till lösenordet så är det hashade värdet troligtvis ganska lång (>10 tecken), vilket gör att brute force attacken är ogenomförbar på grund av antalet beräkningar som måste göras.

Ibland används salt också för att olika användare ska kunna använda samma lösenord. Om man inte skulle använda salt och olika användare hade samma lösenord så skulle de därmed ha samma hash på sitt lösenord. I så fall skulle användare A kunna titta i filen eller databasen där hasherna för allas lösenord ligger, ajämföra med sin egen hash och dra slutsatsen att användare B har samma lösenord som A har. Då kan A logga in med B's användarnamn, komma åt dess resurser och kanske till och med använda kontot för att göra olagliga eller icke tillåtna saker.

5.4.3 DES

DES, Data Encryption Standard, är en standard framtagen för publikt användande av US-As regering. Den togs i bruk som standard 1976[16]. DES är en block-krypteringsalgoritm som använder sig av både substitution och transposition[2]. Substitution innebär att man byter ut tecken eller bitar inom ett block, något som karaktäriseras tydligt av Caesar-algoritmen där A ersätts med D och D ersätts med G, det vill säga att man byter ut ett tecken mot tecknet tre positioner längre fram i alfabetet. Moderna algoritmer använder dock mer avancerade metoder för substitution än vad Caesar-algoritmen gör. Transposition innebär att man byter plats på tecken eller bitar inom blocket, till exempel genom att betrakta blocket som en sträng, och sedan skifta positionerna på varje tecken en eller flera positioner framåt, samt låta det som övergår slutet på strängen rotera tillbaka till början

på den. Att DES använder både substitution och transposition är till fördel för algoritmen eftersom det gör den komplex. Algoritmen repeterar dessa tekniker 16 gånger, vilket gör det mycket svårt att finna samband mellan input och output, något som är mycket viktigt för styrkan av krypteringen.

DES svaghet är att den använder en väldigt kort nyckel på bara 56 bitar. Detta var tillräckligt förr i tiden, men den kraftiga ökningen av beräkningskraften i dagens datorer har gjort att 56 bitar idag helt enkelt inte är tillräckligt. DES är tyvärr låst till nyckellängden, vilket gjort det svårt att förbättra algoritmen. Man har försökt lösa detta genom att att göra nya versioner av DES som använder dubbel och trippel kryptering. Den första av dessa av dessa är Double DES. Den fungerar så att man använder två nycklar och helt enkelt krypterar data två gånger i följd. Det har dock visats att detta inte ger så mycket större säkerhet än vanlig DES. Bara en fördubbling av mängden arbete som krävs för att knäcka algoritmen. Man har däremot funnit att, om man krypterar tre gånger och vid den första och sista krypteringen använder en och samma nyckel, vilket man gör i Triple DES, så förlängs nyckellängden till motsvarande 112 bitar[16]. 112 bitar är signifikant starkare än 56 bitar, men eftersom dagens datorer utvecklas mycket fort kan även denna längd vara för kort snart. Därför rekommenderas att man använder AES, som beskrivs nedan.

5.4.4 AES

1997 utlyste NIST (National Institute of Standards and Technology) en tävling som gick ut på att utveckla ett nytt krypteringssystem som skulle kunna ersätta DES [16]. Några av kraven på algoritmerna som skulle användas var:

- Publika
- Gratis att använda i hela världen
- Symmetriska block-krypteringsalgoritmer för block på 128 bitar
- Kunna användas med nycklar på 128, 192 och 256 bitar

Alla bidragen granskades noggrannt av både experter och allmänhet och till slut fick man fram en vinnare. Den vinnande algoritmen skickades in av två holländska kryptografer och kallades för Rijndael som är en sammansättning av skaparna som heter Vincent Rijmen och Joan Daemen. De bidragen som var kvar till sista urvalet hade lika stark säkerhet som vinnaren men Rijndael ansågs effektivare i sin beräkning. Rijndael är en snabb algoritm som lätt kan implementeras på en enkel processor. Trots sin starka matematiska grund använder den främst substitution, transposition, XOR och adderingsfunktioner. Algoritmen blev döpt till AES (Advanced Encryption Standard) och anses vara en av de säkraste symmetriska krypteringsalgoritmer som finns i dagsläget. Eftersom dess holländska skapare inte har någon koppling till NSA (National Security Agency) eller någon annan del av USA's regering finns inga misstankar om att de har lagt in någon svaghet eller bakdörr i algoritmen. Trots att algoritmen är någorlunda lätt att beskriva så ligger det grundligt genomtänkta matematiska teorier bakom varje steg[16].

5.4.5 Sammanfattning

De algoritmer som tagits upp här är långt ifrån alla som finn men är några av de mest kända. Det kan vara viktigt att välja en algoritm som man inte har hittat några sårbarheter i än. Till exempel så är det inte rekommenderat att använda MD5 eftersom den kan knäckas relativt lätt. Speciellt nu för tiden med den beräkningskraft som finns tillgänglig.

5.5 SQL-injektion

I fall där man på en webbsida tar emot data och information från en användare så måste denna granskas och eventuellt rensas från information som kan vara skadlig. Om man inte gör det så kan det öppna för attacker där resultaten kan vara att information läcker eller läggs till i databasen eller att någon kan logga in utan att veta om ett korrekt lösenordet. Att mata in information mot en applikation på ett sätt som injicerar illegala kommandon i de legala SQL-förfrågningarna kallas att utföra en SQL-injektion. SQL-injektioner är en

väldigt vanlig svaghet i webbapplikationer. Gartner Group utförde en undersökning av 300 webbsidor och fann att majoriteten hade möjligheter till SQL-injektioner[7].

5.5.1 Injektionen

En injektion kan ske på grund av undermålig kontroll av inparamterar när en SQL-förfrågan genereras. Dessa inparametrar kan komma från flera olika källor, till exempel information som matas in i formulär på webbsidan, värden som lagras i kakor eller från servervariabler hämtade ur till exempel HTTP. All denna information kan modifieras av en användare och skall därför inte anses vara pålitbar[7]. En databasförfrågan som genereras i PHP skulle till exempel kunna se ut såhär:

```
$resultat = mysql_query('SELECT * FROM users WHERE user = "'.$_GET['user'].'"');
```

Om `$_GET['User']` inte har rensats på viktiga tecken så som `"` eller `=` så skulle den kunna innehålla strängen `" OR "a" = "a`. Detta skulle skapa SQL-förfrågan:

```
SELECT * FROM users WHERE user = "" OR "a" = "a";
```

`"a" = "a"` utvärderas alltid till sant och därför skulle alla poster i tabellen **users** returneras. Om samma problem existerar för lösenordsfältet i en SQL-förfrågan så öppnar det för inloggningar där man inte behöver veta om lösenordet.

