Webbshop

Introduktion
Vår uppgift är att skapa en säker webbshop.
För att komma igång med en sådan krävs till att börja med ett par program.
Som webserver används Apache, som back-end programmerings-språk används PHP, för att spara användar-data och annan dynamisk information används MySQL.
All information mellan server och användare skickas därefter över en SSL socket.

Design
Hemsidan är mycket enkelt designad, men så att det ändå ser väldigt proffsigt ut.
Detta har jag Twitter bootstrap att tacka, det är ett css ramverk som är till för att just göra snabba men proffsiga startup hemsidor.

Säkerhetsaspekter

Bruteforce, dictionary och rainbow-table attack
En av de mest triviala säkerhetsaspekter en hemsida kan ha är hur man sparar en användares lösenord.
Ifall databasen på något sätt skulle läckas ut, så skall lösenord vara sparade på ett så bra sätt att de blir oanvändbara.
Detta går att åstadkomma genom att kombinera en hash-funktion och ett salt.
Jag har valt att använda hash-funktionen sha512 och ett slumpartat md5 salt vilket effektivt skyddar mot rainbow-table attacker.

Vid inloggning till hemsidan skulle en attackerare kunna utföra en bruteforce, dictionary. För att skydda sig mot detta kan man räkna antalet login-försök en IP har gjort inom en viss tid. Om antalet försök överstiger ett visst antal kan man kräva ett captcha test.
Till min hemsida har jag däremot inte lagt in ett sådant skydd.

Sessioner
Html är I grund och botten ”stateless”, det vill säga att html inte kan komma ihåg en användares tidigare handlingar. För att göra servern medveten om tidigare händelser så används sessioner. Dessa fungerar genom att skapa en sessions-variabel till varje användare och sedan binda denna till ett sessions-id.
För att använda sessions-variabeln till nästkommande request skickas sessions-id med, ofta I form av en cookie.

Sessions Hijacking
Sessions hijacking är ett samlingsnamn till då en attackerare stjäl en annan användares sessions-id.
Attackeraren får då tillgång till användarens sessions-variabel och kan använda hemsidan som att han var inloggad som den inte ont anande användaren.
För att skydda mot detta så har jag lagt in en identitets variabel I sessionen.
Den kollar så att IP och browser-klienten är samma, varje gång.

Adam Hansson Lyrén
dic11aha

Session Prediction
Vid session prediction attacker, så rent av gissar en attackerare ett sessions-id.
För en liten hemsida är det väldigt svårt att gissa ett aktivt sessions-id, men för stora sidor som facebook skulle risken vara betydligt större.
Min hemsida skyddar mot detta genom identitets-variabeln jag beskrev tidigare eftersom attackeraren nu måste gissa rätt på sessions-id I kombination med rätt ip och browser.

Session Sniffing
Vid session sniffing så försöker attackeraren hitta sessions-id genom att avlystna länken en användare har med servern. Som skydd mot detta har jag implementerat SSL. Detta gör att länken inte blir längre blir avlyssningsbar.

Session Fixation
Session fixation sker då en attackerare skickar en länk, med ett färdigt sessions-id I variabelfältet, till en användare som sedan loggar in på sitt konto.
Attackeraren har då tillgång till en session där användaren har loggat in för honom.
Mitt skydd mot detta är att alltid omgenerera sessions-id då auktoritet ändras, exempelvis efter inloggning.

Cross Site Scripting
Denna typ av attack fungerar som så att en attackerare skriver ett script som indata till ett formulär.
Då dessa data senare skrivs ut någonstans I html-koden och visas för vanliga användare så kommer scriptet att köras. Scriptet skulle till exempel ladda upp sessions-id till en tredjeparts server eller liknande.
Skydd mot detta är att alltid kolla så att input-data är giltig.
Det går att göra genom en rad olika funktioner I php så som striptags(), htmlspecialchars() eller genom att matcha vad du förväntar dig med “reguljära uttryck”.

Cross Site Forgery Request
Cross site forgery request fungerar på så sätt att en användare som loggat in på en pålitlig sida sedan besöker en fientlig sida så kan man genom script få användaren att utföra en handling på den pålitliga sidan.
På min webbshop skulle denna handling exempelvis kunna vara att en fientlig sida påtvingar ett köp av diverse produkter.
För att skydda mot detta tvingas användaren skriva in sitt lösenord för att bekräfta att handlingen var medveten.

SQL Injections
SQL injecktion kallas det då en attackerare skriver skadlig SQL-kod som indata i ett formulär. Då servern sedan skall använda indatan i en SQL fråga så kommer SQL-koden att exekveras.
Jag har i princip två skydd mot SQL injections, det första sker i samma veva som skyddet jag tidigare nämnt för cross site scripting där jag använder reguljära uttryck för att validera att indatan är giltigt.

Adam Hansson Lyrén
dic11aha

Det andra skyddet är att använda ”prepared statements”. Då behandlar databasen indatan som enskillda variabler istället för ren text som skulle kunna ändra resultatet för SQL frågan.

Expose PHP

Eftersom gamla så som nyare versioner av PHP kan ha kända defekter så kan det vara bra att inte visa vilket språk/version siten drivs av.
Detta kan åstakommas genom 3 enkla konfigurations-ändringar;

- expose_php = Off, vilket gör så att servern inte skickar information om vilken version av PHP som körs.
- Ändra namn på session cookien från ”PHPSESSID” till ”sid”.
- Ta bort filändelsen till php filer i URL:en. Detta görs med hjälp av URL Rewrite extensionen.

HTTPS/TSL

SSL är en om inte den mest triviala delen inom säkerhetsaspekterna för en hemsida. Utan SSL kan man aldrig vara säker på att någon lyssnar av dig och dina användare. Något som i tur innebär att man aldrig säkert kan överföra känslig data.
Till min webbshop har jag implementerat SSL med ett själv signerat certifikat.
Även ifall certifikatet inte går att verifiera på något pålitligt sätt, så överförs information nu över en säker anslutning.

Referenser: Föreläsnings-bilderna & notiser,
http://www.eit.lth.se/index.php?ciuid=560&coursepage=3877

Adam Hansson Lyrén
dic11aha