Riskanalys inom intrångsäkerhet på webbplatser

Författare:
Alexander Gustafsson

Datum:
2004-05-27

Handledare: Bo-Krister Vesterlund
Examinator: Guohua Bai

Vt 2004

Blekinge Tekniska Högskola

Sammanfattning

Attacker och intrång på webbservrar är idag vanligt förekommande. Webben gör det lätt för hackare, knäckare och andra inkräktare att hitta sårbara servrar, och det finns gott om tips att hämta för den som vill lära sig hur man gör intrång. Det finns ett flertal olika intrångsmetoder som utnyttjar olika typer av svagheter i datorsystemen. Denna uppsats inriktar sig på svagheter i webbplatsernas serverskriptsystem, dess skriptkod och konfiguration. Syftet är att undersöka huruvida intrång kan göras med endast en webbläsare via webbplatsens offentliga webbsidor.
Genom att kombinera tre olika metoder - *litteraturundersökning*, en *enkät* och ett *experiment* - undersöker uppsatsen hur serverskriptintrång fungerar. Den analyserar ett urval vanliga misstag webbprogrammerare kan göra, till exempel att inte kontrollera inkommande data, eller att använda lättgissade variabelnamn och databastabellnamn. Några olika typer av intrång analyseras, som till exempel SQL-injektion. Förebyggande åtgärder tas även upp med ett antal konkreta exempel.
Uppsatsens slutsats är att på webbplatser med svaga serverskriptsystem kan inkräktare göra intrång via webbplatsens egna publika webbsidor, med endast en vanlig webbläsare som hjälpmedel. I uppsatsens avslutande del diskuteras även några orsaker till varför det produceras ogenomtänkt skriptkod, till exempel beroende på att programmeringskurser i allmänhet inte tycks lära ut säker programmering i tillräcklig utsträckning.

Nyckelord:
säkerhet, intrång, webbplats, skript, server, hackare, programmering

Abstract

Attacks and intrusions are common today on web servers. The web makes it easy for hackers, crackers and other intruders to find vulnerable servers, and there are plenty of instructions to download on how to become an intruder. There are many different methods for intrusion that exploits different kinds of weaknesses in computer systems. This report is focused on weaknesses in web sites **server script systems,** its server script code and configuration. The purpose is to investigate if intrusions are possible by using a web sites public web pages and just an ordinary web browser as a tool.

By combining three different examination methods - *litterature examination, a survey* and *an experiment* - this report investigates how intrusions in server script systems are working. It analyzes some common mistakes web programmers can do, for example not checking and validating incoming data, or using variabel names and database table names that are easy to guess. Some different types of intrusion methods are analyzed, for example SQL injections. Preventive measures are presented as well, with some concrete examples.

The conclusion of the report is that on web sites with weak server script systems intrusions can be possible by using the web sites public web pages and a web browser. In the last part of the report some reasons are discussed why programmers produce unsecure code, for example because programming courses generally do not seem to teach enough secure programming.

Keywords:
security, intrusion, web site, script, server, hacker, programming

Innehållsförteckning

1 Inledning

1.1 Bakgrund

De senaste årens globala epidemier av datorvirus, maskar med mera som sprids via Internet indikerar att säkerhetstänkandet generellt sett inte är tillräckligt högt bland programmerare, tillverkare och även användare. Dessa typer av intrång eller attacker som masspridning av elakartade program utgör, får stor uppmärksamhet i massmedia på grund av att ett stort antal användare drabbas.

Andra typer av intrång eller attacker kommer inte till allmänhetens kännedom i lika stor utsträckning, som till exempel vid intrång riktade mot specifika servrar. Troligen finns där ett stort mörkertal eftersom företag som drabbas av intrång kan vara rädda att förlora kundernas förtroende om det skulle framkomma att man brister i säkerheten (*Sandberg & Höij*, 2003). Riktigt stora intrång kan dock vara svåra att dölja för allmänheten, som till exempel då en inkräktare i USA kom över 8 miljoner kontokortsuppgifter tillhörande Visa, Mastercard och andra kontokortsföretag (*Hilley, Sarah.,* 2003).

De fall som kommer till omedelbar allmän kännedom kan till exempel vara där inkräktaren modifierat innehållet på en webbplats (s.k. elektronisk vandalism eller e-graffiti) så att det tydligt framgår för omvärlden att intrång har skett (*Chirillo,* 2001, s 83ff). Som till exempel när regeringens webbplats vandaliserades och som i kvällstidningarna basunerades ut som "Porrkupp i natt mot regeringen", eller då tv4:s webbplats drabbades av nyheten "Bush bakom attack mot Sverige":

Figur 1-A Sabotage hos tv4:s webbplats. Källa: *Bjerre,* 2004.

Figur 1-B Sabotage hos regeringens webbplats. Källa: *Kärrman,* 2001.

Ovanstående exempel utgör mer spektakulära resultat av elektronisk vandalism som kan skada företags anseende och trovärdighet, särskilt om det upprepas flera gånger.
Oavsett typen av intrång så har aktiviteter från inkräktare på webbplatser ökat det senaste året. Enligt en studie rapporterade vartannat undersökt företag att de haft en allvarlig attack den senaste tiden (*Danielsson,* 2004). En inkräktare behöver inte nödvändigtvis komma utifrån

företaget/webbplatsen, det kan lika gärna vara en missnöjd anställd, f.d. anställd eller annan som vistas inom företaget, som utför intrånget. Undersökningar visar att så mycket som 50 till 70 % av alla attacker (inklusive ofullbordade intrång) startar inifrån företagen (*Crume*, 2000, s 86) och (*Brenton,* 2001, s 6).

Internet är ett välfyllt smörgåsbord för inkräktare som vill göra intrång i datasystem. På Internet finns tillgång till både färdiga instruktioner och gratis mjukvaruverktyg som underlättar för inkräktaren, vilket medför att avancerade intrång även kan utföras av fullständiga nybörjare (så kallade skript kiddies) utan tekniska kunskaper.
Många intrång baseras på och utnyttjar brister och säkerhetshål i den mjukvara webbplatser använder och som tillverkas av de stora mjukvarujättarna. Visserligen har det rapporterade antalet säkerhetshål planat ut senaste året (dock ej minskat), men det kan bero på att antalet nytillverkade program var litet under denna tid (*Danielsson,* 2004).
Inte heller internets grundstruktur är anpassad för dagens säkerhetsbehov. Det system som idag kallas Internet och vars utveckling startade kring 1970 användes ursprungligen av mestadels kunniga och ärliga användare inom universitets- och forskningsvärlden. Varken pc:n eller webben (www) var då uppfunna ännu, och den stora allmänheten kände inte till internets existens. Idag består användarna av en stor del okunniga (en orsak till virusepidemiernas stora omfattning), en liten del kunniga och ärliga, en mindre del kunniga och oärliga, och ytterligare en del oärliga och okunniga (*Andersson*, 2001, s 369). Internets användning och omfattning är annorlunda idag än för trettio år sedan, men dess grundläggande teknik är i stort detsamma.

I massmedia kallas ofta inkräktarna sammanfattande för hackare (hackers), men initierade personer vill gärna uttrycka sig mer nyanserat. De kan då använda benämningar som hackare, knäckare (crackers), blackhats, white hats, grey hats, cyberpunks, skript kiddies, attackers, och så vidare. Man gör då skillnad utifrån intrångets syfte, omfattning, skicklighet och så vidare. (*Honeynet,* 2002, s 123), (*Chirillo,* 2001, s 83ff).

Juridiskt sett är dock ett intrång ett intrång, oavsett vad inkräktaren vill kalla sig själv. Intrång innebär enligt brottsbalken (kapitel 4 paragraf 9 c) att olovligen bereda sig tillgång till data eller information, eller "ta del av upptagning", som lagras på en dator eller dylikt. Detta kan ske på avstånd via till exempel Internet, eller genom att fysiskt bryta sig in i till exempel ett serverrum. Det räknas som intrång även om inkräktaren inte förstör eller stjäl något, men det kan då vara svårt att bevisa att inkräktaren verkligen läst, kopierat eller tagit del av en upptagning (*Maiwald & Sieglein,* 2002, s 25).

Det finns en mängd olika metoder för att utföra attacker och begå intrång, och metoderna förändras också hela tiden beroende på vilka aktuella brister och säkerhetshål som finns i de system som de olika webbplatserna har installerat för tillfället på sina servrar. De vanligaste metoderna baseras enligt *Andersson* (2001, s 368) och *Welling & Thomson* (2003, s 270) på:

- brister och säkerhetshål i de mjukvaror webbplatsernas använder (operativsystem med mera),
- svagheter och brister i serverskripten (det vill säga i den källkod som används för att hantera data, skapa webbsidor och så vidare),
- gissning av lösenord (till exempel med hjälp av programvara),
- felkonfigureringar hos webbplatserna.

Företag/webbplatser som vill hålla hög säkerhet mot intrång måste vidta ett antal relativt komplicerade åtgärder. Förutom att placera datorutrustningen i lokaler som är skyddade mot inbrott, brand och så vidare, så bör de ha en brandvägg eller proxyserver, ett effektivt backupsystem, ett antivirussystem som uppdateras regelbundet, eventuellt någon form av buffertnät eller DMZ (demilitariserad zon) mellan sitt lokala nät och Internet, samt se till att systemen alltid är rätt konfigurerade. Dessutom bör de upprätta ett rättighetssystem i sitt nätverk, vilka personer som ska ha tillång till vad i systemet, samt en säkerhetspolicy till exempel gällande hantering av lösenord och vad i systemet som ska vara tillgängligt för besökare och andra utomstående.

Att hålla en hög säkerhet kan bli kostsamt, och ovanstående åtgärder kan falla platt till marken om man inte också har ett högt säkerhetstänkande vid kodningen av de skript som utgör webbsidorna och webbplatsens gränssnitt gentemot besökarna.
Utnyttjande av svagheter och brister i serverskripten och dess system tillhör en av de mer vanliga intrångsmetoderna, och det är detta område denna uppsats inriktar sig på.

1.2 Preciserat problem, syfte och mål

När webbprogrammerare m.fl. bygger en webbplats och dess webbsidor genom att använda skriptsystem som till exempel PHP, ASP, JSP eller Java Servlets, så bestämmer man bland annat vilken funktionalitet webbplatsen får gentemot besökarna, och vilka delar av webbplatsens innehåll som ska vara tillgänglig för webbplatsens besökare. Med "webbplatsens innehåll" menas här till exempel information från olika dokument- och html-filer som ligger i webbplatsens katalogstrukturer, och olika typer av data (till exempel namn, adresser, lösenord mm) i webbplatsens databaser.
Risken finns då att programmeraren i sin skriptkod förbiser en del mer eller mindre ovanliga situationer där besökarnas agerande och handhavande av webbsidorna inte följer ett förväntat beteende. Det vill säga då besökare via sina webbläsare begår misstag på grund av missuppfattning, ovana eller okunnighet, eller då man handlar i direkt syfte att göra intrång och utföra sabotage.

Problemområdet för denna uppsats inriktar sig därför på ett urval av de förbiseenden, misstag och fel en skript-programmerare kan begå, till exempel genom att (omedvetet) bygga in svagheter i webbsidornas programkod, och därigenom öka risken för intrång och sabotage som utförs med hjälp av vanliga webbläsare.

Syftet och målet med uppsatsen är att analysera ett urval intrångsmöjligheter orsakade av ogenomtänkt design och användning av serverskript och serverskriptsystem, så att programmerare, webbplatsägare och andra berörda som läser denna uppsats kan öka sitt säkerhetstänkande och förbättra sin egen (webbplats-)säkerhet.

1.3 Frågeställningar, hypotes och förväntat resultat

Några frågeställningar som ligger till grund för uppsatsen och för dess hypotes är:
- Vilka typer av intrång via serverskriptsystem är vanligt förekommande?
- Hur fungerar och utförs dessa intrång?
- Kan intrång utföras med hjälp av endast en webbläsare som hjälpmedel?
- Hur ser de skador ut som intrången kan resultera i?

- Vilka förebyggande åtgärder kan vidtas?

Anmärkning: frågeställningarna behandlas inom ramen för de avgränsningar som beskrivs längre fram i denna uppsats.

Problemområdet och frågeställningarna utmynnar i följande hypotes:
Om en webbplats innehåller svagheter i serverskriptkoden eller skriptsystemet, så kan det medföra att intrång och sabotage kan utföras med hjälp av en webbläsare.

Eller i annan form:
En webbplats med svagheter i serverskriptkoden eller i skriptsystemet, kan utsättas för intrång och sabotage med hjälp av en webbläsare.

