Webbsäkerhetslaboration med Webgoat

K A M I L H A K I M

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2010

Webbsäkerhetslaboration med Webgoat

K A M I L H A K I M

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2010
Handledare på CSC var Mads Dam
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/
hakim_kamil_K10029.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Webbsäkerhetslaboration med Webgoat

Sammanfattning

Examensarbetet undersöker hur en fiktiv kurs i webbsäkerhet kan lära ut webbsäkerhet genom att använda sig av Webgoat. Kursen är tilltänkt för studenter som studerar på fjärde året på datateknik-linjen. Laborationen som konstrueras i detta examensarbete behandlar de tre mest kritiska säkerhetsluckorna; XSS, SQL-Injection och XPATH. Målet med laborationen är att konstruera den på ett sätt som enkelt tilllåter modifikation och förhindrar fusk. En sådan laboration kunde konstrueras under förutsättningen att examinatorn står för en Webgoat-server samt att varje student får ett eget konto.

Web Security Exercise with Webgoat

Abstract

This paper examines how a course in web security can use Webgoat as a teaching tool. The course is to be elected by students that currently are studying at their fourth year. The exercise that is constructed contains the three most critical security flaws; XSS, XPATH and SQL-Injection. The goal is to construct the exercise in such a manner that it is simple to modify the content in order to prevent cheating. The reason for this is that there are a lot of solutions for the Webgoat exercises on the web. The exercise was successfully constructed under the premises that the teacher hosts the Webgoat server and every student gets a unique account.

Innehållsförteckning

Inledning

Syftet med detta examensarbete är att undersöka huruvida det går att sammanställa en webbsäkerhetslaboration med Webgoat som grund. Säkerhetslaborationen bör vara utformad på ett sätt som tillåter att uppgifterna går att byta ut utan att angrips metoden ändras. En viktig aspekt är att det finns färdiga lösningar till Webgoats uppgifterna på nätet. Laborationen måste alltså vara utformad på ett sådant sätt att fusk förhindras. Om säkerhetslaborationen blir sådan anses projektet lyckat. Examensarbetet kommer att omfatta en webbsäkerhetslaboration uppdelad i tre delar. Varje område omfattar en viktig säkerhetslucka. De tre områden som har valts ut är Xpath, SQL injection och XSS. Säkerhetslaborationen är tänkt att användas i en fiktiv webbsäkerhetskurs för studenter som läser fjärde året på Datatekniklinjen.

Bakgrund

En undersökning framställd av Websense[1] visar att 71% av alla webbapplikationer med legitimt innehåll även innehåller skadligkod. En anledningen till att hackare väljer att angripa legitima webbapplikationer är att dessa webbapplikationer har redan ett stort antal återkommande besökare som litar på webbapplikationen. Undersökningen visar även att 95% av all användargenererad data på de webbapplikationer som undersökningen omfattade är spam eller skadlig kod. Den visar också att 13.7% av sökningar på trendiga ord leder till skadliga webbapplikationer. Vilket betyder att en stor mängd webbapplikationer idag är utsatta för ett ständigt hot från illasinade människor.

Utvecklingen av HTTP

Internet har inte alltid varit en sådan farlig plats. När HTTP protokollet specificerades hade skaparna statiska hemsidor i åtanke. Till skillnad från dagens dynamiska webbapplikationer kräver inte statiska hemsidor några säkerhetskontroller. Anledningen till det är att användaren inte kan interagera med dem utan endast observera dem.Informations flödet är alltså enkel riktat från servern till klienten. Allt eftersom det blev möjligt att överföra större mängder data utvecklades även hemsidorna som började få dynamiskt innehåll. Till en början var det enkla modifikationer som byte av t.ex. färgtema. Cookies uppfanns som ett enkelt sätt att lagra klient-specifika uppgifter. Det fanns alltså inget behov av att skydda en Cookie då den lagrade okänslig information. Allt eftersom hemsidorna fick fler dynamiska komponenter som t.ex. login funktioner började även namn och i vissa fall lösenord sparas ner på Cookies. Detta för att kunna använda t.ex. ”remember me” funktioner. De statiska hemsidorna hade blivit ersatta med dynamiska webbapplikationer som har flera nivåer av ”business logic”. Nivåerna sträcker sig från den sida som användaren observerar i webbapplikation till företagets databaser. Det började nu bli mer relevant att skydda användarens Cookie. Om Bob(illasinnad person) stjäl Alices(oskyldig) Cookie för någon webbapplikation kan Bob utge sig för att vara Alice i den webbapplikationen. Cookies fick skydd som t.ex. endast webbapplikationen som skapade Cookien har rätt att modifiera och se innehållet. Anledningen för detta var att även om Alice gick in på en farlig sida skulle hon inte få alla sina Cookies stulna.