Även andra attacker än otillåtna inloggningar kan ske genom SQL-injektioner. I vissa fall kan man i princip modifiera en förfrågan till att innehålla vilket SQL-kommando man vill. Exempelvis kan man lägga in kommandon som lägger till nya användare i databasen, som skriver ut stora delar av databasen, eller som tar bort delar eller hela databasen. Alla dessa kommandon kan användas på sätt som gynnar en attackerare eller förstör för de ansvariga för webbsidan.

5.5.2 Skydd mot injektioner

När man granskar input för att förhindra att en SQL-injektion kan ske så brukar man leta efter vissa tecken eller sekvenser av tecken. Att ta bort apostrof- och citat-tecken tar bort många attacker som går att utföra men inte alla. Även teckensekvenser som `--`, `/*` och `*/` eller `;` kan användas för att påverkan betydelsen av SQL-förfrågan och behöver därför tas bort. Till och med då kan man komma runt filtren genom att använda ASCII-värden för olika tecken som efter filtret evalueras till de tecken man behöver för att genomföra attacken[10].

Att skydda sig mot SQL-injektioner kan göras på flera olika sätt. Den vanligaste är att man gör manuella kontroller av det data som användaren tillför. Detta kan till exempel göras som i exemplet nedan, som är skrivet i java[20]:

```
String sanitizedName =
  replace(request.getParameter("name"),"'","''");
```

Koden innebär att alla "'" ersätts med "''" och därmed förstörs den syntaktiska meningen med det användarinmatade datat, och detta innebär att det går att skilja på användar-data och applikationsdata i en SQL-förfrågan. Nackdelen med detta är att det lätt blir fel i kontrollerna, på grund av den mänskliga faktorn. Det kan dessutom vara en komplicerad och tidskrävande uppgift att göra kontroller till alla inputs som finns i koden. Det kan dessutom vara svårt att avgöra vad som skall rensas och inte.

Automatiska kontroller kan också göras och ett känt exempel är MaqicQuotes i PHP som rensar all input. MagicQuotes har dock fått mycket kritik av olika anledningar. Problem med kompatibilitet mellan olika servrar kan uppstå om en server stöder MagicQuotes och en annan inte gör det, och skapar även mer arbete när data som gått genom MagicQuotes behöver bearbetas innan det kan användas igen till exempel vid utskrift till skärm. Detta eftersom "\" läggs till framför farliga symboler. MagicQuotes har i senare versioner av PHP tagits bort.

Perl stöder en typ av märkning av osäker data och kan även ge varningar om datan används utan att ha kontrollerats. Detta är bra eftersom det kan hjälpa till att ge en överblick över en kod med mycket användarinmatning. Dock så behöver utvecklaren själv skapa de tester som skall användas för att kontrollera data och detta ger fortfarande attackmöjligheter om utvecklaren gör fel.

SQLrand är ett annat sätt att skydda sig mot injektioner. SQLrand går igenom programkoden och kodar alla SQL-förfrågningar på ett visst sätt. Input kodas inte på samma sätt, och när sedan hela förfrågan skickas till databasen så går det genom en proxy som bara skickar vidare kommandon som är kodade på rätt sätt. På så vis filtreras eventuella injektioner ut.

Det finns många olika tekniker för förenkling och automatisering av upptäckt och förhindring av SQL-injektioner, utöver de nämnda finns till exempel AMNESIA, Java Dynamic/Static Tainting, SQLCheck, SQLGuard, JDBC-Checker, Security Gateway och SecuriFly[7].

5.5.3 Motivation

Det är väldigt vanligt att webbapplikationer använder en databas och det är webbapplikationen som matar in och begär information från databasen. Databasen kan inte utvärdera om ett kommando som kommer från webbapplikationen verkligen är utformat som applikationens skapare menat eller inte. Därför är det upp till webbapplikationen att se till att illegala kommandon inte kan skapas av applikationens användare. Beroende på vilka teknologier man använder för webbapplikationen så kan man skydda sig på olika sätt. Det är bra att använda inbyggda funktioner om det finns tillgängligt eftersom det minskar risken för undermåliga kontroller. Finns inte det, så skall manuella kontroller skapas, och då är det viktigt att kontrollerna utformas och utvärderas noggrant.

5.6 Cross Site Scripting

5.6.1 Definition

Cross Site Scripting, förkortat XSS, är en svaghet hos en webbsida som tillåter att en användare lägger in egen HTML-kod i den ursprungliga webbsidan på ett sådant sätt att koden kan påverka hur sidan visas för andra användare. På detta vis kan man ladda in script som körs på klientsidan, såsom, JavaScript, VBScript, XHTML, ActivX, Flash med mera, som exekveras på andra användares klienter och på så vis kan man få tag på information om den användaren eller dennes session[17].

5.6.2 Typer av attacker

5.6.2.1 Session highjacking

Detta är den vanligaste typen av attack som utförs med hjälp av script som exekveras på användarens dator. Scriptet används för att få tillgång till användarens Cookie, en textfil som kan innehålla information om användarens session mot sidan. Med den informationen kan attackeraren utge sig för att vara användaren och kapa dennes session. Detta kringår eventuell inloggning som annars krävs för att initiera en session.

5.6.2.2 Cookie Poisoning

Tillvägagångssättet i den här attacken är likt det föregående, det vill säga att man använder script, men i det här fallet är målet en cookie som innehåller någon slags information man vill modifiera. Ett exempel på det skulle kunna vara en cookie som innehåller information om ett tillstånd för en användare i en webbbutik. Om tillståndet beskriver hur mycket varorna i användarens kundkorg kostar, och man kan ändra på informationen i cookien, så kan det vara möjligt att lura webbutiken och få dem att ta betalt en annan summa än den riktiga, till exempel 0 kr. Omkring år 2001 publicerades ett antal attacker som tillåtit attackerarna att betala för lite genom cookie poisoning[15].