Med "svagheter" menas här att skriptkod eller skriptsystem innehåller säkerhetsbrister på grund av att det inte är tillräckligt säkert kodat, byggt, konfigurerat eller dylikt.

Hypotesen beskriver även en del av uppsatsens *förväntade resultat*, nämligen att kunna visa att man med hjälp av endast en webbläsare kan begå intrång på webbplatser som har svagheter i skriptkod eller skriptsystem.

Hypotesen prövas med underlag från tre olika metoder som kompletterar varandra: litteraturundersökning, enkät och experiment.

1.4 Avgränsningar

Problemområdet avgränsas till att omfatta svagheter i serverskriptkod och serverskriptsystem, inklusive vissa områden gällande konfigurering av serverskriptsystem. Uppsatsen kan av naturliga skäl (begränsad projekttid) inte göra anspråk på att vara heltäckande inom dessa områden. Den tar upp några av de problem som kan uppstå inom skripthantering, i syfte att utgöra tillräckligt underlag för hypotesprövningen.
I litteraturen och enkäten behandlas PHP och ASP skriptsystem, eftersom de är vanligast förekommande i litteraturen. I experimentet avgränsas skriptspråk och skriptsystem till PHP (och databassystemet till MySQL).
Även andra typer av språk ingår som brukar användas i samband med skriptkod, till exempel databasspråk i databasanrop som utförs inuti skriptkoden.

2 Teoretisk bakgrund

Användning av skript på webbservrar började användas omkring 1995 i form av s.k. CGI-skript. CGI (Common Gateway Interface)[1] är en specifikation för ett standardgränssnitt mellan webbserver och exekverbara skript/program på serversidan.
Syftet är att ge möjlighet för webbservern att till exempel hantera/manipulera data och returnera dokument med data till klienten (webbläsaren). Till exempel. för att kunna utföra sökningar på webbplatsen och returnera resultatet till klienten, hantera och bearbeta

[1] CGI utvecklades ursprungligen för unix-system under 1980-talet.

formulärdata som skickas in av klienten, kontrollera kontonummer och hantera affärstransaktioner hos en webbshop, hantera databaser med mera. Det vill säga att erhålla så kallade *dynamiska* webbsidor (istället för *statiska*), där innehållet i en sida kan ändras från en sidvisning till en annan beroende på de val klienten gör på sidan.

CGI-skript kan bestå av i princip vilka exekverbara program som helst som följer CGI-specifikationen, det vill säga även kompilerade program skrivna i C, C++, Basic och så vidare (*Felton,* 1997, s7). Ursprungligen skrevs CGI-programmen med så kallade *UNIX shell skripts* och *Perl*, och därför kallas CGI-program ofta för CGI-*skript*. Det interpreterande[2] språket Perl har fått stor spridning som CGI-språk på grund av att det är lättanvänt, kraftfullt och att det finns en mängd (gratis) tillbehör och moduler skrivna i öppen källkod (*Ladd & O'Donell*, 1998, s356).

Under senare hälften av 1990-talet och senare utvecklades alternativ till CGI och CGI-skript, till exempel PHP, ASP, JSP, JavaServlets och ColdFusion. Dessa system är ofta mer lättanvända än CGI och kan vara mer integrerade i webbservern än CGI.

- **PHP**: liknar skriptspråket Perl men dess syntax bygger även på C och Java. Interpretatorn är integrerad i webbservern. Stöder inbäddad php-kod i hmtl-kod.
- **ASP**: används vanligen med språket VBScript, men andra språk kan användas. Interpretatorn är integrerad i webbservern. Stöder inbäddad kod i hmtl-kod.
- **JSP**: används med språket Java. Integrerad i webbservern. Stöder inbäddad kod i hmtl-kod.
- **JavaServlets**: språk Java. Servlets liknar CGI genom att det är kod som skapar dokument, men för övrigt är det en annan teknik (*Guelich,* 2000, s194ff).
- **ColdFusion**: Interpretatorn är integrerad i webbservern. Gör större skillnad på skriptkod och html-kod (än php och asp), stöder inbäddade funktionsanrop i hmtl-kod.

Idag är system som PHP, ASP, JSP, JavaServlets m.fl. mycket vanligt förekommande, men CGI-system används fortfarande.
Notera att denna uppsats fortsätter med traditionen att kalla programspråk som används för programkörning på servern för *skript*språk, oavsett om det handlar om PHP, VBScript i ASP, Java i JSP eller i Servlets, eller något vanligt CGI-språk som C eller Perl.

Oavsett vilket skriptspråk som används, så innebär det alltid en säkerhetsrisk då klienter ges möjlighet att exekvera skript eller program på servern. Riskerna kan utgöras av:
- ogenomtänkt kodning/programmering i skripten. Detta inkluderar även databasanrop via skriptens databasfunktioner, filhantering i skripten, med mera.
- inbyggda brister och säkerhetsluckor i de olika språken (det vill säga konstruktionsfel),
- felaktig konfigurering av systemet.

Inbyggda säkerhetsluckor/buggar och felaktig konfigurering är problem som är mer specifika för respektive skriptspråk och dess konstruktion. Olika konfigurationsalternativ samt olika typer av säkerhetsluckor kan variera mellan olika språk och även mellan olika versioner av samma språk.

[2] Interpreterande: skriptet översätts/kompileras till maskinkod vid varje körning/sidvisning, vilket kan ge något sämre prestande än med färdigkompilerade program skrivna i till exempel C eller C++.

Problem beroende på *ogenomtänkt kodning* beror mer på hur säkerhetsmedveten webbprogrammeraren är eller inte. Ogenomtänkt eller slarvig kodning kan till exempel innebära:

- otillräcklig kontroll av inkommande data från formulär, URL:er eller kakor (cookies),
- olämplig filhantering eller olämplig tilldelning av filrättigheter i skripten,
- användning av lättgissade lösenord,
- hårdkodade lösenord (det vill säga lösenord visas i skriptkoden),
- användning av lättgissade namn för databaser, tabeller med mera,
- olämplig hantering av sessioner eller sessionsvariabler,
- olämplig konstruktion och/eller kontroll av databasanrop.

Det sistnämnda innebär att olämplig användning av databasspråk i skriptens databasfunktioner också anses utgöra ogenomtänkt skriptkodning. Det kan gälla databaser som MS SQL, MySQL, PostgreSQL och så vidare. I denna uppsats behandlas framför allt MySQL.
Gällande inkommande data (från användarna) så bör all sådan data betraktas som en potentiell risk, och därför kontrolleras/valideras på lämpligt sätt (*Dimov*, 2004).

Det kan även finnas egenskaper specifika för vissa språk som gör att programmerare lättare begår vissa typer av misstag i de språken, som till exempel *globala variabler* som i vissa fall kan ge sämre säkerhet mot intrång. PHP hade till exempel fram till för något år sedan globala variabler aktiverade i sin grundkonfiguration, men har sedan ändrat på det.

I grunden handlar intrång via ogenomtänkt skriptkod om att inkräktaren med hjälp av en vanlig webbläsare utnyttjar de möjligheter som webbprogrammeraren (omedvetet) tillhandahåller i sina skript. Genom att inte ta sig tid att fundera på de mer eller mindre udda användningsområden som skriptet kan tänkas få i händerna på olika användare, så bäddar webbprogrammeraren själv för intrång via webbplatsens publika webbsidor.

3 Metod

En kombination av undersökningsmetoder valdes för att pröva hypotesen, nämligen:

- **Litteraturundersökning**: ger teoretiskt och praktiskt underlag för både enkäten och experimentet.
- **Enkät**: ger ett praktiskt inriktat underlag från skarpa miljöer. Samt visst underlag för experimentet.
- **Experiment**: ger ett praktiskt inriktat underlag från en kontrollerad och stabil testmiljö. Kan bekräfta det teoretiska underlaget från litteraturundersökningen samt uppgifter från enkäten.

Tillsammans kompletterar och bekräftar dessa metoder varandra, och ger ett starkt underlag för prövningen av hypotesen.

3.1 Litteraturundersökning

Syftet med litteraturundersökningen var att samla in teoretiskt underlag samt detaljerad praktisk information om olika intrångsmetoder. Detta användes sedan som underlag för utformningen av enkäten samt för experimentet och dess olika tester.
Litteraturundersökning fokuserade på olika intrångsmetoder gällande serverskripkod och dess system, men även i viss mån intrångsmetoder mer generellt.

3.2 Enkät

Syftet med enkäten var följande:

- Visa att intrång baserade på svagheter i skriptkod och dess system förekommer i *skarp miljö*, det vill säga att det förekommer i praktiken och inte enbart i teorin/litteraturen.
- Bidra med underlag för utformning av tester i experimentet.
- Styrka och bekräfta resultaten från de övriga två undersökningsmetoderna.

3.3 Experiment

Experimentets syfte var att testa ett urval intrångsmetoder för att visa att de fungerar i praktiken, och för att bekräfta och stärka resultatet från de övriga undersökningsmetoderna, samt att ge underlag för hypotesprövningen.
Experimentet utfördes i en *stabil* och *kontrollerbar* lokal miljö, där ett antal olika tester utgjorde olika värden på en *oberoende invariabel*, och respektive testresultat utgjorde olika värden på den *beroende utvariabeln*.
De intrångsmetoder som användes i testerna utformades och utfördes av mig, med resultatet från litteraturundersökning och enkät som underlag.

Alternativa undersökningsmetoder
Ovanstående metoder valdes efter att ha övervägt några andra alternativ, till exempel att enbart använda en *större och djupare frågeundersökning*. Men det bedömdes dock som mycket osäkert eftersom IT-säkerhet är en känslig fråga för företag och webbplatser (se Inledning - Bakgrund). Risken bedömdes som stor att det skulle bli svårt att hitta villiga företag.

En annan metod som övervägdes var att utföra en *fallstudie* av till exempel en webbyrå som utvecklar webbplatser, och som hanterar säkerhetsfrågor hos många olika webbplatser/klienter. Men även detta bedömdes kunna bli minst lika svårt på grund av säkerhetsskäl, både gällande webbyråns egna system och framför allt deras ansvar och sekretess gentemot sina klienter/kunder.

4 Genomförande

Här beskrivs genomförandet för de tre metoder som användes: litteraturundersökning, enkät samt experiment.

4.1 Litteraturundersökning

Observera att här redovisas endast ett urval av olika intrångsmetoder (se Avgränsningar). Det finns ytterligare skriptbaserade och närliggande metoder för intrång, och olika varianter av de olika metoderna, beskrivet i litteraturen och som inte tas upp här. Till exempel. vid hantering av kakor (cookies) och vid användning av sessioner (*Williams & Lane,* 2002, s 358ff).

I texten talas om åtkomstmetoder som GET och POST, och med detta menas olika åtkomstmetoder som används (av HTTP-protokollet) för att överföra variabelvärden från klientens webbläsare till servern och serverskriptet. Enligt (*Jensen,* 2000, s 598):

- GET används då variabler överförs via URL:en, det vill säga i den sökväg som används då klienten anropar servern. Till exempel. om man vill överföra en variabel med namnet "test" och som har värdet 3 till sidan index.php, så kan sökvägen se ut enligt:
  `http://www.exempel.se/index.php?test=3`
  Sökvägen kan till exempel bestå av en klickbar länk i html-koden, eller kan skrivas direkt i webbläsarens Location-fönster.
- POST används för variabler som överförs via textfält, lösenordsfält och så vidare som kan finnas i formulär i webbsidans html-kod som visas hos klienten.

Intrång baserade på svagheter i serverskriptsystem hos en webbplats beror bland annat på programmeringsmissar, dålig programlogik och brist på dagligt underhåll av hela systemet. Denna typ av intrång kan utföras med en vanlig webbläsare via webbplatsens egna öppna, offentliga webbsidor. Att skydda sig mot den här typen av intrång kräver ständig uppmärksamhet hos de programmerare, tekniker, webbmästare m.fl. som utvecklar, underhåller och driver en webbplats (*McClure,* 2002, s 710 ff).
De svagheter som redovisas här gäller inbyggda språkspecifika brister, ogenomtänkt skriptkod samt ogenomtänkt konfigurering av skriptsystem.

4.1.1 Inbyggda språkspecifika säkerhetsbrister

Inbyggda språkspecifika brister samt buggar i skriptsystemet ligger strax utanför kärnan för denna uppsats men ändå inom dess avgränsning. Som ett exempel på denna typ av fel visas i följande exempel den s.k. punkt-buggen i skriptspråket ASP. Det är från en tidigare programversion, och felet förekommer därför troligen inte längre i någon större omfattning bland dagens webbplatser.