SessionID

Cookies fick ytterligare en ny egenskap, de fick s.k. SessionIDs. Detta fick de för att flera användare skulle kunna använda olika instanser av samma webbapplikation. Ett SessionID är en simpel textsträng som skickas med i varje header i HTTP trafiken till webbapplikationen. Utan SessionsID skulle både Bob och Alice besöka samma instans av en webbapplikation och följdaktligen skulle de störa varandra och framför allt se vad den andra användaren gjorde. Ett SessionID korresponderar

[1] http://www.bitpipe.com/detail/RES/1268779969_882.html

alltså mot en specifik instans av webbapplikationen. Detta ger upphov till att varje användare av applikationen endast ser sina egna uppgifter. Det finns två sätt för Bob att stjäla Alice instans av webbapplikationen, det ena tillvägagångssättet är att försöka stjäla Alice Cookie och det andra sättet är att försöka räkna ut hennes SessionsID.

Stöld av SessionsID

De följande sektionerna beskriver två olika sätt att stjäla SessionsID.

Bruteforce

Att försöka räkna ut SessionsID för en specifik användare kan vara väldigt svårt. Där emot kan Bob försöka räkna ut SessionsID för en mängd okända användare. Att räkna ut SessionsID kan alltså liknas med att fiska, det går inte att förutspå vilken fisk man får på kroken. Ett SessionsID kan vara väldigt enkelt att räkna ut om utdelningen av SessionsIDet är dåligt implementerat. Det kan t.ex. vara att SessionsIDn delas ut sekventiellt, vilket betyder att första användaren som loggar in och får sessionsID nummer 1 , nästa får 2 etc. Bob behöver i det här enkla fallet endast begära en Cookie och sedan räkna baklänges för att undersöka om något SessionsID fortfarande är giltig.Utvecklaren för webbapplikationen kan utföra två olika åtgärder för att förhindra SessionsID stöld. Utvecklaren kan dels se till att ett SessionID bara är giltig för en kortare tidsperiod samt att återvändande användare får ett nytt SessionsID vid varje inloggning. Den åtgärden leder till att även om Bob kan räkna ut ett SessionsID som tillhör Alice så har Alice redan vid det här tillfället hunnit få ett nytt SessionsID. Den här åtgärden förekommer i t.ex. SEBs och Swedbanks internetbanker. Det går inte att slutföra en överföring innan användaren har autentiserat sig vid två tillfällen, vid varje autentisering får användaren ett nytt SessionsID. Den andra åtgärden som en utvecklare kan göra är att se till att SessionsID inte tilldelas sekventiellt till användarna. SessionsIDt bör istället bero på faktorer som är unika för varje användare. Tänkbara faktorer är t.ex. IP-nr eller användarnamn. Utvecklaren bör alltså se till att det inte existerar något mönster i de SessionsIDn som webbapplikationen delar ut.