5.6.2.3 Malformed URL

En attackerare skapar en URL som är utformad på ett sätt som utför en XSS-attack och sprider sedan URLen som en länk till offer via till exempel mail, chatt eller andra webbsidor. När någon klickar på länken så initieras attacken. Säg till exempel att offret bara tillåter sin internetbank att köra script medan alla andra sidor blockeras. Om en attackerare lyckas injicera sitt script i bankens hemsida genom en URL så kommer det scriptet köras med bankens rättigheter.

5.6.2.4 IFRAME

Genom XSS kan man mata in HTML-kod som genererar en IFRAME, i denna IFRAME kan man sedan ladda in andra sidor som kan innehålla farliga script som till exempel kan utnyttja svagheter i webbläsaren, generera popups eller visa annan information.

5.6.2.5 DoS-attack

En denial of service-attack, dvs en attack som minskar eller förhindrar åtkomst till en tjänst kan utföras med hjälp av XSS. Om den utförs med XSS kan det vara genom att på en välanvänd sida som har en XSS-svaghet lägga in ett script som anropar en annan sida till exempel genom att ladda ner en stor bild. Användarna på siten med svagheten medverkar då omedvetet i en DoS-attack mot offer-sidan, genom att förbruka offrets resurser i form av bandbredd eller beräkningskraft.

5.6.3 Hur man kan motverka XSS-attacker

Om man ska minimera riskerna för att en webbapplikation ska innehålla svagheter som går att utnyttja för en XSS-attack så finns det tre riktlinjer man kan följa enligt OWASP. Alla tre går ut på att ställa olika krav på utformningen av all inmatning som en användare kan utföra. De tre riktlinjerna är[4]:

- Acceptera enbart kända legitima data
- Avvisa kända dåliga data
- Sanera dåliga data

Den första tekniken går ut på att man specifierar vad som är godkänt input. Allt som inte faller under den kategorin anses vara dåligt input och kasseras. Den här tekniken förutsättar att man kan specifiera vad som är legitimt. Exempel skulle kunna vara att bara tillåta siffror och bindestreck i ett telefonnummer samt sätta en maximal längd som motsvarar vad man kan förvänta sig för telefonnummer[4].

Om man vill använda de andra alternativen så behöver man specifiera vilket slags input som avvisas eller saneras. Detta görs svårare eftersom data kan utformas på olika sätt som kringgår kontrollen. Om man till exempel filtrerade bort alla “>” och “<” från input så kan det försvåra inmatningen av script, men man kan också använda URL-encoding, en teknik som används för att hantera problematiska tecken som till exempel Å, Ä, Ö och mellanslag i en URL. Denna teknik skulle kunna gå förbi filtret, men ändå så kan användarens browser tolka informationen till “>” eller “<”-tecken, och därmed fortfarande vara utsatt för en attack. Det finns också problem med hur en browser hanterar sidor som inte definierar vilken teckenkodning de använder. Om filtret letar efter ett tecken med en viss kodning och browsern har en annan teckenkodning så öppnar det också upp för fall där information kan gå igenom filtret men ändå drabba användaren[2, 4].

5.7 Loggning

Loggning innebär att man samlar och sparar information om hur en webbapplikation används. Eller mer formellt:

> *“Logging is the recording of events or statistics to provide information about system use and performance.”*
> Bishop, [2]

Informationen kan man sedan använda på olika vis, till exempel för att spåra felaktig eller olaglig användning, belastning eller säkerhetsbrister. Om applikationen ska ingå i ett större system så kan loggarna användas i ett IDS(Intrusion Detection System)[2].

Det finns en stor mängd information man kan logga. I säkerhetssammanhang kan det till exempel finnas ett stort värde i att logga inloggningar, IP-adresser och begärda resurser. Det är viktigt att också logga misslyckade inloggningsförsök och när begäran av resurser misslyckas[16].

Misslyckade inloggningar kan användas för att spåra försök att olovligen komma åt användarkonton och därmed resurser. En person som försöker komma åt en inloggning kan använda sig av en teknik som kallas för Brute Force, en attack när man helt enkelt testar alla möjliga sorters inloggningar. Denna sorts attacker syns mycket tydligt i loggar eftersom ett stort antal misslyckade inloggningar sker, till skillnad mot om man av misstag råkar skriva in fel användaruppgifter, för då blir det vanligen bara fel ett fåtal gånger.

Att logga användarnas IP-adresser kan vara till godo när någon användare har begått ett brott eller har brutit mot reglerna för applikationens användande. Detta kan till exempel underlätta en polisanmälan, eller en banlysning utav användarens IP, för förhindran av framtida användande av applikationen.

Begärda resurser kan till exempel vara vilka dokument och sidor för applikationen som begärts och detta kan vara viktigt att spara om en olaglig åtkomst har skett. Det kan ge en bild över vilka resurser attackerarna har kommit åt eller försökt komma åt.

När man loggar bör man vara försiktig med vilken information som skrivs ner. Till exempel kan uppgifter om användare, eller viktiga resurser hamna i loggen. Om loggen ska användas i sammahang där informationen i den kan avslöjas för obehöriga personer så bör den saneras från känsliga data. Vad som bör saneras beror på den policy man arbetar efter[2]. Ett alternativ till sanering är att man krypterar loggarna så att bara autentiserade personer kan läsa den[26].

5.7.1 Sammanfattning

Loggning sker ofta på systemnivå, eller på underliggande servernivå, men uppgifter som är specifika för applikationen loggas inte alltid och därför kan det finna en säkerhetsvinst i att införa loggning av applikationsspecifika händelser. Säkerhetsvinsten i detta är att användandet kan övervakas och att händelser i efterhand kan undersökas.

6 Punkter som inte testas

Inom säkerhet finns många områden, och för att få en fullständig säkerhet inom ett system så behöver alla dessa punkter beaktas och säkras, men eftersom vi bara inriktar oss på mjukvaror, specifikt webbapplikationer, så kan vi inte ta alla dessa punkter i beaktning. Det kan ändå vara bra att ge en kort samling av punkter som vi inte beaktar, för att förtydliga gränserna för vårt arbete.