Punkt-bugg i ASP
Genom att infoga en punkt i slutet på URL:en vid anrop av en asp-sida, kunde inkräktaren få tillgång till sidans källkod (*McClure, 2002, s 720*), som i värsta fall (med ogenomtänkt kod) kan ge inkräktaren tips om lösenord, databasnamn med mera. Exempel:

```
http://www.exempel.se/index.asp.
```

Microsoft släppte en säkerhetsfix som åtgärdade problemet, men säkerhetsfixen gav upphov till en ny brist där man kunde ersätta punkten i filnamnet med dess ascii-värde i hexadecimal form (2E) och fortfarande få tillgång till källkoden, det vill säga:

```
http://www.exempel.se/index%2easp
```

Anmärkning: procenttecknet % betyder att följande två tecken tolkas hexadecimalt.

Analys
Denna typ av inbyggda fel beror på av tillverkaren gjorda konstruktionsfel i skriptsystemen, och oftast brukar tillverkarna tillhandahålla säkerhetsuppdateringar (patchar) relativt omgående när bristerna väl blir upptäckta och kända. Dock, även om säkerhetsuppdateringar är tillgängliga inom kort tid hos en webbplats så kan denna typ av fel orsaka skada, till exempel därför att man på webbplatsen helt enkelt inte känner till problemet och inte installerar uppdateringen. Och som även framgår i ovanstående exempel, så kan en säkerhetsuppdatering innehålla nya fel som inkräktare upptäcker och utnyttjar tills en ny säkerhetsuppdatering släpps (*Danesh,* S2002, s 7).

4.1.2 Svagheter i ogenomtänkt skriptkod

Nedan följer exempel på olika områden och situationer där ogenomtänkt serverskriptkod kan utnyttjas av inkräktare så som det beskrivs i litteraturen.

4.1.2.1 Variabler

Om skriptsystemet stöder s.k. *globala variabler,* som omfattar variabler från klienten via åtkomstmetoderna GET, POST och så vidare, och om globala variabler är aktiverade i konfigurationen, kan det öka risken för intrång. Enligt *Bakken* (2004, kap. 5 Security) så kan i en ogenomtänkt kod inkräktare då bifoga eller "injicera" variabelvärden med till exempel GET-metoden in i php-variabler som egentligen är avsedda att endast användas inuti skriptet.

I följande PHP-exempelkod (**Figur 4-A**) som antas ligga i en sida med namnet "loggain.php" visas hur variabel $login kan "injiceras" utifrån via till exempel GET. Variabel $password är en variabel som kommer från ett inloggningsformulär på webbsidan, och $login används enbart i skriptkoden som en flagga i en villkorssats.

```
if ($password == $password_from_db)
{
   $login = 1;
}

if ($login)
   echo "Inloggning lyckades.";
else
   echo "Inloggning misslyckades.";
```

Figur 4-A Exempel på olämplig PHP-kod som medger injicering av variabelvärde.

Om inkräktaren inte vet lösenordet men känner till att $login används som flagga, så kan denne istället skicka med $login=1 i URL:en när sidan anropas, till exempel:

http://www.exempel.se/loggain.php?login=1

Ovanstående förutsätter att inkräktaren känner till eller kan gissa variabelnamnet login, samt att login inte är initierad tidigare i skriptsidan (det vill säga inte är satt till exempel $login=0).

I PHP kan globala variabler stängas av i konfigurationen för att öka säkerheten. Det görs i filen **php.ini** med parametern

register_globals = off

För att sedan kunna hantera variabler som ska skickas från klienten (till exempel via GET eller POST), så används PHP:s s.k. superglobala variabler varmed man specifikt kan ange vilken typ av variabel man hämtar.
Till exempel. för att läsa en variabel som skickas via POST och som har namnet *minVariabel* så anges:

$_POST['minVariabel ']

istället för att direkt använda variabelnamnet $minVariabel. På så sätt är man även förvissad om att variabeln skickats via ett formulär (POST) och inte via URL:en (GET). Anmärkning: detta förutsätter att *track_vars* är aktiverad, och det är den alltid från och med php version 4.0.3.

Analys
För att förhindra detta intrång bör alltså variabel $login initieras, alternativt att man inte använder globala variabler (det vill säga inte har det aktiverat i konfigurationen).

4.1.2.2 Filuppladdning

En webbplats kanske vill ge användarna möjlighet att via ett formulär på en webbsida ladda upp egna filer, till exempel bildfiler eller andra dokument som de kan lagra på servern. Enligt *Allen & Hornberger* (2002, s 287) kan en ogenomtänkt skriptkod medföra att en inkräktare kan ladda upp filer som kan vara till skada för webbplatsen.

Följande exempelkod (**Figur 4-B**)visar innehållet i en fil "visakod.php" som inkräktaren laddar upp med hjälp av det filuppladdningsformulär som webbplatsen har på sin webbsida.

```
<?php

highlight_file ("../loggain.php");

?>
```

Figur 4-B Exempel på PHP-kod innehållande en funktion som visar källkoden i en fil.

Koden i "visakod.php" exekverar en php-funktion som returnerar källkoden som finns i inloggningsfilen "loggain.php".
När filen är uppladdad exekverar inkräktaren skriptkoden i "visakod.php" genom att skriva sökvägen i sin webbläsares url-fält, till exempel:

http://www.exempel.se/uppladdadefiler/visakod.php

Resultatet blir att källkoden i filen "loggain.php" visas i inkräktarens webbläsare. Det kan medföra att inkräktaren får reda variabelnamn, tabellnamn och så vidare som inkräktaren sedan kan användas i ytterligare intrång.

Som förebyggande åtgärd för att försvåra denna typ av intrång kan skriptkoden som hanterar de uppladdade filerna kontrollera att filerna är av rätt (förväntad) typ innan de placeras i destinationskatalogen. Alternativt (eller som ett komplement) kan även kontroll ske att filändelsen inte motsvarar en exekverbar fil, och att man sätter filrättighet på filen så att den inte går att exekvera.

Analys
Denna typ av intrång förutsätter att inkräktaren är insatt i PHP och har "turen" att hitta en webbplats där webbprogrammeraren inte tänkt på filsäkerhet när koden skrevs. Men om en sådan miss finns på en välbesökt webbplats med många besökare och filuppladdningar, så ökar förstås risken att någon med experimentlusta eller sabotage i sinnet förr eller senare upptäcker och utnyttjar den svagheten.

I detta fall torde det vara säkrast och enklast för webbprogrammeraren om man i koden definierar exakt vilka filtyper som ska vara tillåtna (till exempel enbart word- och exceldokument), det vill säga att filerna är av förväntad typ, och att man inte släpper igenom övriga filtyper.

4.1.2.3 SQL-injektion

Enligt (*Allen & Hornberger,* 2002, s 300), (*Bakken*, 2004, kap. 5 Security) m.fl. så är *SQL-injektion* en metod som går ut på att inkräktaren lägger till eller ändrar i befintlig SQL-kod i skriptet i syfte att se, ändra eller radera data i databasen.

Ex 1: SQL-injektion som lägger till extra SQL-sats
Till exempel. så kanske en webbplats har en webbsida som innehåller ett sökformulär där användarna kan skriva sökord i ett textfält. När de trycker på formulärets "sök-knapp" så tar serverskriptet emot sökordet i en variabel som används i ett databasanrop, och servern returnerar resultatet av sökningen i användarens webbläsare.
Ett sådant databasanrop kan i skriptet se ut som till exempel:

```
SELECT * FROM users WHERE username='$username'
```

Anropet hämtar de data från tabell *users* där fält *username* innehåller sökordet.
Om en inkräktare istället för ett sökord skriver till exempel följande i sökformulärets textfält:

```
'; DELETE * FROM users; --
```

så blir det resulterande databasanropet istället:

```
SELECT * FROM users WHERE username=''; DELETE * FROM users; --'
```

Det vill säga inkräktaren lägger till en SQL-sats genom att avsluta den ursprungliga SQL-satsen med semikolon. Ovanstående satser resulterar i att en sökning utförs i alla poster som har en tom sträng i fält *username*, och därefter raderas samtliga poster i tabell *users*, det vill säga hela tabellens innehåll raderas.

Anmärkning: de avslutande två bindestrecken (--) är vanligtvis ett kommentarstecken i SQL. I detta exempel medför det att databasen bortser från det avslutande original-citattecknet som hamnade i slutet (och att databasen därmed inte betraktar anropet som felaktigt).

Ex 2: SQL-injektion som modifierar/utökar befintlig SQL-sats

En inkräktare kan (med rätt förutsättningar) till exempel få tillgång till lösenord från en tabell genom att manipulera vad som presenteras i ett sökresultat på en webbsida som har en sökfunktion.
I ett exempel från (*Bakken*, 2004, kap. 5 Security) visas principen för hur detta kan gå till om databasen stöder UNION så att sökning från flera tabeller kan utföras samtidigt och kombineras. Det vill säga flera SELECT kan göras i samma SQL-sats.
Det ursprungliga databasanropet i skriptet kan se ut som:

```
SELECT id, name, inserted, size FROM products
  WHERE size = '$size'
  ORDER BY $order LIMIT $limit, $offset;
```

Figur 4-C SQL-sats. Källa: *Bakken*, 2004, kap. 5 Security

Om inkräktaren lyckas tilldela variabel $size följande (till exempel via GET):

```
'
union select '1', concat(uname||'-'||passwd) as name, '1971-01-01', '0' from usertable;
--
```

Figur 4-D UNION som kan infogas i SQL-sats. Källa: *Bakken*, 2004, kap. 5 Security

så kan sökresultatet som visas på webbsidan komma att visa samtliga användarnamn och lösenord i par (separerat med ett bindestreck). UNION medför här att sökning görs med SELECT i två tabeller, *products* och *usertable*, och i den andra SELECT sammanfogas användarnamn och lösenord och placeras i fält *name* som visas i sökresultatet. Detta sker alltså i en och samma SQL-sats.

Angående **Figur 4-D** notera det inledande enkelcitattecknet som avslutar sökvärdet för *size* i den ursprungliga SQL-satsen, samt det avslutande SQL-kommentarstecknet (--) som gör att databasen bortser från resten av den ursprungliga SQL-satsen (det vill säga från och med ORDER BY...).

Analys

I båda ovanstående exempel på SQL-injektion krävs att inkräktaren är insatt i syntaxen för både skript- och databasspråket, och känner till eller kan gissa tabellnamn och tabellernas fältnamn. I exempel 2 även kunskap om vilka datatyper som används i de olika fälten (till exempel datumformatet).
I exempel 2 (union-exemplet) framgår inte tabellernas hela utseende, struktur, fältnamn, datatyper och så vidare. Dessutom känner vi inte till databassyntaxen. Om båda tabellerna dock är union-kompatibla (*Connolly & Begg,* 2002, s 145ff), det vill säga har samma struktur

(det vill säga samma antal kolumner, datatyper och så vidare), så bör denna typ av intrång vara möjlig att genomföra i praktiken.

4.2 Enkät

4.2.1 Inledning

Enkäten bestod av 2 stycken frågor angående intrång. Frågeformuläret låg på en webbplats hos BTH. Mejl med förfrågan inklusive länk till frågeformuläret skickades ut till 87 stycken webbplatser, de flesta kända och stora och i många fall även kommersiella.

Enkäten var utformad enligt följande:

- *Kvalitativt* inriktad, den undersökte i första hand intrångsmetoderna och deras praktiska tillvägagångssätt (den syftade alltså inte till att ge någon statistisk sammanställning eller jämförelse mellan olika intrångsmetoder och dylikt).
- Likalydande frågor i samma ordning för varje tillfrågad person (det vill säga hög grad av standardisering), samt hade *öppna frågor* (ej fasta svarsalternativ, det vill säga låg grad av strukturering).
- Enkätdeltagandet var helt *anonymt* för försöka få så många som möjligt svara. Detta med tanke på ämnets säkerhetsmässigt känsliga karaktär (se Inledning - Bakgrund).

Resultatet från enkäten analyserades enligt:

- Har det begåtts något/några intrång som utnyttjat svagheter i serverskriptkod eller dess system? Ett eller flera Ja stärker hypotesen.
- Vilka intrångsmetoder har använts. Resultatet användes som underlag för utformning av oberoende invariabel i experimentet, samt för att verifiera teoretiska uppgifter från litteraturundersökningen.