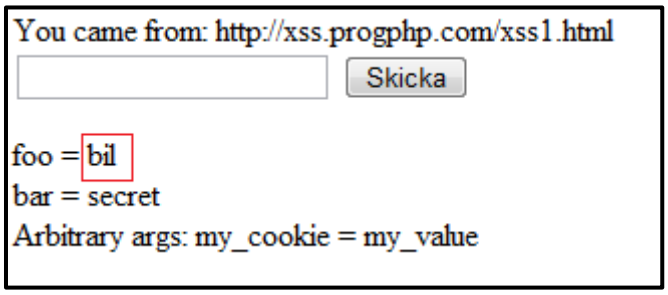
Cross Site Scripting

Istället för att försöka beräkna ett SessionsID kan Bob använda sig av en mer sofistikerad angreppsmetod, Bob kan använda sig av Phishing. Phishing är när Bob förmår Alice att på något sätt lämna ifrån sig sina inloggningsuppgifter eller sitt SessionsID. Tekniken bakom Phishing heter Cross Site Scripting(XSS). XSS förlitar sig på avsaknandet av korrekt hantering av indata. Ett mycket vanligt exempel är när en användare söker efter något i en sökfunktion i webbapplikationen, när sökningen är klar står det ”din sökning på : <sökord>” gav följande träffar. Om sökordet inte saniteras från skadlig kod kan Bob plantera ett helt html formulär i sökfältet. Formuläret i sin tur ber Alice logga in på nytt och sedan skickas dessa uppgifter till Bob. För att Bob skall kunna utnyttja denna sårbarhet måste sökordet förekomma i URL-adressen till sidan där resultatet visas, t.ex. www.minWebbApplikation.se/&search=bilar. Sökordet ”bilar” ersätts enkelt med koden som skapar formuläret, sedan skickar Bob denna adress till Alice. Den skadliga koden skickar i sin tur alltingen Alice Cookie eller inloggningsuppgifter till Bob. Denna typ av XSS attack kallas ”reflected XSS”. Anledningen till att det heter reflected beror på att inget sparas på webbapplikationen utan Bob reflekterar skadlig kod via webbapplikationen.

Demonstration av en XSS attack.

Här illustreras en XSS-attack. Webbapplikationen[2] som används för illustrationen är designad för att studera hur xss fungerar. I Figur 1 illustreras hur det ser ut när man söker på ”bil”.

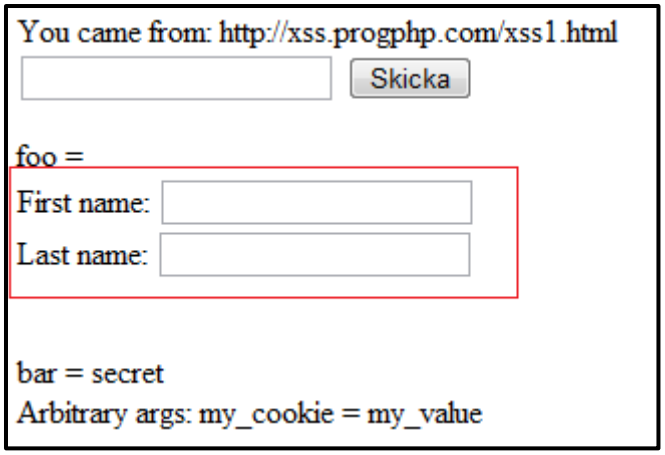
[2] http://xss.progphp.com/xss1.html



Figur 1 Demonstrerar en sökning på ordet "bil"

Webbapplikationen uppfyller alltså det viktigaste kravet för XSS, den visar användar inmatad data osaniterat. Detta utnyttjas i Figur 2 där sökströngen byts ut till:

```
"<form>First name:<input type="text" name="firstname" /><br />Last name:<input
type="text" name="lastname" /></form>"
```



Figur 2 Demonstrerar en sökning där sökordet är en attackstäng

Sökströngen resulterar i Figur 2 som är ett enkelt proof of concept. Sökordet saniteras ej utan blir till en del av webbapplikationen.

Motåtgärder till Cross Site Scripting

Här beskrivs vilka motåtgärder som kan implementeras i webbapplikationen samt hur funktionalliteten i webbläsaren kan utnyttjas.