6.1 Fysisk säkerhet

Fysisk säkerhet används för att beskriva skydd som behövs utanför datasystemet [16]. Till exempel inkluderar detta vakter, lås och stängsel för att motverka direkta attacker. Dessutom finns det andra skydd mot indirekta hot, till exempel naturkatastrofer (eld, översvämning, jordbävning m.m). Naturkatastrofer är svåra att förhindra, men det går att skydda sig mot vissa med till exempel brandsläckningssystem, tätning av lokaler och förstärkta väggar/tak. När det gäller fysisk säkerhet så beror nästan allt på hur välkonstruerade rummen/byggnaderna som datorerna står i är. Bra rum/byggnader skyddar mot de flesta fysiska hoten utifrån.

Eftersom hela målet med detta arbete är att undersöka enbart enskilda mjukvaror/applikationer så är det inte relevant att ta med fysisk säkerhet som en punkt i testerna.

6.2 Policy

En säkerhetspolicy ska svara på tre frågor [16].

- Vem ska ha access?
- Vilka resurser får användaren komma åt?
- Vad får användaren göra på resurserna?

Ett exempel på en policy kan vara att endast de som arbetar på löneavdelningen får komma åt alla anställdas löneuppgifter och ändra dem. Sedan kan man ha med som policy att endast chefen för organisationen får lägga till och ta bort anställa, eller att alla ändringar i registret över anställda måste godkännas av chefen.

Ett annat exempel är att ingen får ta med sig några datorer och/eller flyttbara media varken in i eller ut ur lokalerna. Detta för att förhindra läckage av hemlig information och införande av virus.

Det är inte relevant att fördjupa sig för mycket i denna punkt eftersom testerna ska göras på enskilda mjukvaror/applikationer och inte på organisationen i sig.

7 Test

7.1 Testapplikationer

7.1.1 Nessus

Nessus är ett verktyg som letar efter svagheter och hål i system. Applikationen är inte inriktad enbart mot webbapplikationer men kan fortfarande ge en bra bild om det finns svagheter i en sådan. För privatpersoner så är Nessus gratis och att kommersiellt använda Nessus kostar $1200 per licens och år. Nessus är en av de mest kända testapplikationerna.

Klienten till Nessus finns till både UNIX-baserad och Windows-baserade OS. Nya uppdateringar till Nessus släpps dagligen, och om man är en gratis-användare får man tillgång till dem efter en vecka, medan licensierade användare får tillgång direkt.

Nessus arbetar genom att genomföra en portscanning mot den eller de datorer som skall undersökas och sedan utför den attacker mot de portar den hittar. Dessa attacker kan, beroende på hur Nessus är inställt, krasha den dator som testas, och därför bör man inte använda Nessus mot maskiner som kör mjukvara som verkligen används för tillfället, om man inte är säker på hur det kommer påverka maskinen.

Nessus är den testmjukvara som är mest populär i dagsläget och gillas till exempel av stora säkerhetsorganisationer såsom SANS Institute. I början på april 2009 kom version 4.0 av Nessus ut[32].

7.1.2 SARA

SARA står för Security Auditor's Research Assistant och är en vidareutveckling av SATAN, Security Administrator's Tool for Analyzing Networks som var populärt i mitten av 90-talet. SARA söker efter svagheter och försöker utföra både SQL-injektioner och XSS-attacker. Tyvärr verkar SARA vara dött sedan början på 2008 och utan uppdateringar för nya svagheter så blir SARA ganska oanvändbart.

7.1.3 Webscarab

Webscarab är en applikation som inriktar sig på att analysera och manipulera HTTP och HTTPS-protokollen. Därför är det ett utmärkt verktyg för att testa en webbapplikations säkerhet. Med hjälp av den kan man fånga meddelanden som skickas mellan browser och server och inspektera innehållet. Detta kan vara bra om man vill förvissa sig om att till exempel lösenordet skickas krypterat och inte i klartext. Med hjälp av Webscarab kan man också modifiera de meddelanden man fångar och på så vis kan man utföra attacker av olika slag såsom XSS eller SQL-injektioner. Webscarab utvecklas av OWASP, en organisation

som arbetar för att skapa fritt tillgängliga verktyg, teknologier, metoder och artiklar inom säkerhetsområdet[14].

7.1.4 Wireshark

Wireshark, före detta Ethereal, är en protokollanalysator som kan sniffa paket från ett nätverksinterface. Wireshark klarar av att tolka hundratals olika protokoll och har en mängd funktioner såsom filtrering av paket och dekryptering av protokoll som IPsec, SSL/TLS med flera. Wireshark klarar av att fånga alla paket som passerar på ett interface, även paket som inte är menade för den host som kör Wireshark. Detta gör Wireshark till ett väldigt användbart program när man vill analysera nätverkstrafiken, oavsett protokoll. Wireshark är gratis och kan köras på både Windows, Linux och Mac OS.