Närmare detaljer kring enkäten, dess upplägg, enkätfrågorna, lista med webbplatser, med mera finns i **bilaga 1**.

4.2.2 Redovisning och analys

Svarsfrekvensen låg på närmare 22%. Av 87 stycken tillfrågade under perioden 8/3 2004 -- 25/3 2004, inkom19 stycken svar enligt följande:

- Har haft intrång: 4 stycken.
- Har ej haft intrång: 7 stycken.
- Ville/kunde ej svara på grund av säkerhetspolicy, med mera: 8 stycken.

Deltagare som haft intrång

Av de tillfrågade som haft intrång hade 1 deltagare haft skriptintrång via SQL-injektion (skriptspråk ASP), samt 1 deltagare beroende på felkonfigurerat skriptsystem (PHP).
Se **Tabell 4.2-A**.

Tabell 4.2-A: enkätdeltagare som haft intrång

Nr	Inkom	Metod	Skada	Åtgärd
1	2004-03-09 9:54:04	SQL-injektion via ASP-skript.	"Modifiering av data i enstaka tabeller. Inget allvarligt som tur var".	Ändrade rättigheter för SQL konton. Extra hantering av citationstecken i indata.
2	2004-03-11 9:21:44	Utnyttjande av felkonfigurerad webbserver, en ftp-port låg öppen.	Filstrukturen var sabbad och disken mer eller mindre full. Servern användes som lagringsplats och illegal spridning av filer, filmer mm.	Hela servern installerades på nytt. "Numera finns det inga ftp-portar öppna förutom en som är skyddad av både verifiering av ip-nummer samt id + lösen."
3	2004-03-25 0:49:09	Utnyttjande av felkonfigurerat PHP-system, gav obehörig tillgång till lösenordsfiler.	[Svar ej lämnat].	PHP-konfigurering ändrades (safe mode aktiverades). Lösenord byttes.
4	2004-03-25 21:57:53	Utnyttjande av svagt lösenord, genom inloggning med default-konto som ej ändrats.	"Inga skador då enbart namn och epostadresser hade lagrats."	Ändrade lösenord. Raderade eventuella kvarvarande installationsfiler. Såg över skyddet för databaser och inloggning.

Kommentarer till ovanstående tabell:

- Nr 1: Vid följdfrågor jag ställde till deltagaren framkom att de hade lättgissade tabell- och fältnamn: UserData, UserName, UserPassword. Databasen var MS SQL 2000.
  För att fastställa orsaken till intrånget hade man utfört en rekonstruktion, där man lyckades göra en SQL-injektion som lade till en UPDATE-sats till den ursprungliga SQL-satsen, och som modifierade data i databastabellen så att samma uppdateringar återskapades som efter inkräktarens intrång. Nämnas kan också att man lagt mycket möda på att ta reda på orsaken ("2 dagars slit"), och att intrången upphörde efter att åtgärder vidtagits.
- Nr 2: det var webbhotellet de anlitade som brustit med konfigurationen.
- Nr 3: "safe mode" används i PHP bland annat vid delad server och dylikt, till exempel på webbhotell med mera. Till exempel. så att endast ägaren till en (system-)fil kan läsa filen.

Anmärkning: eftersom enkätdeltagandet var anonymt så kan följaktligen inte källorna redovisas. Dock stärks svarens trovärdighet av att flera deltagare ändå lämnade uppgift om vilken webbplats de kom ifrån, och jag hade även mejlkontakt med en av deltagarna som haft intrång. Det framgår alltså i flera fall av svaren att det inte är vilken besökare som helst som råkat komma till webbplatsen där svarsformuläret låg och svarat på enkäten.

Egentligen var det ytterligare 1 deltagare som svarade att de haft intrång via skript, och att deras webbsidor hade modifierats av besökare men att "Inga åtgärder vidtogs".
Deras svar visade dock ganska tydligt att de skämtade, eftersom det gällde en s.k. wiki[3]-sajt, där den normala funktionen är att användarna kan redigera webbsidorna själva. Jag mejlade dock en följdfråga till dem, men fick inget svar. Därför bedömdes svaret som oseriöst och ströks från enkätsvaren.

[3] Wiki-webbplats: alla besökare kan redigera och ändra sidornas innehåll. PS: en tid senare fick denna sajt begränsa möjligheten för användarna att redigera sidorna, på grund av för mycket sabotage.

Deltagare som haft intrångsförsök
Bland de som uppgivit att de inte haft intrång men som varit utsatta för kända intrångsförsök, uppger 1 av de tillfrågade att det skett via skriptsystemet. Ytterligare 1 kan eventuellt ha varit relaterat till skriptsystem. Några citat angående vilka intrångsförsök de haft:

- "Enkla skript hackförsök."
- "Bara allmänna scanningsförsök. Till exempel på webbservern ser vi ständiga försök att hacka med kända problem. Vi har en linux server och de flesta kända problem finns på windows så det löser sig av sig själv".
- "Mest portscanningar och maskar"

Deltagare som ej haft intrång
Bland de som ej haft intrång svarade några av deltagarna på frågan om vilka förebyggande åtgärder man vidtagit:

- Svar: "Vi har analyserat alla flöden och försökt förutse risker, dessutom har vi tagit in en konsult som är expert inom området (av typen före detta hacker) som har hjälpt oss att hitta brister i systemet som vi sedan har åtgärdat".
- Svar: "* Rutiner för patchning av system och applikationer
  * Realtidsövervakning
  * Externa och interna brandväggar
  * Regelbundna granskningar av externa revisorer
  * Filintegritetsskydd för bevakning av kritiska systemfiler
  * Antivirusskydd
  * Säkerhetspolicy".

En svarade alltså att man anlitat en konsult av typen f.d. hacker, och som då troligen har testat olika typer av intrång under kontrollerade former, och eventuellt omfattar detta även skriptintrång. Och en annan svarade bland annat att de utvecklat rutiner för patchning, vilket förstås även kan omfatta eventuella skriptsystem.

Deltagare som ej ville svara på frågorna
Eftersom det nämns i inledning på denna uppsats att intrång kan vara ett känsligt ämne att diskutera för webbplatser och företag, och därför kan göra det svårt att få dem att vilja samarbeta i en sådan här enkät, så redovisas även här några av de skäl de tillfrågade vänligen uppgivit till varför det inte ville svara på frågorna.

- Svar: "Vår policy är att inte berätta något om den teknik eller de säkerhetsåtgärder banken använder..."
- Svar: "Tyvärr tillåter inte våra säkerhetspolicies att vi lämnar ut information om våra datasystem."
- Svar: "Vi väljer av olika skäl att generellt inte svara på enkäter likt denna."

Från en av våra större myndigheter tog man telefonkontakt och förklarade att man hade en säkerhetspolicy som ej tillät utlämnande av information om deras datasystem. De kunde dock berätta att Försvarets Radioanstalt då och då gjorde intrångsförsök hos dem under kontrollerade former för att testa säkerheten hos deras webbplats.
Ytterligare några angav tidsbrist som skäl att inte delta.

Analys
Det var 2 stycken som svarade att man haft intrång relaterade till svagheter i skriptkod respektive brister i konfigurationen av skriptsystemet. För intrånget med SQL-injektion uppgavs att man rekonstruerat intrånget för att fastställa intrångsmetoden. Där finns förstås en risk att man dragit fel slutsatser och att intrånget egentligen utförts på ett annat sätt. Men man har tydligen lagt ned en hel del arbete på rekonstruktionen, och deltagaren redovisade även detaljerat och seriöst i sina svar hur man gått tillväga, så jag utgår ifrån att uppgifterna stämmer.

Det andra intrånget berodde på fel eller olämpligt konfigurerat skriptsystem. Det handlade om rättigheter för lösenordsfiler som egentligen ingår i serversystemet, men där PHP ger möjlighet att styra detta på grund av att alternativen i serversystemet anses otillräckliga (*Bakken*, 2004, kap. 22 Safe Mode). Detta används av många ISP:er (Internet Service Provider), internetleverantörer eller webbhotell.
Så även detta svar anses ligga inom ramen för intrång via skriptsystem.

Vad som inte framgår av enkätsvaren är om intrången skett med hjälp av webbläsare eller inte. Gällande svaret med SQL-injektion så kan det mycket väl ha utförts med en webbläsare, eftersom det troligen är naturligast/enklast för inkräktaren.

Noteras kan även att intrånget i svar nr 2 i tabell **Tabell 4.2-A** berodde på felkonfiguration hos det webbhotell man anlitade. I detta fall gällde det själva servern, men det kunde förstås lika gärna gälla konfigurationen av serverskriptsystemet. Det finns alltså all anledning att vara ständigt uppmärksam då man använder webbhotell, eftersom man själv inte har någon kontroll över konfigurationen och därför att webbhotellet när som helst kan ändra konfigurationen.

Angående de som svarat att de haft *försök* till intrång, så finns det förstås frågetecken *hur* man konstaterat detta. Det framgår inte om man loggat verksamheten, analyserat loggfiler eller har någon typ av övervakningssystem och så vidare, eller om det helt enkelt är mer eller mindre kvalificerade antaganden. De lämnade svaren låter dock inte på något sätt orimliga, och många av de tillfrågade webbplatserna var stora och etablerade sajter som troligen har både goda resurser och kunniga tekniker som övervakar systemen.

4.3 Experiment

Som tidigare nämnts (se Metod) bestod experimentets testmiljö av en webbplats bestående av skriptsidor med ogenomtänkt kod (enligt underlag från litteraturundersökningen och enkäten). Varje test i experimentet utgjordes av en viss intrångsmetod, som motsvarar testtillfällets s.k. *oberoende invariabel*.

Experimentet bestod av följande:

- **Experimentmiljö**: en skript- och databasbaserad *lokal* webbplats innehållande webbsidor med osäker skriptkod som kunde nås med en vanlig webbläsare. *Lokal* innebär att webbplatsen ej är ansluten till Internet.
  Detta bedömdes ge en *stabil* och *kontrollerbar* experimentmiljö.
- **Oberoende invariabel**: intrångsmetod. Olika intrångsmetoder motsvarade olika variabelvärden.

- **Beroende utvariabel**: resultat av intrångsförsök. Kunde anta något av värdena "Ej intrång" eller "Intrång". Det noterades även vad intrånget eller intrångsförsöket resulterade i (till exempel om data kunde läsas, kopieras, modifieras, raderas och så vidare, eller om något annan märkbar förändring inträffade), för att underlätta senare analys och diskussion.

De olika intrångsförsökena i testerna planerades och utfördes av mig enligt det underlag som framkom ur litteraturundersökningen och enkätsvaren. I testerna redovisas även förslag på motåtgärder, och i vissa tester utförs även kompletterande tester för att undersöka motåtgärder.

De tekniska specifikationerna och övrig information om webbservern, skriptsystemet, databasen, samt innehållet i de olika skriptsidorna på servern finns i **bilaga 2**.

Nedan redovisas och analyseras resultaten av experimentets olika tester. I hela experimentet används PHP skriptsystem och MySQL databas (se bilaga 2).

4.3.1 Test 1: Variabler

Testdatum: 2004-04-01.
Scenario: inkräktaren vill komma in på webbplatsens lösenordsskyddade sidor, men har inte tillgång till lösenord.
Syfte: kontrollera i PHP om ett variabelvärde kan skickas in (injiceras) i skriptet via URL:en (GET) så att inloggning sker.

Inledning
Test 1 i sida *testsida2.php* (se bilaga 2) simulerar ett inloggningsförfarande (koden är medvetet ogenomtänkt, och i detta fall på mer än ett sätt). Variabel $login används i koden som en flagga för inloggningen.

Testförfarande:
Sida *testsida2.php* öppnades i webbläsaren med följande URL i dess Location-fält:

http://dgc-67de713ba5c/testsida2.php?login=1

Resultat:
INTRÅNG. Inkräktaren kunde logga in utan att ange lösenord.

Analys:
Intrånget förutsätter att inkräktaren känner till variabelnamnen i källkoden och vad variabel $login har för funktion. Denna kunskap kan inhämtas via andra metoder som beskrivs i denna uppsats, eller genom att gissa sig fram.
Eftersom globala variabler är aktiverade OCH eftersom variabel $login ej är initierad, så kan inkräktaren injicera ett variabelvärde i php-variabeln $login via URL:en.

Test, motåtgärd 1:
Satte register_globals = off i php.ini.
Upprepade testförfarandet ovan.
Sidan visade "Inloggning misslyckades."

Resultat av motåtgärden: EJ INTRÅNG.