Webbapplikationen

Anledningen till att det är svårt att förhindra XSS attacker är oftast att det är svårt att identifiera alla angreppsytor i webbapplikationen. Studdard och Pinto[3] anser att tre åtgärder bör utföras för att förhindra XSS attacker. Den första åtgärden är att validera indata, den andra åtgärden är att validera

[3] The web application hackers handbook, sid 423

utdata och den tredje är att finna och eliminera alla angreppsytor. Med angreppsyta menas en plats i webbapplikationen där data som är inmatad av en användare visas. Det svåraste steget är att finna och eliminera alla onödiga användarinmatningar. Det effektivaste sättet att finna alla dessa ställen i webbapplikation är genom att granska koden.T.ex. bör felmeddelanden som inte är hårdkodade undvikas till en så stor utsträckning som möjligt då dessa ofta är en bra angreppsyta för en reflected XSS. Sedan bör indata alltid valideras, webbapplikationen bör undersöka följande:

- Att indata inte är för långt
- Att indata endast innehåller tillåtna tecken
- Att indata matchar ett reguljärt uttryck[4]

Dessa regler bör användas så restriktivt som möjligt för att förhindra att skadlig kod kommer in i webbapplikationen. Den viktigaste åtgärden som bör utföras enligt Studdard och Pinto är att validera utdata. Även David A. Wheeler behandlar detta steg i sin bok[5]. Alla tre författare föreslår att utdata måste valideras. Den bör utsättas för samma rigorösa kontroll som vid validering av indata. Denna kontroll utförs endast för att eliminera risken för att man har missat att validera någon indata. Förutom att validera datat igen bör webbapplikationen även ändra teckenkodning. Detta krävs för att kunna oskadliggöra de special tecken som används vid XSS attacker. Exempel på dessa tecken är:

- < (mindre än)
- > (större än)
- = (lika med)
- ” (citattecken)
- % (procent)
- & (och)

Om dessa finns med bör dessa ersättas med deras motsvarighet i htmlteckenkodningen. Det här steget är nödvändigt för att desarmera dem. Den observante läsaren inser att dessa är tecken som har en speciell betydelse i html och javascript. Genom att byta teckenkodning uppfattar webbläsaren tecknet som en del av en skriven text och inte som körbar kod, vilket förhindrar XSS attacker.

Webbläsaren

Det finns ytterligare ett sätt att förhindra XSS attacker,nämligen genom att ange HTTPOnly flaggan. Denna flagga används på Cookien som webbapplikationen skapar. Detta leder till att Cookien endast kan manipuleras genom de headers som skickas från webbapplikationen. Denna åtgärd förhindrar att javascript får åtkomst till Cookies. När skriptet försöker komma åt Cookien returneras endast en tomsträng till skriptet. Tyvärr kräver HTTPOnly att webbläsaren stödjer det, om webbläsaren inte stödjer den ignoreras flaggan. De största[6] webbläsarnas senaste versioner, med undantag av Internet Explorer(IE), stödjer HTTPOnly. Anledningen till att IE stödjer det sen tidigare (IE6) är att HTTPOnly utvecklades av Microsoft i samband med utformningen av Service Pack 1 till IE6. HTTPOnly är alltså ett väldigt bra skydd mot XSS om webbläsaren stödjer det. Utvecklare som dock endast förlitar sig på HTTPOnly har ett väldigt svagt skydd mot XSS attacker eftersom inte alla webbläsare stödjer det.

[4] http://www.regular-expressions.info/
[5] Secure Programming for Linux and Unix HOWTO
[6] http://www.owasp.org/index.php/HTTPOnly

Hotbilden

En undersökning[7] visar att XSS, SQL-injection och XPATH tillhör de mest kritiska säkerhetsluckorna i webbapplikationer under år 2009. De tillhör även de mest förekommande säkerhetsluckorna[8].
Det är tämligen svårt att finna en rimlig förklaring till varför en säkerhetslucka som är så pass enkel att åtgärda är den säkerhetsluckan som med stora marginaler är den mest spridda. En förklaring kan vara att det är svårt att upptäcka varje angrepps punkt i en webbapplikation, en annan kan vara naivitet hos utvecklarna.

Social engineering

Phishing går att utföra på andra sätt än med hjälp av XSS. Angriparen kan t.ex. skicka ett mail till personer som använder sig av en viss webbapplikation och utge sig för att vara en administratör. I mailet skriver angriparen att varje användare snabbt bör logga in på en sida för att ändra sina inloggningsuppgifter då en fiktiv angrepp har ägdt rum. Angriparen har själv kontroll över den sidan som länkas till alla användare. Angriparen får alltså tillgång till många konton utan att ens behöva röra webbapplikationen ifråga.