7.1.5 Granskningsverktyg för loggar

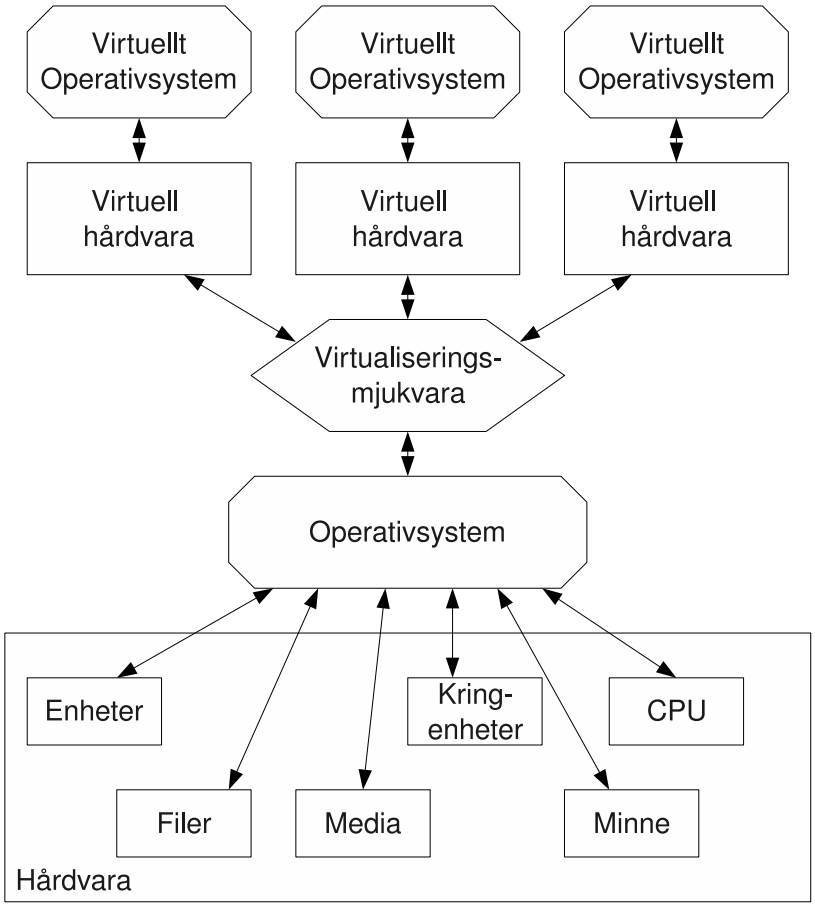
7.2 Testmiljö

Om man vill utföra tester på en webbapplikation så kan det ställas vissa krav på olika tillgängliga miljöer. Detta gäller både server- och klientmiljön. Olika webb-applikationer kan ha olika krav på vilket operativsystem som behövs och klientmiljön kan kräva olika verktyg som bara finns till vissa operativsystem. När man skall sätta upp en testmiljö så kan man i huvudsak göra på två olika sätt. Antingen sätter man upp en fysisk miljö, eller så sätter man upp en virtuell miljö.

7.2.1 Virtuell miljö

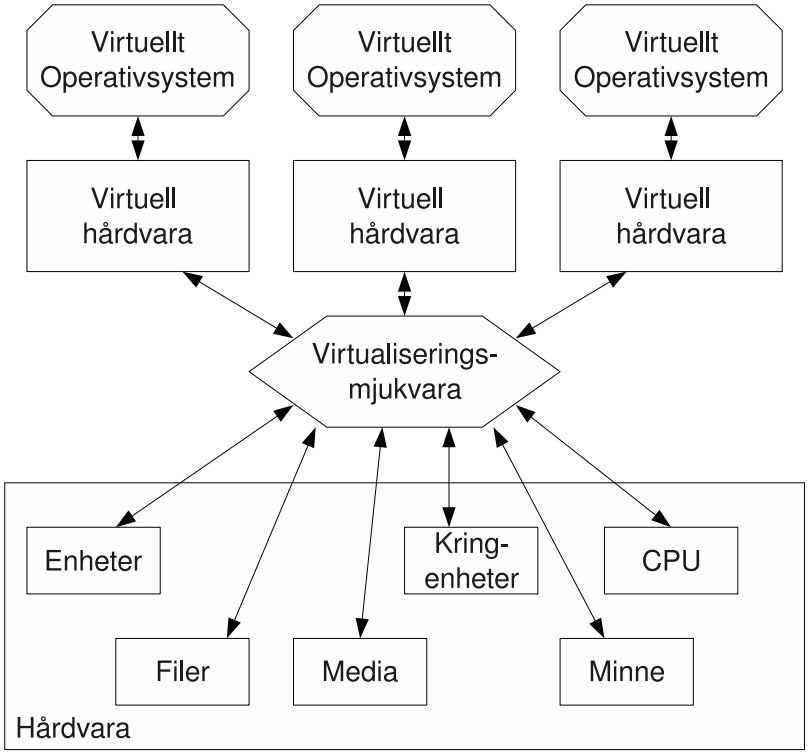
En virtuell miljö bygger på att man virtualiserar hårdvaran i ett datorsystem. Detta gör man genom att köra en speciell mjukvara som agerar som värd för ett eller flera gäst-OS. Mjukvaran kan antingen köras ovanpå ett annat operativsystem som då kallas för värd-OS, eller direkt på hårdvaran. Mjukvaran simulerar hårdvara som sedan gäst-operativsystemen

arbetar emot. I figurerna 7.1 och 7.2 kan du se en översikt över hur ett virtualiserat system kan se ut.



Figur 7.1: Virtualisering ovanpå värd-OS

I stor mån kan man styra åtkomsten till den riktiga hårdvaran, till exempel hur nätverksaccess ska ske och hur mycket diskutrymme och minnesutrymme som ska få användas för varje OS. Anledningen till att man gör detta är att en maskin vanligtvis bara utnyttjas runt 5-15%, men med virtualisering kan man öka utnyttjandegraden till 60-80%[24, 23] eftersom man låter en maskin ta hand om fler servrar. Detta resulterar i att man kan minska antalet fysiska maskiner, och därmed minskar även administration, energiförbrukningen och kostnaden för inköp och underhåll av hårdvara. Däremot så innebär det att man klumpar ihop



Figur 7.2: Virtualisering direkt på hårdvaran

ett visst antal servrar på en maskin och om den maskinen skulle krasha eller gå sönder så drabbas alla dessa servrar. Detta innebär att man måste öka driftsäkerheten på andra vis, till exempel genom tätare backuper eller genom policyförändringar.

Vi har under vårt arbete kommit i kontakt med ett par olika mjukvaror för att skapa en virtuell maskin. Dessa är VMware och VirtualBox som vi även har använt tidigare. Utöver dessa två finns en uppsjö av andra alternativ. Vi har valt ut några av de största och kanske mest intressanta alternativ. Dessa fyra följer nu en beskrivning av. Andra alternativ kan man hitta information om på Wikipedia[28].

7.2.1.1 VirtualBox

Virtualbox simulerar en x86-arkitektur och kan simulera en eller flera virtuella maskiner. I tabell 7.1 listas vilka värd- respektive gäst-OS som stöds.

Tabell 7.1: Operativsystem som stöds av VirtualBox[22]

Gäst-OS	**Värd-OS**
Windows (NT 4.0, 2000, XP, Server 2003 och Vista)	Windows
DOS/Windows 3.x	Linux
Linux (2.4 och 2.6)	Macintosh
Solaris och OpenSolaris	OpenSolaris
OpenBSD	

Virtualbox har ett speciellt filformat som den lagrar de emulerade hårddiskarna till datorn i. Detta format kallas "Virtual Disk Images". Utöver det kan man även montera iSCSI-enheter, diskar som skapats av VMware(.vmdk) och det finns även ett verktyg för att konvertera diskar som skapats av Virtual PC. VirtualBox simulerar ett VESA-kompatibelt grafikkort och till vissa gäst-OS finns drivrutiner som ger bättre prestanda och utökad funktionalitet. En mängd nätverkskort kan simuleras för att tillåta att nätverksdrivrutiner inte ska behövas installeras för de flesta operativsystem. Ljudkort stöds också, och då i form av ett ICH AC'97-kort eller ett SoundBlaster 16-kort. Virtualbox har även stöd för att montera USB-enheter på värddatorn och kan också montera ISO-filer direkt i VirtualBox. Fler funktioner anges på [39].