Test, motåtgärd 2:
Återställde först till register_globals = on i php.ini.
Lade till och initierade variabel $login = 0 enligt:

```
$login = 0;
if ($password == "hemligt")
  $login = 1;
```

Upprepade testförfarandet ovan.
Sidan visade "Inloggning misslyckades."
Resultat av motåtgärd: EJ INTRÅNG.

Analys:
Då exekveringen kommer till raden där variabeln initieras i skriptkoden, så ändras det värde som sattes av inkräktaren från 1 till 0, och följaktligen sker ingen "inloggning".

4.3.2 Test 2: Filuppladdning

Testdatum: 2004-04-06.
Scenario: en inkräktare vill se webbsidornas skriptkällkod, för att få reda på variabelnamn, databasnamn, tabellnamn och ta del av annan information som kan användas för vidare intrång.
Syfte: kontrollera i PHP om det går att ladda upp en exekverbart skript som avslöjar källkoden i webbsidorna.

Inledning
På sida *index.php* finns ett formulärfält för filuppladdning, som användes för att ladda upp filen *visa_kod.php* (bilaga 2). Denna fil innehåller en PHP-funktion som kan visa (avslöja) källkoden för en sida som anges som parameter, i detta fall sida *testsida2.php.*
Skriptet i sida *testsida2.php* tar emot och placerar den uppladdade filen i katalog */testkat/* (katalogen ligger alltså i roten, se även bilaga 2 för beskrivning av webbplatsens katalogstruktur).

Testförfarande:
Från sida *index.php* laddades filen *visa_kod.php* upp till webbplatsens */testkat/* katalog.
Därefter: i webbläsaren öppnades filen *visa_kod.php* genom att skriva följande i webbläsarens sökvägsfält (Location) och trycka Enter:

```
http://dgc-67de713ba5c/testkat/visa_kod.php
```

Resultat:
INTRÅNG. Hela källkoden för sida *testsida2.php* visades i webbläsarens fönster.

Analys:
När filen *visa_kod.php* öppnades kördes dess skript som innehåller funktionen som visar källkoden för *testsida2.php*. Genom att ladda upp en exekverbar fil på en webbplats med ogenomtänkt kod och felaktiga filrättigheter, kunde inkräktaren ta del av information i webbplatsens källkod. Till exempel. det hårdkodade (det vill säga i klartext skrivna) lösenordet samt namn på databastabell. Detta bäddar för vidare intrång.

Motåtgärd:
Den uppladdade filen kan kontrolleras så att den inte är en exekverbar fil (till exempel genom att kontrollera filändelsen), eller att filens rättighet sätts så att den inte går att exekvera.

4.3.3 Test 3: SQL-injektion som lägger till extra SQL-sats

Testdatum: 2004-04-07.
Scenario: webbplatsen har en webbsida innehållande ett sökformulär där sökord kan skrivas in för att söka i databasen. En inkräktare vill sabotera webbplatsens databas genom att utnyttja svagheter i databashanteringen.
Syfte: kontrollera i PHP om det i sökfältet kan skickas med en SQL-sats som läggs till (injiceras) den befintliga SQL-satsen och som raderar en tabell.

Inledning
På sida *index.php* finns ett formulär med ett textfält för sökning i tabell *users* i databas *dvc001* (bilaga 2). Skriptet i sida *testsida2.php* tar emot sökordet i en variabel som infogas i databasanropet, som sker via PHP:s funktion för databasanrop *mysql_query()*. Skriptet skriver även ut databasanropets utseende på sidan (det vill säga hur anropet ser ut efter injiceringen av den text som skrivits in i sökordsfältet).
Databasanropet i skriptet ser ut så här:

```
SELECT * FROM users WHERE username='$searchtext'
```

Anmärkning: innehållet i sökordsfältet hamnar i variabel $searchtext.

Testförfarande, försök 1:
Följande text skickades iväg via sökfältet på sida *index.php*:

```
'; DELETE FROM users #
```

Anmärkning 1: för att radera alla poster i en tabell i MySQL så är syntaxen DELETE FROM, det vill säga utan asterisk (*) mellan DELETE och FROM.
Anmärkning 2: kommentarstecknet i MySQL är # (det vill säga inte -- som i vissa andra databaser).

Sida *testsida2.php* visade hur anropet ser ut efter injiceringen:

```
SELECT * FROM users WHERE username='\'; DELETE FROM users #'
```

Resultat, försök 1:
EJ INTRÅNG. Tabellen hade inte raderats.

Analys, försök 1:
PHP hade infogat ett s.k. escape-tecken (det vill säga backslash: \ ) före det inledande citattecknet (vilket innebar att DELETE FROM... ingick i det värde som jämfördes med fält username, och alltså inte betraktades som ett kommando). Detta är en förvald inställning hos PHP-konfigurationen och kallas **magic_quotes_gpc.** Det används för att skydda mot den här

typen av fel då användarna skickar upp tecken som inte är tillåtna i databasfältvärden, till exempel citattecken (" och '), backslash (\ ) och NULL (*Allen & Hornberger,* 2002, s 242).

Testförfarande, försök 2:
Eftersom första försöket ej resulterade i intrång på grund av en konfigurationsinställning hos PHP, så gjordes ytterligare ett försök med en annan konfigurationsinställning.

Först sattes **magic_quotes_gpc = off** i PHP-konfigurationsfilen php.ini.
Därefter upprepades testförfarandet i försök 1 ovan.
Sida *testsida2.php* visade:

```
SELECT * FROM users WHERE username=''; DELETE FROM users #'
```

samt ett varningsmeddelade från PHP:

...supplied argument is not a valid MySQL result resource...

Resultat, försök 2:
EJ INTRÅNG. Tabellen hade inte raderats.

Analys, försök 2:
För att undersöka varför intrånget ej lyckades, kontrollerades de resulterande SQL-kommandona som visades på sida *testsida2.php.* De konstaterades vara korrekta genom att kopiera och klistra in kommandona direkt i databasens kommandofönster, och då raderades tabellen (men när de skickades via PHP raderades alltså ej tabellen). En stunds sökande i PHP:s dokumentation antydde att PHP:s funktion för MySQL-databasanrop, mysql_query(), ej stöder flera SQL-satser i samma anrop (det nämndes i ett inlägg av en användare i forumet för mysql_query(), och var alltså inte en uppgift från tillverkaren själv). Samtidigt visas i PHP-dokumentationen från tillverkaren ett liknande exempel som antyder att denna typ av på SQL-injektion är möjligt, så kanske har PHP infört denna begränsning i senare versioner men inte uppdaterat all sin dokumentation, eller så har man bara uttryckt sig slarvigt. Det kan också vara ett medvetet säkerhetstänkande att inte offentliggöra den här typen av säkerhetsåtgärder, i syfte att försvåra för hackare som försöker ta sig förbi deras åtgärder.

Det vill säga: denna typ av SQL-injektion, där extra SQL-satser läggs till den ursprungliga satsen, går inte att utföra i PHP version 4.3.4 på grund av att PHP-funktionen för MySQL-databasanrop ej tycks stödja flera SQL-satser i samma anrop.

Motåtgärd
I detta test framgår motåtgärderna i de olika försöken där intrång ej lyckades. Förutom att det i den aktuella php-versionen inte går att skicka flera SQL-satser i databasanropet, så kan en motåtgärd även vara att konfigurera systemet så att det automatiskt lägger till escape-tecken framför citattecken.

4.3.4 Test 4: SQL-injektion som modifierar SQL-sats

Testdatum: 2004-05-10.

Scenario: på en webbsida finns ett formulärtextfält "Ändra lösenord" där användare Loke kan ändra sitt lösenord som finns i databasen. Det görs genom att skriva in lösenordet i fältet och trycka på Ändra-knappen. Loke vill göra intrång genom att ändra administratörens lösenord så att han själv kan logga in som administratör.
Syfte: kontrollera om användaren kan ändra administratören Admins lösenord istället för sitt eget, genom att skriva SQL-kod i textfältet så att det injiceras i den ursprungliga SQL-satsen i serverskriptet.

Inledning
På sida *index.php* finns ett formulär med ett textfält för inmatning av nytt lösenord i tabell *users* i databas *dvc001* (bilaga 2). Skriptet i sida *testsida2.php* tar emot det nya lösenordet (eller den text som användaren skriver i fältet) i en variabel som infogas i databasanropet.
OBS: i PHP var **magic_quotes_gpc = off** i konfigurationen, så att php accepterar citattecken i anropet. Det konstaterades i föregående test att detta är en förutsättning för SQL-injektion ska lyckas då citattecken förs in i den ursprungliga SQL-satsen.

Databasanropet i skriptet såg ursprungligen ut så här:

```
UPDATE users SET password='$newpassword' WHERE username='Loke'
```

Anmärkning: innehållet i Ändra-lösenordfältet hamnar i variabel $newpassword.

Testförfarande:
Följande text skrevs i Ändra-lösenordfältet på sida *index.php*:

```
hack' WHERE username LIKE '%adm%' #
```

Anmärkning: uttrycket LIKE '%adm%' innebär att alla användare vars namn innehåller "adm" berörs.

Sida *testsida2.php* visade hur anropet ser ut efter injiceringen:

```
UPDATE users SET password='hack' WHERE username LIKE '%adm%' #' WHERE
username='Loke'
```

(men utan radbrytning). Koden efter # betraktas som kommentar, det vill säga databasen exekverade alltså följande:

```
UPDATE users SET password='hack' WHERE username LIKE '%adm%'
```

Resultat:
INTRÅNG. Lösenordet för Admin ändrades till "hack".

Analys:
Under förutsättning att **magic_quotes_gpc** inte är aktiverad i PHP-konfigurationen, så är SQL-injektion som enbart modifierar befintlig SQL-sats möjlig. Ovanstående exempel kan alltså leda till att inkräktaren får full tillgång till de delar av webbplatsen som normalt sett endast ska vara tillgängliga för administratören.

Anmärkning: enligt PHP:s rekommendationer bör magic_quotes_gpc vara aktiverat (*Bakken*, 2004, kap. 5 Security), och i så fall hade detta intrång inte varit möjligt.
Anmärkning: MySQL är i grundkonfigurationen inte skiftkänslig vid jämförelser med LIKE (*MySQL.,* Reference Manual, 2004). Därför hittades "adm" i username "Admin", och ovanstående sats och intrång kunde fungera.

Motåtgärd
Aktivera magic_quotes_gpc i PHP-konfigurationen.

5 Resultat

Här redovisas resultatet av analyserna i genomförandet, och de ursprungliga frågeställningarna besvaras samtidigt.

I samtliga av uppsatsens tre undersökningar framgår det att svagheter i serverskriptkod eller serverskriptsystem kan utnyttjas vid intrång på en webbplats. Litteraturundersökningen visar flera exempel på dessa typer av intrång, och enkäten ger några exempel, både från de som haft intrång och från de som enbart haft intrångsförsök. (Av enkäten framgår även att företagens datasäkerhet är ett känsligt område som man inte gärna talar offentligt om).
Experimentet bekräftar flera av rönen från de övriga undersökningarna, och experimentet tillsammans med litteraturundersökningen visar även att dessa typer av intrång kan utföras med en vanlig webbläsare via webbplatsens egna öppna offentliga webbsidor.

De undersökta typerna av intrång kan baseras på utnyttjande av:
- ogenomtänkt skriptkod,
- brister/buggar i skriptsystem,
- olämplig konfigurering av skriptsystem.

Ogenomtänkt skriptkod kan exempelvis vara då man i skriptkoden använder globala variabler på ett felaktigt sätt, oinitierade variabler, ogenomtänkt filhantering vid uppladdning av filer till webbplatsen, eller ogenomtänkt hantering av databasanrop. Detta kan medföra att inkräktaren kan föra in oönskade data som kan ställa till problem, och det kallas då att inkräktaren "injicerar" data. Det görs till exempel genom att skicka en variabel till skriptet via GET-metoden, eller genom att lägga till extra data i till exempel ett söktextfält som finns på en webbsida.

Några konkreta exempel på svagheter i ogenomtänkt skriptkod:
- Då en variabel ej initierats i skriptkoden och om globala variabler används, kan variabeln sättas (injiceras) utifrån via GET (det vill säga i URL:en).
- Då webbsidan har filuppladdning, och kontrollen av uppladdade filer ej är tillräcklig, så att inkräktare kan ladda upp skadliga exekverbara filer.
- Då webbsidan har en sökfunktion, och hanteringen av databasanrop i skriptkoden är ogenomtänkt, så att inkräktaren kan skriva skadlig databaskod i sökfältet (istället för ett sökord) som sedan hamnar (injiceras) i det ursprungliga databasanropet.