Tekniken kallas Social Enginnering(S.E). Genom att få användaren att lita på angriparen kan denne få värdefull information utan att riskera något. Låt oss undersöka hur S.E kan fungera i verkliga livet. Alla som någon gång har besökt ett community på internet, t.ex. Facebook har med all säkerhet observerat hur villiga människor kan vara att dela med sig om information om sig själva. Ett exempel på det här är ”x fakta om mig”[9]. I en sådan enkät ställs en mängd frågor till användaren. Användaren svarar på dessa frågor och uppmanar sedan alla dess bekanta att även de skall svara på enkäten. En listig angripare kan utnyttja en sådan enkät på följande sätt. Genom att konstruera en liknande enkät själv och i den plantera frågor som ofta förekommer i ”hemliga frågan”. Hemliga frågan används ofta vid återställning av lösenord i t.ex. Hotmail samt flertalet andra webbapplikationer. Sedan sprider han denna enkät via t.ex. Facebook. När angriparens vänner sedan svarat på enkäten kan angriparen oftast utan problem ta över respektive mailkonto. Webbapplikationer har ibland även en funktion som skickar lösenordet till den mailadress som användaren angav när denne skapade sitt konto på webbapplikationen. Ett förlorat mailkonto kanske verkar harmlöst men i själva verket kan en angripare med kontroll över ett mailkonto även ta kontroll över flertalet andra konton på olika webbapplikationer. Detta visar att det inte räcker att webbapplikationen är säker utan även att dess användare måste vara säkerhetsmedvetna. Annars kan även den säkraste webbapplikation lätt knäckas då en slarvig användare har skrivit ner användarnamn och lösenord på en lapp på sitt skrivbord.

Verktyg

De två viktigaste verktygen för någon som vill angripa en webbapplikation är en intercepting proxy och en sårbarhets scanner. Båda beskrivs nedan.

Intercepting proxy

Teori

En Intercepting Proxy(I.P) tillåter angriparen att inspektera eller ändra på varje medelenande som skickas mellan servern och dennes webbläsare. Angriparen kan även undersöka vad som händer med webbapplikationen om för få parametrar skickas eller om parametrarna har får helt orimliga värden. Anledningen till att en I.P är en av de viktigaste verktygen för en hacker är att en I.P fångar upp alla HTTP/HTTPS trafik mellan hackerns webbläsare och webbapplikationen. För att en webbapplikation

[7] http://www.owasp.org/index.php/OWASP_Top_Ten_Project
[8] http://projects.webappsec.org/Web-Application-Security-Statistics
[9] X representerar antalet frågor, vilket kan variera.

skall vara av nytta för en användare krävs interaktion mellan användaren och webbapplikationen. Interaktionen sker över protokollet HTTP, vilket betyder att s.k. Headers skickas mellan användarens webbläsare och webbapplikationen. Dessa Headers består av data som beskriver för webbläsaren vad den skall visa eller vad webbläsaren begär att webbapplikationen skall utföra. Det här innebär att alla parametrar som en användare anger i sin webbläsare skickas till webbapplikationen via HTTP. Dessa parametrar är av intresse för en angripare då denne kan förbigå sanitering av data på klientsidan innan den skickas till webbapplikationen Det kanske mest förvånande är att även HTTPS trafik går att observera och modifiera på samma sätt som det okrypterade HTTP protokollet. Det fungerar på följande sätt; I.P programmet utger sig för att vara servern och låter webbläsaren öppna en krypterad SSL anslutning mot I.Pn. Sedan öppnar I.Pn en ny SSL anslutning mellan sig själv och den riktiga servern.