Virtualbox fanns inledningsvis bara som proprietär mjukvara men släpptes under 2007 också som öppen källkod. Den senare versionen saknar vissa funktioner jämfört med den första. Dessa funktioner är:

- Inbyggd RDP-server
- USB-stöd
- iSCSI-stöd
- Gigabit Ethernet
- SATA-stöd

7.2.1.2 Virtual PC

Virtual PC är en produkt av Microsoft som likt VirtualBox simulerar en x86-arkitektur. I tabell 7.2 listas Värd- och Gäst-OS som stöds av Virtual PC.

Tabell 7.2: Operativsystem som stöds av Virtual PC[31]

Gäst-OS	**Värd-OS**
Windows Server 2008	Windows Vista Ultimate (32-bit och 64-bit)
Windows Vista (32-bit)	Windows Vista Enterprise (32-bit och 64-bit)
Windows XP (32-bit)	Windows Vista Business (32-bit och 64-bit)
Windows Server 2003 Standard Edition	Windows Vista Business N (32-bit och 64-bit)
Windows 2000 Professional och Server	Windows Server 2003
Windows 98 Second Edition	Windows XP Professional (32-bit och 64-bit)
Vissa versioner av IBM OS/2	Windows XP Tablet PC Edition (32-bit)
	Windows XP Media Center Edition 2005
	Windows Vista Home Premium (64-bit)
	Windows Vista Home Basic (32-bit och 64-bit)

En väldigt specifik hårdvara simuleras och den är framtagen för att ge så stor kompatibilitet som möjligt. Den består av:

Pentium III-processor eller, om värd-OSet är Windows, en virtualisering av värd-processorn

VESA-kompatibelt grafikkort med 4-16 MB minne

440BX-chipset

Creative Sound Blaster 16 ISA som ljudkort

DEC 21140 från Digital Equipment Corporation som nätverkskort

USB stöds bara till den grad att vissa standardenheter såsom tangentbord och styrdon som används av värd-OSet kan användas. Virtual PC använder en image-fil som hårddisk. Denna fil är av formatet Virtual Hard Disk (VHD). Du kan länka en VHD-fil till en fysisk hårddisk och på så vis montera den i Virtual PC. Du kan, däremot, inte länka filen till en partition, utan enbart en hel hårddisk. Virtual PC är gratis att använda men koden är proprietär[6].

7.2.1.3 VMware

VMware är den mjukvara för virtualisering som är mest utspridd. Det började utvecklas under 1998 och används idag av många stora företag. Till en början fanns bara stöd för Windows och GNU Linux som värd-OS men 2006 lades även stöd för Mac OS X till. VMware finns i ett flertal versioner. I tabell 7.3 och 7.4 kan du läsa mer om dessa versioner.

Tabell 7.3: Desktop-versioner av WMvare

Version	**Förklaring**
VMware Workstation	En användare kan skapa och köra flera maskiner med 32- eller 64-bit operativsystem.
VMware Player	En VMware versionen med fri licens. Kan spela upp maskiner men kan inte skapa dem.
VMware Fusion	Liknande VMware Workstation men är gjord för Intelbaserade Mac OS X-datorer.

Tabell 7.4: Server-versioner av WMvare[31]

Version	**Förklaring**
VMware ESX	Körs direkt på hårdvaran utan värd-OS. Ger bättre prestanda än desktop-versionerna. Företagsinriktad.
VMware ESXi	En lättare version av VMware ESX med mindre minnes- och disk-användning. Är gratis att använda sedan 2008.
VMware Server	Tillåter skapande och uppspelande av virtuella maskiner. Gratis att använda. Körs på ett värd-OS.

Vissa av versionerna är gratis att använda och det är de eftersom VMware hoppas att användare skall börja med dem, och sedan gå över till de versioner som kostar pengar för att kunna ta del av större funktionalitet. VMware utvecklar även en mängd andra verktyg som är till för att underlätta vissa moment. Till exempel finns verktyget VMware Converter som konverterar en fysisk maskin till en virtuell. VMware ThinApp använder virtualisering för att lura program att de är installerade på en riktig maskin och på så vis kan programmen göras portabla. VMotion används för att kunna flytta körande maskiner mellan olika värdar.

7.2.1.4 Xen

Xen utvecklas av Cambridges Universitet i England och är en virtualiseringsmjukvara som körs direkt på hårdvaran, likt VMware ESX och ESXi. Vinsten med detta är ökad prestanda när resurserna som användes till värd-OSet istället kan användas till virtuella maskiner och virtualiseringsmjukvaran får en mer direkt kontakt med hårdvaran. Mjukvaran fungerar på det viset att ett visst OS installeras ovanpå den, detta OS kallas domain 0 (dom0), och har speciella privilegier när det gäller hårdvaruaccess och hantering av ytterligare OS. Endast speciella versioner av Linux, Solaris och FreeBSD kan köras som dom0-OS. Från detta OS startar en administratör de eventuella andra OS som också skall köras. Dessa OS kallas i så fall för domain U (domU) och modifierade versioner av UNIX-lika operativsystem kan köras som domU. I version 3.0 av Xen så har även stöd för olika Windows-operativsystem lagts till, förutsatt att viss hårdvara används och att processorn stödjer x86-virtualisering, en teknik som gör det enklare att köra virtualiserade OS mot processorn.

7.2.2 Fysisk miljö

Motsatsen till en virtuell miljö är att använda en fysisk miljö. I en sådan så kör varje maskin bara ett OS. Dessutom är nätverket fysiskt med kablar och annan nätverksutrustning. Detta gör att man slipper faran med virtualisering, nämligen att fler servrar klumpas ihop, och därmed riskerar att krascha samtidigt om något händer den fysiska maskinen. Du slipper även den prestandaförlust som sker i virtualisering på grund av overhead som skapas av virtualiseringsmjukvaran. Men det ger också ökade kostnader, för inköp och drift. Detta kan vara ett problem om olika system som skall testas ställer olika krav på miljön. En viss webbapplikation testas bara vid ett tillfälle och därför är det inte kostnadseffektivt att köpa in ny samling hårdvara om ett visst test kräver det.