Intrång kan även baseras på ogenomtänkt konfiguration av serverskriptsystemet. Där finns ofta många parametrar att hålla reda på, så det kan vara lätt att göra misstag. Till exempel. några kritiska inställningar i PHP är:

- register_globals :Styr huruvida globala variabler ska kunna användas eller inte. Rekommenderas att inte vara aktiverad, det vill säga "Off".
- magic_quotes_gpc :Hanterar citattecken m.fl. tecken inuti strängvariabler genom att lägga till escape-tecken (backslash \ ) före tecknen så att de inte kan ställa till problem till exempel i databasanrop. Bör vara aktiverad, det vill säga "On".

Vilka intrång som är möjliga beror på konfigurationen, samt på typ av skriptspråk och vilken version det är. Detta framgick bland annat i experimentet som visade hur konfigurationen i PHP kan påverka intrångsmöjligheterna (till exempel med magic_quotes_gpc). Och i litteraturundersökningen redovisades "punkt-buggen" som var ett konstruktionsfel som fanns i en tidigare version av skriptsystemet ASP.
I den version av PHP som användes i experimentet var SQL-injektion som lägger till extra SQL-satser inte möjlig, på grund av att PHP inte tillät flera SQL-satser i samma MySQL-databasanrop.
En av enkätdeltagarna som använde ASP hade haft intrång med SQL-injektion med tillägg av extra SQL-sats, och det framgick att deltagarens ASP-system inte hade skydd mot varken flera SQL-satser i samma anrop, och inte heller mot införande av citattecken inuti strängen för databasanropet.
Ofta måste alltså inkräktaren vara någorlunda insatt i det aktuella skriptspråkets syntax och skillnader i olika versioner, samt eventuellt även i databasspråkets syntax.

Gemensamt för många av dessa intrång är att data som kommer från användarna inte kontrolleras och hanteras på ett bra sätt i skriptkoden. Till exempel. att man inte kontrollerar att en numerisk variabel verkligen innehåller enbart ett tal och inte en sträng, att man inte kontrollerar längden på en strängvariabel, och så vidare.
En annan återkommande faktor är användning av lättgissade variabelnamn, tabellnamn och så vidare. Om det inte utförs ordentlig kontroll av inkommande data, så kan användning av lättgissade namn vara ödesdigert. Genom att injicera variabelvärden via till exempel GET-metoden kan en inkräktare prova sig fram, och då underlättar det förstås om koden innehåller lättgissade namn.
Även om koden inte innehåller lättgissade namn så finns det intrångsmetoder där inkräktaren kan få tillgång till skriptens källkod och variabelnamnen med mera, till exempel som i exemplet med filuppladdning som beskrivs i experimentet.

Flera av de detaljer som redovisas i uppsatsen ger exempel på i vilka sammanhang förebyggande åtgärder lämpligen bör sättas in, till exempel att kontroll bör utföras av all inkommande data. Högt säkerhetstänkande i alla steg i utvecklingen av serverskriptsidorna, och full kontroll på hur skriptsystemet är konfigurerat, är viktiga grundläggande faktorer för att förebygga och försvåra intrång.

Omfattningen av de skador som kan orsakas av intrång via svagheter i skriptkod kan variera beroende på typ av intrång och på inkräktarens agerande. I en del fall kanske inkräktaren endast är inne och ser sig om i systemet, i andra fall kanske syftet är förstöra så mycket som möjligt. Intrång i en databas kan till exempel medföra att data kan raderas, modifieras eller läsas av inkräktare.

I enkäten framgick till exempel att hos en av enkätdeltagarna hade inkräktare modifierat data i en databastabell, men att det inte var kritiska data. I det fallet medförde intrånget några dagars extraarbete. Men i experimentet utfördes ett test där inkräktaren lyckades ändra administratörens lösenord. Sådana intrång medför att inkräktaren själv kan logga in som administratör och kanske få tillgång till en mängd känslig information (kontonummer, epostadresser, fler lösenord, och så vidare), inklusive underlag för ytterligare intrång med andra intrångsmetoder. I sådana fall kan skadorna bli mer omfattande, eller till och med katastrofala om att inkräktaren får tillgång till exempel företagsekonomiskt känsliga data eller andra typer av känslig information.

6 Slutsats

Intrång på webbplatser där inkräktaren utnyttjar svagheter i serverskriptkod och serverskriptsystemet tillhör en av de vanligaste intrångsmetoderna. För serverskriptsystem i allmänhet och för PHP och ASP i synnerhet (se även Avgränsningar samt Diskussion), dras följande slutsatser enligt de rön som framkommit i uppsatsens litteraturundersökning, dess enkät och dess experiment.

Intrång kan utföras med en vanlig webbläsare som hjälpmedel, via webbplatsens egna publika webbsidor. Följande grundläggande förutsättningar krävs för att denna typ av intrång ska kunna ske:

- **Att webbplatsen har en publik webbsida** (ett gränssnitt) som ger inkräktaren möjlighet att agera.
- **En eller flera svagheter i serverskriptsystem** på webbplatsen, vilket kan innebära en eller flera av följande alternativ:
  - *Ogenomtänkt serverskriptkod*. Till exempel dålig kontroll och validering av från klienten inkommande data, användning av lättgissade variabelnamn och tabellnamn, hårdkodade lösenord och tabellnamn.
  - *Olämplig konfigurering av skriptsystemet*. Till exempel att globala variabler är aktiverade, eller att man inte aktiverar skydd mot citattecken inuti strängar.
  - *Säkerhetsluckor i serverskriptsystemet*. Det vill säga buggar från mjukvarutillverkarens sida, och att webbplatsen inte har uppdaterat med de senaste säkerhetsuppdateringarna.
- **Viss kunskap hos inkräktaren** om det aktuella serverskriptspråket/skriptsystemet, och/eller om den aktuella skriptkoden.

Några vanliga metoder vid serverskriptintrång är:

- **SQL-injektion**. Kan till exempel utföras genom att inkräktaren använder ett sökformulär som finns på en webbsida, och där skriver in ytterligare en SQL-sats (istället för ett sökord) som i skriptkoden sedan läggs till det ursprungliga databasanropet.
- **Variabelinjicering**. Inkräktaren påverkar (sätter) interna variabler i skriptkoden via till exempel URL:en, genom att utnyttja att serverskriptsystemkonfigurationen har globala variabler aktiverade och att variablerna inte är initierade i skriptkoden.
- **Uppladdning av exekverbar kod** via ett filuppladdningsformulär på webbplatsens webbsida. Till exempel att inkräktaren laddar upp en egen serverskriptsida innehållande kod vars syfte är avslöja data eller att sabotera. När inkräktaren sedan öppnar sidan i sin webbläsare så exekveras skriptet på webbservern.

Gemensamt för ovanstående intrångsmetoder är att webbplatsens serverskriptkod och/eller konfigurering är ogenomtänkt. Vilka typer av intrång som är möjliga på en viss webbplats beror även på typ av skriptsystem och version. Olika skriptsystem och versioner kan ha olika förutsättningar för vilka intrång som är möjliga.

Skador som kan uppstå vid denna typ av intrång kan vara:
- Raderade eller modifiera data, till exempel i en databastabell. Om ingen backup finns kan data då vara helt förlorade.
- Kopierade/stulna data. Till exempel lösenord, så att inkräktaren får tillträde till andras konton eller andra delar av systemet.

Förebyggande åtgärder kan till exempel vara:
- Kontrollera/validera alltid all inkommande data, och stoppa eller rensa bort otillåten data. Både som kommer via GET, POST och kakor (cookies).
- Kontrollera eventuella inkommande (uppladdade) filer. Sätt även filrättigheter på uppladdade filer som begränsar möjligheter för oseriös/skadlig användning.
- Undvik lättgissade namn och lösenord i skriptkoden. Gå emot den direkt olämpliga traditionen att använda tydliga och beskrivande variabelnamn, tabellnamn med mera, eftersom det kan resultera i lättgissade namn.
- Undvik hårdkodade lösenord, tabellnamn med mera i skriptkoden.
- Undvik användning av globala variabler.
- Konfigurera skriptsystemet för hög säkerhet.
- Håll systemet uppdaterat med de senaste säkerhetsuppdateringarna.
- Utarbeta en tydlig säkerhetspolicy för programkodning, hantering av lösenord, filrättigheter med mera (och se till att personalen läser och följer den).
- Tänk som en "hackare", granska din skriptkod och konfiguration ur en inkräktares perspektiv.

7 Diskussion

Jag upptäckte redan tidigt, då jag sonderade terrängen i litteraturen inför val av ämne till uppsatsen, att det finns en mängd intrångsmetoder som baseras på olika tekniker och på olika typer av svagheter i olika delar av datorsystemen. Att i denna uppsats täcka alla typer av intrångsmetoder bedömde jag bli för omfattande arbete för enbart en person. Risken var att det hade blivit en ganska ytlig genomgång och redovisning av olika metoder. Jag var mer intresserad av att gå djupare inom något av dessa områden, så att detaljerna bakom tekniken kunde utforskas och praktiskt inriktade resultat redovisas som sedan kunde användas av uppsatsens målgrupp. Eftersom målgruppen är programmerare och utvecklare så har de troligen kapacitet att gå vidare efter genomläsningen och på egen hand och i andra källor upptäcka fler intrångsmetoder och tekniker. Eftersom serverskriptintrång enligt litteraturen tillhör den vanligare typen av intrång, så föll valet på denna typ av intrång.

Arbetet inriktar sig på serverskriptsystemen PHP och ASP eftersom det i stort sett är enbart dessa serverskriptsystem som tas upp i litteratur och andra källor. I experimentets tester har uppsatsen av praktiska skäl avgränsat sig till användning av PHP. I litteraturgenomgången och i enkätsvaren förekommer både PHP och ASP. Dessa system var i princip de enda system

som förekommer i litteraturens beskrivningar av serverskriptintrång. Troligen beroende på att de tillhör de vanligare serverskriptsystemen på marknaden och har existerat sedan senare hälften av 1990-talet.
Notera dock att principerna är detsamma för flera av intrången, oavsett om det är PHP eller ASP som beskrivs i texten, så dessa intrång bör fungera på liknande sätt även i andra typer av skriptsystem som JSP med flera.
De konfigurationsparametrar som tas upp gäller PHP, men i andra serverskriptsystem kan det förstås finnas liknande och andra/fler möjligheter i konfigurationen.

Tyngdpunkten i uppsatsens tre undersökningar ligger i litteraturundersökningen och experimentet, där återfinns majoriteten av ämnets teoretiska och praktiska material och underlag. Litteraturgenomgången gav en mängd teoretiskt och praktiskt underlag, och en del av detta har sållats bort vilket förhoppningsvis gjort uppsatsen mer koncentrerad än den skulle blivit annars.

Enkäten bidrar med några intressanta resultat från skarpa miljöer eller verkliga livet, och bekräftar och stärker dessutom resultatet från de övriga undersökningarna. Förväntningen på enkäten var från början låg på grund av att ämnets säkerhetsmässigt känsliga karaktär riskerade medföra ett lågt deltagande. Detta bekräftades också senare av några enkätdeltagare som uppgav just säkerhetsskäl eller säkerhetspolicy till varför de inte ville/kunde delta. Det externa bortfallet blev 78% vilket troligen beror på en kombination av orsaker: säkerhetsskäl, tidsbrist, ointresse och kanske att mejlet inte kommit fram till adressaten. Men med en svarsfrekvens på 22%, varav ett par enkätsvar som behandlade intrång av just den typ som låg inom uppsatsens avgränsningsområde, så får enkäten ändå betraktas som lyckad. Faktum är att hälften av de inrapporterade intrångsmetoderna låg inom uppsatsens avgränsning (det vill säga två stycken av fyra intrång). Syftet var inte att samla in statistiskt jämförelsematerial, utan endast att få in exempel från "verkliga livet" som kunde bekräfta övriga undersökningar och visa att denna typ av intrång inte endast existerar i teori och experimentmiljö.

Experimentet fungerade utan problem, och de olika testerna kunde genomföras som förväntat. Testerna i experimentet utformades efter inspiration och idéer från litteraturen och enkäten. Vikt lades vid att göra skriptkoden så enkel och koncentrerad som möjligt, så att inga onödiga parametrar behövde beaktas vid analyserna av testresultaten.
Experimentmiljön utgjordes av en lokal webbserver med en webbläsare på samma dator. Programvaran som användes var en vanligt förekommande webbservermjukvara som använder http-protokollet i sin kommunikation med klientens webbläsare. Det enda som skiljde den från en webbserver ansluten till Internet, är troligen att man i de olika testerna inte fick den tidsfördröjning som annars uppstår då data måste transporteras via Internet mellan server och klient. Testerna i mitt experiment bedömdes inte vara tidskritiska, så troligen hade testresultaten blivit exakt samma även om servern varit internetansluten.