Detta tillåter en angripare att använda sig av en I.P för att utföra en s.k. Man In The Middle attack. Det betyder att en angripare lyssnar på någons HTTPS trafik. Det finns dock ett skydd mot MITM attacker av denna typ. Skyddet består av certifikat som är beräknade med hjälp av kryptografi. Problemet som uppstår för en angripare är att när en I.P används för att lyssna på HTTPS trafik så ändras dessa certifikat. Då webbläsaren märker att certifikatet är ändrat meddelas användaren. Denne måste då ta ställning till hur vida han skall acceptera det nya certifikatet. En försiktig användare vet att ändring i ett certifikat är en stor indikation på att något är fel. Detta är dock inget problem för en angripare när denne använder en I.P tillsammans med sin egen webbläsare.

Mjukvara

De mest använda I.Ps ingår i följande programpaket: OWASP Webscarab, Paros och Burp Suite. I dessa tre ingår flertalet andra verktyg för att analysera en webbapplikation. Det är dock I.Pn som är det viktigaste verktyget i samtliga programpaket. En av de andra verktygen som ingår är oftast ett program som utför Spidering. Spidering är ett verktyg som passivt eller aktivt kan skapa en virtuell karta av webbapplikationen. När spidering utförs passivt lyssnar endast verktyget på HTTP trafiken, det enda som syns på kartan då är de sidor som angriparen har besökt. Aktiv spidering är ett sätt att hitta gömda filer och sidor, det kan samtidigt vara väldigt farligt för webbapplikationen. Aktiv spidering är när programmet besöker varje sida automatiskt och försöker att använda varje funktion eller länk på webbapplikationen som den finner. Om verktyget hittar en administratörssida kan den t.ex. råka ta bort alla användare eller låsa dem ute från webbapplikation genom lösenordsbyte. Detta kan bidra till att dessa säkerhetsluckor täpps igen men också att all information av värde går förlorad. Spidering kan alltså vara ett väldigt bra verktyg för att finna dold information men också ett verktyg som bör användas med ytterst försiktighet.

Med hjälp av verktyget Cain and Able kan en angripare utföra en MITM attack av ett annat slag. Denna mjukvara kräver inte att en webläsare konfigrueras för att använda Cain and Able som en proxy. Den använder sig istället av en teknik som kallas Poision Routing. Poison Routing innebär att angriparen utger sin dator för att vara gatewayen mot internet i ett nätverk, vilket i många fall är en router. Det i sin tur betyder att alla HTTP trafik skickas via angriparens dator. Angriparen har alltså full kontroll över dataflödet och kan snappa upp lösenord och användarnamn eller SessionsIDn. Detta är bara en av de många funktioner som Cain and Able klarar av, resterande verktyg faller dock utanför ramen för den här rapporten. För att motverka denna typ av attack måste allting som skickas gå via HTTPS för att angriparen inte skall kunna lyssna på trafiken.

Sårbarhets Scanner

Teori

Att leta efter säkerhetsluckor för hand kan vara en väldigt tidskrävande process då det finns många olika attacker som måste prövas och variationer av varje attack. Sårbarhets Scanners (S.S) används för att snabbt finna många vanligt förekommande säkerhetsluckor i webbapplikationen. Detta kan snabbt ge en bild av var i webbapplikationen sårbarheter förkommer, tyvärr kan inte en S.S upptäcka alla säkerhetsluckor. För att en S.S. skall fungera på många olika webbapplikationer skulle det krävas att den hade någon form av artificiell intelligens. Anledning till detta är att angriparen måste förstå hur en webbapplikation fungerar för att kunna angripa den, vilket en S.S inte klarar av. En S.S scannar igenom koden för webbapplikationen. Den har dock ingen djupare förståelse för vad koden egentligen gör. S.S förstår med andra ord att variabeln k får ett nytt värde, men den förstår inte vad k representerar. En S.S kan alltså upptäcka att variabler i en webbapplikation kan ändras på ett inkonsekvent sätt men förstår inte på vilket sätt detta kan utnyttjas. Det finns fler brister med att använda en S.S ,nämligen samma problem som spidering lider av. En S.S går på samma sätt igenom en webbapplikation som ett spidering verktyg. Vilket betyder att den försöker använda så många funktioner som den kan komma åt.