8 Slutsats

8.1 Resultat

När vi inledde arbetet så hade vi från vår uppdragsgivare fått uppdraget att ta fram ett test för säkerheten(eng. security) i olika applikationer. Detta test skulle vara lättviktigt, det vill säga, det skulle ha en rimlig kostnad, men ändå tillräckligt för att ge en bra bild av applikationen säkerhet. Det fanns ett antal problem med att uppnå detta mål. För det första så finns det väldigt lite litteratur om säkerhet på applikations nivå. Traditionellt har säkerhet alltid har varit ett begrepp man har använt på systemnivå, det vill säga att applikationens omgivning har varit en stor del att beakta när man talat om säkerhet. Omgivningen innefattar till exempel operativsystem, yttre skyddmurar såsom brandväggar och accesskontroller med mera. Det materiall vi hittade som pratade om applikationstester handlade bara om säkerhet i applikationskoden. En stor anledning till att detta område är så lite omskrivet kan vara att applikationer skiljer sig så väldigt mycket sinsemellan, och att tester av applikationer då är svåra att göra i rimlig storlek och omfattning. För att kunna fortsätta med vårt arbete har vi därför fått göra en begränsning. Vi har inte försökt ta fram ett säkerhetstest för applikationer av olika slag, utan har fokuserat på en viss kategori av applikationer, nämligen webbapplikationer. En webbaplikation är en applikation som erbjuder en tjänst över ett nätverk. Vi har därefter letat efter information om vilka säkerhetsbrister en webbapplikation kan tänkas ha.

Resultatet vårt arbete visar att det finns ett antal olika områden där säkerheten av olika webbapplikationer kan behövas testas. En stor del av dessa uppstår i de webbgränssnitt som användaren använder och de uppstår på grund av bristande kontroller av riktigheten i de data man accepterar från användaren. Andra säkerhetsproblem kan uppstå beroende på hur man väljer att hantera och spara de data man samlar in via applikationen.

8.2 Förslag på testfrågor

Vårt arbete har visat på ett antal åtgärder för olika säkerhetsbrister som bör göras om en webbapplikations skall anses vara säker. Eftersom vår uppdragsgivare vill ha ett test som kan användas för att utvärdera en applikation har vi tagit fram ett antal frågor. Nedanför följer en lista av frågor samt en kort motivation till varför frågan är intressant.

8.2.1 Kryptering

- Används hashning av lösenordsuppgifter i databasen?

Hashning är en envägskryptering som förhindrar att man kan läsa ut ett lösenord ur databasen om till exempel en attackerare lyckas få tag på den.

- Används salt?

Även om hash ökar säkerheten för lagrade lösenord så kan man komma runt den genom att i förväg generera hashar(Rainbow Tables) av olika lösenord och sedan jämföra dem med de hashar som finns i databasen. Detta förhindrar man genom att använda salt, ett värde som läggs ihop med lösenordet innan det hashas.

- Länkas salt och lösenord samman på ett unikt vis?

Salt och lösenord skall sättas ihop innan hashning men detta kan göras på många sätt. Att bara lägga dem efter varandra i en sträng är det vanligaste sättet men om man sätter ihop dem på ett udda vis så blir det ännu svårare för en angripare att ta fram ett samband mellan hash och lösenord.

8.2.2 Cross Site Scripting och SQL-injections

- Filtreras användarinmatad information som skall visas på webbsidan?

När en användare matar in information så måste den filtreras för att inte otillåten syntaktisk information skall visas på sidan. Sådan information kan användas för att förändra funktion och utseende på sidan. Till exempel kan otillåtna javascript laddas in från en annan sida, och detta script kan utföra vissa attacker som att till exempel stjäla användarens

information.

- Vilken av dessa tre tekniker används för att filtrera informationen?

- **Endast kända bra data accepteras**
- **Kända dåliga data avvisas**
- **Kända dåliga data saneras**

Den första tekniken är bäst att använda eftersom den inte ger någon attackyta. De två andra är inte lika bra eftersom alla dåliga data inte behöver vara kända. Detta visas till exempel i fall där teckenkodning har tillåtit attacker att kringgå filter.

- Filtreras användarinmatad information som skall lagras i databasen?

Precis som för Cross Site Scripting måste information som skall lagras i databasen rensas från syntaktisk information. Detta eftersom det annars finns en risk för att användare kan utföra otillåtna operationer på databasen. Detta kan ske eftersom en databas inte kan skilja på information som kommer direkt från webbapplikationen och information som kommer från användaren.

- Används fördefinierade funktioner för att rensa användarinmatad information?

När en person ska implementera ett filter för information så händer det lätt att man missar något. Därför är det bättre att använda fördefinierade filterfunktioner om det finns tillgängligt. Dessa funktioner har antagligen blivit grundligt granskade och testade av ett stort antal personer.

8.2.3 Loggning

- Loggas misslyckade inloggningar?

En logg över misslyckade inloggningar kan vara värdefull för att kunna se om någon försöker ta sig in på ett konto. Detta kan till exempel synas i loggen som en stor mängd felaktiga inloggningar.

- Loggas IP-adresser?

Att logga IP-adresser kan vara väsentligt för att spåra lyckade och misslyckade attacker, eller för att spåra någon somn har utfört olagliga aktiviteter med hjälp av, eller på din sida.

- Loggas begärda resurser?

Att logga begärda resurser kan hjälpa till vid felsökning eller spårning av attacker.

- Saneras loggar från känsliga uppgifter?

Det mesta kan loggas i en applikation, men detta kan innebära att känsliga uppgifter hamnar i loggen. Det kan vara bra att sanitera loggarna för att skydda den personliga intergiteten.

8.2.4 Personlig Integritet och Kommunikation

- Anonymiseras data som inte behöver vara personligt bundet?

En webbapplikation kan samla ihop mycket information om en användare. För att förhindra att informationen skall användas felaktigt kan det vara bra att anonymisera informationen. Detta kan till exempel göras med pseudonymer.

- Krypteras trafik med SSL(HTTPS) eller IPSec eller annan motsvarande teknik?

Den här frågan är aktuell eftersom information som skickas över Internet inte nödvändigtvis är skyddad mot avlyssning. Det är speciellt viktigt att använda när känsliga uppgifter såsom inloggning eller personuppgifter skickas.