Experimentet visade också att även facklitteratur bör granskas kritiskt. I samband med testet där SQL-injektion som lägger till extra SQL-sats, visade det sig att med experimentets version 4.3.4 av PHP så är denna typ av intrång ej möjlig med MySQL. Kanske har möjligheten funnits i tidigare versioner av PHP, men detta har ej kunnat konstateras. Inte ens PHP:s mycket detaljerade ändringsloggar säger något om detta. Men i PHP:s egen färska dokumentation finns dock exempel på denna typ av intrång, och i litteraturen enligt *Allen* (2002, s 221, 300) så har denna typ av intrång testats av författaren med PHP version 4.1 och med MySQL (och även med PostgreSQL). Denna versionskillnad kan tyckas vara en

obetydlig detalj, men för en inkräktare är det förstås relevant. Om det är författarna som är otydliga eller ger felaktig information, eller om det beror på versionsskillnader i PHP är alltså obekant i nuläget. Detta visar även att denna typ av intrång kräver en del kunskap eller tur hos inkräktaren.

Reflektioner kring orsaker och risker

I de exempel på intrångsmetoder som visas i uppsatsen ingår till exempel injicering av variabelvärden och databaskod, och i de fallen krävs oftast att inkräktaren känner till eller kan gissa flera olika faktorer (som till exempel variabelnamn, tabellnamn i skriptkoden, vilket skriptspråk, databasspråk med mera som används).
Det kanske därför kan tyckas som att sannolikheten för skriptintrång är låg eftersom det är flera detaljer som måste stämma för inkräktaren. Jag tror man riskerar att underskatta problemet om man resonerar så. Enligt min erfarenhet plus vad som framkommit i denna uppsats så finns det flera saker som pekar på att även din/er webbplats mycket väl kan ha svagheter i serverskriptsystemet och därmed riskerar att utsättas för intrång.

- Ogenomtänkt skriptkod kan vara vanligt förekommande därför att generellt sett tar programmeringskurser på högskolorna enligt min erfarenhet inte upp sådana säkerhetsfrågor i någon större utsträckning, eller till och med inte alls. Därför är risken stor att programmerare inte har tillräcklig rutin och erfarenhet av säkerhet i programkoden när de kommer ut till olika företag som programmerare, systemerare, webbmästare eller dylikt.
- Programmeringskurser lär ut att man ska använda tydliga, beskrivande namn på variabler och databastabeller med mera, vilket kan medföra att de även blir lättgissade och därmed gör det enklare för inkräktaren.
- Det är helt enkelt lätt för en programmerare att göra misstag eller förbiseenden bland alla detaljer i koden. Och ofta har programmeraren en deadline att passa, det är tidspress och i första hand gäller det att få koden att fungera som det är tänkt. Saker som säkerhet kan då komma i andra hand.
- Internet och webben gör att inkräktare kan vara fruktansvärt effektiva. Med hjälp av mjukvaruverktyg skannar och letar inkräktare på kort tid upp många sårbara webbplatser innehållande till exempel den typ av skriptsystem som passar inkräktaren bäst (*McClure,* 2002, s 706). En enda person kan utan någon större ansträngning på kort tid attackera och begå intrång på många webbplatser.
- Alla webbplatser riskerar att utsättas för intrång, oavsett hur små eller stora de är, beroende på att många inkräktare enbart gör det för nöjes skull (*Crume,* 2000, s 27ff). Skälet behöver alltså inte vara personligt mot en viss sajt, orsakat av vinstintresse eller dylikt. Det kan helt enkelt vara en kul grej och ett tidsfördriv, och därför spelar det ingen roll för inkräktaren vilken webbplats det är, man attackerar istället de första bästa man hittar som råkar vara sårbara.
- Ett antal undersökningar visar att många drabbas av attacker, vartannat företag enligt *Danielsson* (2004, Internet). Även de senaste årens mycket omfattande virusattacker indikerar att säkerhetstänkandet *generellt sett* är för lågt bland utvecklare, programmerare med flera.

Jag upprepar ett av råden till förebyggande åtgärder som nämns i uppsatsens slutsats, och som syftar till att öka säkerhetstänkandet: tänk som en "hackare". Granska din skriptkod och

konfiguration ur en inkräktares perspektiv, så att du kan korrigera misstagen innan någon annan hinner utnyttja dem.

7.1 Hypotesprövning

Det förväntade resultatet beskrivs i början av uppsatsen, nämligen att skriptsystembaserade intrång med hjälp av webbläsare är möjliga. I planeringen av de olika undersökningsmetoderna var det grundläggande målet att producera tillräckligt underlag för att kunna bekräfta eller förkasta hypotesen.

Erhållet resultat:

- De 3 utförda undersökningarna visar var och en för sig att denna typ av intrång kan förekomma.
- Litteraturen och experimentet visar även att denna typ av intrång kan utföras med webbläsare.
- Det finns alltså ett omfattande teoretiskt och praktiskt underlag som bekräftar hypotesen.

Det anses därför vara bekräftat att hypotesen stämmer.

7.2 Förslag till fortsatt arbete

Förslag 1:
Undersöka anledningen till att ogenomtänkt kod produceras, om det beror på okunnighet, ointresse, tidsbrist eller annat. Till exempel med en frågeundersökning, eller genom att göra ett experiment som testar ett antal programmerare.
Syftet kan då vara att ge underlag för hur utbildningar och kurser inom programmering och liknande kan förbättras med avseende på säkerhet.

Förslag 2:
Serverskriptsystem-baserade intrång utgör en av flera olika intrångsmetoder, och det handlar då om att ta sig in via webbplatsens offentliga eller publika sidor. Ett fortsatt arbete kan gå ut på att undersöka andra intrångsmetoder, till exempel intrång via de kanaler som normalt endast används av administratörerna. Även ytterligare metoder kan vara intressanta att utforska, eftersom sådana intrång även kan ge inkräktaren information/data/underlag som sedan kan användas för serverskriptbaserade intrång.

Övrigt
Ytterligare en tanke är att undersöka hur lätt eller svårt det är att utföra serverskriptsystem-baserade intrång i *skarpa miljöer*, det vill säga på webbplatser anslutna till Internet.
Anmärkning: liknande förslag som skulle ge resultat från skarpa miljöer har även lagts fram tidigare i denna uppsats (se Metod - Alternativa undersökningsmetoder).
Svårigheten med en sådan undersökning är att det knappast räcker att undersöka några enstaka webbplatser, det räcker inte som statistiskt underlag för att besvara frågan om det generellt sett är "lätt eller svårt" att utföra denna typ av intrång. Och för att resultatet från en sådan undersökning skall bli realistiskt bör man arbeta som inkräktare gör, nämligen att systematiskt söka upp misstänkt sårbara webbplatser och göra intrångsförsök där. Att få tillstånd från ett större antal webbplatser för att utföra intrångsförsök hos dem är troligtvis mycket svårt, med

tanke på resultaten gällande säkerhetsfrågor som framkommer i denna uppsats undersökningar.
Enkäten i denna uppsats visade i flera fall att sådana intrång förekommer i skarp miljö på Internet, men det räcker förstås inte som underlag för att avgöra om det generellt sett är lätt eller svårt. Uppsatsens experiment använde en webbserver som visserligen inte var ansluten till Internet, men enligt min bedömning så hade det blivit likadana testresultat även om den varit ansluten till Internet (se Bilaga 2 samt Diskussion). Dock, en undersökning kunde utföras för att klarlägga eventuella avvikelser mellan uppsatsens experimentresultat och ett liknande experiment på en enstaka webbplats i skarp miljö.

8 Referenser

8.1 Litteratur

Allen, Jeremy & Hornberger, Charles., Mastering PHP 4.1, Sybex inc, 2002.

Andersson, Ross., Security Engineering. A guide to building dependable distributed systems, John Wiley & Sons inc, 2001.

Brenton, Chris m.fl., Active defense. A comprehensive guide to network security, Sybex, 2001.

Chirillo, John., Hack attacks revealed. A complete reference with custom security toolkit, John Wiley & Sons inc, 2001.

Connolly, Thomas & Begg, Carolyn., Database Systems. A practical approach to design, implementation, and management, 3:e uppl, Addison-Wesley, 2002.

Crume, Jeff., Inside Internet security. What hackers don't want you to know, Addison-Wesley, 2000.

Danesh, Arman, m.fl., Safe and secure. Secure your home network and protect your privacy online, Sams publishing, 2002.

Felton, Mark., CGI Internet Programming with C++ and C, Prentice Hall, 1997.

Guelich, Scott, m.fl., CGI Programming with Perl, 2:a uppl, O'reilly, 2000.

Home, Alexl., Professional ASP techniques for webmasters, Wrow press ltd, 1998.

Honeynet, the project., Know your enemy. Revealing the security tools tactics, and motives of the blackhat community, Andra tryckningen, Addison-Wesley, 2002.

Jensen, Stig, m.fl., Datakommunikation, Liber, 2000.

Ladd, Eric & O'Donell, Jim., HTML, Java och CGI, Prentice Hall, 1998.

Maiwald, Ross & Sieglein William., Datasäkerhet i praktiken, Pagina, 2002.

McClure, Stuart, m.fl., Hacking i fokus. Hemligheter och lösningar för nätverkssäkerhet, 3:e uppl, Andra tryckningen, Pagina, 2002.

Welling, Luke & Thomson, Laura., PHP and MySQL web development, 2:a uppl, Sams publishing, 2003.

Williams, Hugh E. & Lane, David., Web Database Applications with PHP & MySQL, O'reilly, 2002.

8.2 Artiklar & dokumentation från Internet

Anmärkning: för vissa källor krävs medlemsskap eller lösenord för att få tillgång till artiklarna. Till exempel. Computer Swedens artikelsamling kräver ett "månadens lösenord".

Bakken, Stig, m.fl., PHP Documentation/Manual, http://www.php.net/, Hämtad 2004-02-18.

Bjerre, Lisa., TV4:s nyhetssajt hackad, Internetworld, 2004-04-22, http://internetworld.idg.se/ArticlePages/200404/22/20040422163010_IW765/20040422163010_IW765.dbp.asp, Hämtad 2004-05-09.

Danielsson, Lars., Symantec presenterar dåliga nyheter, Computer Sweden, 2004-03-19, http://computersweden.idg.se/ArticlePages/200403/18/20040318173924_CS559/20040318173924_CS559.dbp.asp, Hämtad 2004-03-29.

Dimov, Jordan., JSP Security, http://www.developer.com/java/article.php/883381, 2004. Hämtad 2004-02-18.

Hilley, Sarah., Hacker breaches 8 million credit card accounts, Computer fraud & security, 2003. Hämtad 2004-05-13. ELIN@Blekinge: http://bth.lub.lu.se.miman.bib.bth.se/elin/loadf/?f=search

Internetworld., Sveriges bästa sajter 2003, Internetworld, 2003-03, http://internetworld.idg.se/ArticlePages/200306/03/20030603120702_IW179/20030603120702_IW179.dbp.asp, Hämtad 2004-03-14.

Kärrman, Jens., Porrkupp i natt mot regeringen, Aftonbladet, 2001-03-14, http://www.aftonbladet.se/vss/nyheter/story/0,2789,39541,00.html, Hämtad 2004-03-26.

MySQL., Reference Manual, http://dev.mysql.com/doc/mysql/en/index.html, 2004. Hämtad 2004-05-05.

Sandberg, Dan & Höij, Magnus., Få vill erkänna intrång, Computer Sweden, 2003-09-05, http://computersweden.idg.se/ArticlePages/200309/04/20030904155100_CS011/20030904155100_CS011.dbp.asp, Hämtad 2004-03-29.

9 Appendix

9.1 Bilagor

9.1.1 Bilaga 1: Enkäten

9.1.1.1 Upplägg

Enkätens upplägg var enligt följande:

- Ett svarsformulär (se nedan) lades upp på en webbadress som tillhör BTH. Det framgick tydligt på flera ställen i texten att enkätdeltagandet var anonymt.
- En förfrågan (se nedan) innehållande en länk till svarsformuläret skickades med epost till 87 stycken webbplatser (se nedan).
- Svaren som lämnades i svarsformuläret samlas upp, dels via epost till mig och dels i en fil på hemsidan (för att minska risken att svar försvann på grund av eventuella tekniska fel eller dylikt).

Urvalet av webbplatser gjordes enligt följande:
Underlag hämtades från tidskriften Internetworlds årliga sammanställning av "100-bästa sajter" för år 2003 (Internet: *Internetworld*, 2003), av vilket de flesta webbplatser på listan bedömdes vara relativt stora och välbesökta. Många av dem var även kommersiella, och därför eventuellt mer utsatta för intrångsförsök.
Epostadresserna till webbplatserna införskaffades genom att gå in på respektive webbplats och söka reda på dem. Ett mindre antal hade mejlformulär istället för epostadresser, och några enstaka hade ingen kontaktinformation alls (antingen ville de inte bli kontaktade, eller så hade de glömt att lägga ut kontaktinformation).

Varje svar kontrollerades för att upptäcka eventuella oseriösa svar och dylikt. På grund av enkätens inriktning var den inte känslig för eventuellt *spam,* därför att om någon skickade in flera identiska svar så togs endast ett av de svaren med i analysen (under förutsättning att svaret för övrigt verkade seriöst, till exempel att en beskriven intrångsmetod bedömdes fungera teoretiskt och praktiskt).
Förfrågningarna gjordes huvudsakligen via epost, och i några fall via meddelandeformulär på respektive webbplats (då information om epostadress saknades på deras webbplats).

9.1.1.2 Frågorna i svarsformuläret.

Svarsformuläret fanns på adress http://www.student.bth.se/~algu02/survey.php och innehåller två frågeställningar för den som haft intrång respektive inte haft intrång.

Om ni haft intrång:	Om ni INTE haft intrång:
Fråga 1: Vilka dåvarande svagheter hos Er utnyttjades vid intrånget? (till exempel svagheter i er skriptdesign, konfigurering, mjukvara och så vidare). Beskriv gärna så tekniskt detaljerat som möjligt:	Fråga 1: Vilka åtgärder har Ni vidtagit för att förebygga intrång? (det vill säga angående skriptdesign, konfigurering, säkerhetspolicy och dylikt). Beskriv gärna så tekniskt detaljerat som möjligt:
Fråga 2: Vilka följder/skador hade intrånget? Vilka åtgärder vidtogs?	Fråga 2: Har ni varit utsatta för FÖRSÖK till intrång (enligt er vetskap)? Om ja, vilken typ av intrångsförsök?

9.1.1.3 Mejlutskick.

Följande förfrågan skickades till samtliga webbplatser som ingick i enkäten.

Subject: Riskanalys inom intrångsäkerhet

Till:
webbmaster eller webb-ansvarig

Hej!

Mitt namn är Alexander och jag arbetar med examensarbetet "Riskanalys inom intrångsäkerhet på webbplatser" hos BTH.

Jag har en mini-enkät innehållande två (2) frågor på följande adress:

http://www.student.bth.se/~algu02/survey.php

och jag undrar om någon tekniskt insatt person hos Er har möjlighet att besvara de frågorna?
Det tar högst några minuter.

Full anonymitet garanteras. Varken Ert namn eller adress skall uppges och kommer inte att visas någonstans.

::::::::::::::::::::::::::::::::::::::

Kontakta gärna min handledare på Blekinge Tekniska Högskola för bekräftelse att detta är seriöst:

Bo-Krister Vesterlund
Epost: bo-krister.vesterlund@bth.se
Telefon: 0457 - 38 55 64

::::::::::::::::::::::::::::::::::::::

Med vänlig hälsning,
Alexander Gustafsson
Epost: algu02@student.bth.se

9.1.1.4 Tillfrågade webbplatser.

Listan består av de 87 webbplatser som enkäten skickades till.

buzz.bazooka.se	www.foraldrar.com	www.proprint.se
www.adlibris.se	www.forsakringskassan.se/	www.quickresponse.nu
www.aftonbladet.se	www.fotosidan.se	www.resume.se
www.ahlens.com	www.fritidsresor.se	www.riksdagen.se
www.ams.se	www.ginza.se	www.rsv.se
www.arla.se	www.hemnet.se	www.sandrewmetronome.se
www.atg.se	www.ica.se	www.sf.se
www.avanza.se	www.idg.se	www.sf-anytime.com
www.blocket.se	www.ikea.se	www.shockplay.com
www.bokfynd.nu	www.infomedica.se	www.sj.se
www.bokus.com	www.inwarehouse.se	www.skandiabanken.se
www.bovision.se	www.kent.nu	www.skolutveckling.se/skolnet
www.brafilm.com	www.komplett.se	www.smartson.se
www.bth.se	www.konsumenternasforsakringsbyra.se	www.sourze.se
www.bytbil.com	www.konsumentverket.se	www.spray.se
www.cdon.com	www.livenetworks.se	www.sr.se
www.cint.se	www.lovesearch.se	www.studera.nu
www.clasohlson.se	www.lunarstorm.se	www.susning.nu
www.cyberphoto.se	www.mediekritik.nu	www.svd.se
www.dagensskiva.com	www.modernista.se	www.sweclockers.com
www.demoex.net	www.msn.se	www.svenskafans.com
www.di.se	www.mytravel.se	www.svenskaspel.se
www.discshop.se	www.ne.se	www.svenskaturistforeningen.se/
www.dn.se	www.netdoktor.se	www.svt.se
www.dustin.se	www.netonnet.se	www.tasteline.se
www.dvdforum.nu	www.novasol.se	www.thecricket.nu
www.eniro.se	www.nyteknik.se	www.ticnet.se
www.expressen.se	www.posten.se	www.tradera.com
www.fass.se	www.pricerunner.com	www.tv4.se

9.1.2 Bilaga 2: Experimentet

9.1.2.1 Teknisk specifikation för testmiljön

Testutrustningen i experimentmiljön bestod av en PC med:

- Intel Celeron CPU 1.7 GHz, 256 MB RAM,
- Windows 2000 5.00.2195 Service Pack 3 och delar av SP4 (med alla senaste "hotfixes" till och med februari 2004),
- Microsoft webbserver IIS 5.0 (Internet Information Services),
- Microsoft webbläsare Internet Explorer 6.0 SP1 (med alla senaste "hotfixes" till och med februari 2004).
- MySQL databas, server och klient version 3.23.56. Rättigheter är förvalda, det vill säga fulla rättigheter.
- PHP serverskriptsystem, version 4.3.4. Konfigurationsfilen php.ini innehåller förvalda inställningar.

Webbserverns domännamn/URL är: http://dgc-67de713ba5c/

För webbservern IIS användes i huvudsak den av tillverkaren förvalda konfigurationen, med följande undantag:

- katalogen för webbplatsens rot valdes,
- samt namnet på webbplatsens startsida ändrades till *index.php*.

Även för konfigurationen av MySQL och PHP användes av tillverkaren förvalda inställningar.

Testdatorn agerade både server och klient, vilket är en vanlig uppställning vid till exempel utveckling och test av webbplatser. Testresultaten bedöms motsvara samma resultat som hade erhållits på en internetansluten webbserver med samma innehåll och konfiguration som experimentmiljön.

Webbplatsens filkatalogstruktur var enligt följande:

```
[rot]
  |- testkat
```

Roten innehöll följande filer:

- index.php
- testsida2.php

Katalog *testkat* innehöll följande filer:

- Fil som laddades upp i test med filuppladdning.

Till webbplatsen skapades databas *dvc001*, innehållande tabell *users* med de tre fälten ID, username och password. Databastabellen fylldes med följande data i 5 poster:

ID	**username**	**password**
1	Admin	12345
2	Eva	qwerty
3	Gustav	asdfg
4	Vera	zxcvb
5	Loke	qazws

Anmärkning: koden för att skapa databasen och så vidare finns nedan i denna bilaga.
Anmärkning: val av tabellnamn med mera är medvetet ogenomtänkt.

9.1.2.2 Databas för testmiljön

Testdatabasen inklusive dess tabell samt data skapades med följande inmatning i databasens kommandofönster. Anmärkning: även denna kod är, liksom skriptkoden, medvetet ogenomtänkt.

```
CREATE DATABASE dvc001;

USE dvc001;

CREATE TABLE users (
  ID INT NOT NULL AUTO_INCREMENT,
  username VARCHAR (30) NOT NULL,
  password VARCHAR (10) NOT NULL,
  PRIMARY KEY (ID)
);

INSERT INTO users SET username='Admin', password='12345';
INSERT INTO users SET username='Eva', password='qwerty';
INSERT INTO users SET username='Gustav', password='asdfg';
INSERT INTO users SET username='Vera', password='zxcvb';
INSERT INTO users SET username='Loke', password='qazws';
```

9.1.2.3 Webbsidorna i testmiljöns webbplats

Anmärkning: koden i dessa bilagor innehåller oönskade radbrytningar på grund av att sidorna inte är lika breda som originalsidorna. För att återställa koden till originalutseende, kopiera och klistra in den i bredare sidor.

index.php

Flera olika test ligger i samma sida. De åtskiljs med URL-variabel *testnr* som används i sida *testsida2.php* för att skilja testerna åt så att rätt skriptkod exekveras där.

```
<html><head><title>index.php</title></head>
<body>
<h2>Testsida för Riskanalys inom intrångsäkerhet på webbplatser</h2>
<hr>

<form method="post" action="testsida2.php?testnr=1">
  Lösenord:<br>
  <input type="text" name="password" size="30">
  <input type="submit" value="Logga in" name="b1">
</form>
<hr>

<form method='post' action='testsida2.php?testnr=2'
enctype='multipart/form-data'>
  Ladda upp bildfil:<br>
  <input type='file' name='userfile' size='30'><br>
  <input class='inpButton' type='submit' name='ok' value='Ladda upp fil'>
</form>
<hr>

<form method="post" action="testsida2.php?testnr=3">
  Sök i databas:<br>
  <input type="text" name="searchtext" size="30">
  <input type="submit" value="Sök" name="b1">
</form>
<hr>
```

```
<form method="post" action="testsida2.php?testnr=4">
  Ändra lösenord:<br>
  <input type="text" name="newpassword" size="30">
  <input type="submit" value="Ändra" name="b1">
</form>
<hr>

</body>
</html>
```

testsida2.php

Innehåller hantering av flera olika tester, som åtskiljs med URL-variabel *testnr*.

```
<html><head><title>testsida2</title></head>
<body>

<?php


# TEST 1 ---------------------------------------------------
if ($testnr==1)
{
    echo "INLOGGNING.<br>";
    if ($password == "hemligt")
        $login = true;

    if ($login)
        echo "Inloggning lyckades.";
    else
        echo "Inloggning misslyckades.";
}


# TEST 2 ---------------------------------------------------
if ($testnr==2)
{
    echo "UPPLADDNING AV FIL.";
    # Temporärfilnamnet som php tilldelat:
    $uploadedFile = $_FILES['userfile']['tmp_name'];
    # Flytta filen till testkatalog  testkat, och döp den till
originalnamnet:
    move_uploaded_file ( $uploadedFile,
"testkat/".$_FILES['userfile']['name'] );
}


# TEST 3 ---------------------------------------------------
if ($testnr==3)
{
    echo "SÖKNING I DATABAS.<br>";

    # Upprätta anslutning till databasen och utför sökning (med ogenomtänkt
kod):
    mysql_connect("dgc-67de713ba5c", "", "") or die ("Unable to connect to
database");
    mysql_select_db("dvc001") or die("Unable to select database");

    $query = "SELECT * FROM users WHERE username='$searchtext'";
```

```
    echo "Databasanropets utseende:<br>$query<br>";
    $result = mysql_query( $query );
    echo "Antal hittade poster: " .mysql_numrows($result);
}


# TEST 4 ----------------------------------------------------
if ($testnr==4)
{
    echo "ÄNDRA LÖSENORD.<br>";

    # Upprätta anslutning till databasen:
    mysql_connect("dgc-67de713ba5c", "", "") or die ("Unable to connect to
database");
    mysql_select_db("dvc001") or die("Unable to select database");

    $query = "UPDATE users SET password='$newpassword' WHERE
username='Loke'";

    echo "Databasanropets utseende:<br>$query<br>";
    $result = mysql_query( $query );
    echo "Ändring utförd!";
}


?>

</body>
</html>
```

visa_kod.php

Används vid test av filuppladdning, och ligger då i katalog */testkat*

```
<?php

# Skriv ut källkoden för testsida2.php som ligger i roten:
highlight_file ( "../testsida2.php");

?>
```