8.2.5 Autentisering

- Klarar applikationen av att krav ställs på lösenordet?

Det är upp till policyn att definiera vilka krav som skall ställas på lösenordet, men det spelar ingen roll om inte applikationen klarar av att ställa dessa krav.

- Används en kombination av autentiseringstekniker, till exempel lösenord och Smart Card

En kombination av flera olika autentiseringstekniker kan ge en säkrare lösning. Detta är vanligt att banker använder. Till exempel så använder Swedbank både en PIN-kod och en dosa för att man ska kunna logga in.

8.3 Vad vi skulle ha gjort annorlunda

Till att börja med, så borde vi planerat mer redan från start. Det har blivit för mycket att vi haft några punkter som vi har gjort, utan att försöka hitta fler punkter. Vi skulle ha försökt få kontakt med någon som har praktisk erfarenhet av säkerhetstester för att få idéer om vilka säkerhetsaspekter som är viktiga att tänka på. Det hade varit en bra grund att stå på, istället för att själva försöka komma på allting från grunden. Vi skulle också lagt ner mer tid på att komma fram till olika metoder man kan använda för att testa de punkter som vi har kommit fram till. Vi skulle dessutom kunnat ta kontakt med någon inom målgruppen för testerna för att undersöka vilket perspektiv som är intressant för dem.

Referenser

[1] Alessandro Acquisti. Privacy and security of personal information. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.6360&rep=rep1&type=pdf, 2004.

[2] Matt Bishop. *Computer Security: Art and Science*. Addison Wesley Professional, 2003.

[3] A. Bosselaers, H. Dobbertin, and B. Preneel. The ripemd-160 cryptographic hash function. *Dr. Dobb's Journal*, 22(1):24, 26, 28, 78, 80 –, 1997/01/. RIPEMD-160 cryptographic hash function;data integrity;MD4;MD5;secure replacement;.

[4] Steven Cook. A web developers guide to cross-site scripting. As part of GIAC practical repository, January 2003.

[5] Datainspektionen. http://www.datainspektionen.se/lagar-och-regler/personuppgiftslagen/, Februari 2009.

[6] Dan Goodell. Virtual pc tutorial. http://www.goodells.net/virtualpc/vhd.htm, April 2009.

[7] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso. A Classification of SQL-Injection Attacks and Countermeasures. In *Proc. of the International Symposium on Secure Software Engineering*, Mars 2006.

[8] Justitiedepartementet. http://www.riksdagen.se/webbnav/index.aspx?nid=3911&bet=1998:204, Februari 2009.

[9] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (4th Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2007.

[10] David Morgan. Web application security - sql injection attacks. *Network Security*, 2006(4):4 – 5, 2006. Arbitrary SQL commands;Web applications;.

[11] Nationalencyklopedin. http://www.ne.se, Februari 2009.

[12] NIST (National Institute of Standards and Technology). Cryptographic hash project. http://csrc.nist.gov/groups/ST/hash/index.html, Maj 2009.

[13] Rolf Oppliger. *Security technologies for the World Wide Web*. Artech House, Inc., Norwood, MA, USA, 2000.

[14] OWASP. Owasp webscarab project. http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project, April 2009.

[15] Steve Pettit. Anatomy of a web application. White paper, July 2001.

[16] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing.* Prentice Hall Professional Technical Reference, 2002.

[17] Jayamsakthi Shanmugam and M. Ponnavaikko. A solution to block cross site scripting vulnerabilities based on service oriented architecture. pages 861 – 866, Piscataway, NJ 08855-1331, United States, 2007. WEB applications;Languages (traditional);International (CO);Cross site scripting;international conferences;Cross-site scripting (XSS);Research data;Applied (CO);Application servers;Other interfaces;web pages;Service oriented architecture (SOA);.

[18] Adam Slagell, Jun Wang, and William Yurcik. Network log anonymization: Application of crypto-pan to cisco netflows. 2004.

[19] William Stallings. *Cryptography and Network Security (4th Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005.

[20] Zhendong Su and Gary Wassermann. The essence of command injection attacks in web applications. *ACM SIGPLAN Notices*, 41(1):372 – 382, 2006. Command injection attacks;Web applications;Parsing;Runtime verification;.

[21] Smart card based authentication - any future? *Computers & Security*, 24(3):188 – 191, 2005.

[22] VirtualBox. `http://www.virtualbox.org/`, April 2009.

[23] virtualisering.nu. Konsolidering eller virtualisering? `http://www.virtualisering.nu/templates/virtualisering.php?categoryID=126&id=413`, April 2009.

[24] VMware. Wmvare cost savings. `http://www.vmware.com/solutions/cost-savings/`, April 2009.

[25] K.-P.L. Vu, R.W. Proctor, A. Bhargav-Spantzel, B.-L. Tai, Joshua Cook, and E.E. Schultz. Improving password security and memorability to protect personal and organizational information. *International Journal of Human-Computer Studies*, 65(8):744 – 57, 2007/08/. password security;password memorability;authentication;information access;information security;.

[26] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and D. K. Smetters. Building an encrypted and searchable audit log. 2004.

[27] Wikipedia. Cia triad. `http://en.wikipedia.org/wiki/CIA_Triad`, Februari 2009.

[28] Wikipedia. Comparison of platform virtual machines. http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines, April 2009.

[29] Wikipedia. Fingerprint recognition, april 2009.

[30] Wikipedia. Iris recognition, april 2009.

[31] Wikipedia. Microsoft virtual pc. http://en.wikipedia.org/wiki/Microsoft_Virtual_PC, April 2009.

[32] Wikipedia. Nessus. http://en.wikipedia.org/wiki/Nessus_(software), April 2009.

[33] Wikipedia. Rainbow table. http://en.wikipedia.org/wiki/Rainbow_table, Februari 2009.

[34] Wikipedia. Retinal scan, april 2009.

[35] Wikipedia. Sha hash functions. http://en.wikipedia.org/wiki/SHA_hash_functions, Maj 2009.

[36] Wikipedia. Speaker recognition, april 2009.

[37] Wikipedia. Transmission control protocol. http://en.wikipedia.org/wiki/Transmission_Control_Protocol, Februari 2009.

[38] Wikipedia. User datagram protocol. http://en.wikipedia.org/wiki/User_datagram_protocol, Februari 2009.

[39] Wikipedia. Virtualbox. http://en.wikipedia.org/wiki/VirtualBox, April 2009.