Mjukvara

De två mest kända S.S på marknaden är AppScan och WebInspect. De kan lära sig hur login funktionaliteten är implementerad genom att de har en inbyggd webbläsare där användaren kan logga in. Det går även att ställa in att den låter bli administratörssidor för att undvika att förstöra webbapplikationen.

En S.S är alltså ett ypperligt sätt att snabbt få en överblick av hur säker en webbapplikation är. Dock måste man som användare av en S.S också inse att inte alla fel upptäcks, därför bör även manuell testning utföras.

Kursinnehåll

I stycket nedan diskuteras vad en webbsäkerhetskurs bör omfatta samt hur webgoat kommer till använding i den kursen.

Enligt en rapport[10] från North Carolina State university(NCSU) bör en kurs i webbsäkerhet omfatta tre viktiga aspekter.

- Kursens innehåll måste hållas relevant.
- Studenterna skall utveckla en säker webbapplikation.
- Studenterna måste få laborera med olika attacker.

För att uppfylla de första kravet krävs det att innehållet i kursen revideras något varje år. Den andra aspekten uppfylls genom att låta studenterna jobba i mindre grupper på ett projekt. I laborationsdelen av kursen kan Webgoat användas som en inledande laboration.

Det finns flertalet andra webbapplikationer med liknande koncept som Webgoat. Exempel på sådana är www.hackthissite.org och www.try2hack.nl. Dem har många utmaningar som är väldigt snarlika dem i Webgoat. Fördelen med att använda Webgoat är att lektionerna är enkla att modifiera och är helt under examinatorns kontroll.Varje lektion är en java-klass vilket gör modifikationer av redan

[10] Teaching a Web Security Course to Practice Information Assurance

befintliga uppgifter en simpel uppgift. Det blir då enklare att undvika fusk i form av att studenterna hämtar färdiga fjolårslösningar från internet.

Laboration

XSS, XPATH och SQL-injection är de tre områden som har blivit utvalda till laborationen. Dessa tre valdes då dem tre är bland de mest kritiska och förekommande säkerhetsluckorna (Hotbilden, sid 8). De tre utvalda områdena har även gemensamt att tillvägagångssättet för angreppet är relativt enkelt. De tilltänkta elever på den fiktiva webbsäkerhetskursen har de förkunskaper som krävs för att kunna ta till sig kurserns innehåll.

Ordlista

XSS	Cross Site Scripting (hette CSS förut men blandades då ihop med Cascading Style Sheets).
XPATH	Path traversal.
DoS	Denial of Service.
Proxy	Proxy-server, är en benämning för en server som agerar mellanhand mellan klienten och målservern.
I.P.	Intercepting Proxy
S.S.	Sårbarhets Scanner
S.E	Social Engineering
MITM	Man in the middle.

Verktyg till Laborationen

Här beskrivs de två olika verktyg som krävs för att klara laborationen.

Webgoat

Webgoat är en plattform utvecklad av OWASP[11] i syfte att lära ut säkerhet i webbapplikationer. Webgoat är uppdelat i lektioner, där varje lektion behandlar en säkerhetslucka. Varje lektion har en teoretisk del och en praktisk del, examensarbetet ämnar att ersätta den teoretiska delen för tre av de redan befintliga lektionerna.

Webgoat är lätt att distirbuera då allt som krävs för att köra Webgoat är en Java runtime och en Tomcat server, båda följer med vid nedladdningen av Webgoat. Lektionerna distrubueras tillsammans med Webgoat i form av java klasser. Det är alltså simpelt för en kursledare att konfigurera Webgoat med en egen uppsättning lektioner som sedan distribueras till kursens elever. Alternativt så sätter examinatorn upp en egen Webgoat server och delar sedan ut konton till varje student.

Webscarab

I den här säkerhetslaborationen skall Webscarab användas i den första delen (XPATH) för att fånga upp och modifiera HTTP trafik till Webgoat. Studenten skall alltså använda Webscarab som en I.P. Det går även att använda sig av någon annan I.P om så önskas.

Laboration

I den här sektionen behandlas laborationen. Den är uppdelad i tre område där varje område är upp delad i introduktion, kunskapskrav, mål, uppgiftskonstruktion och lösningsförslag.

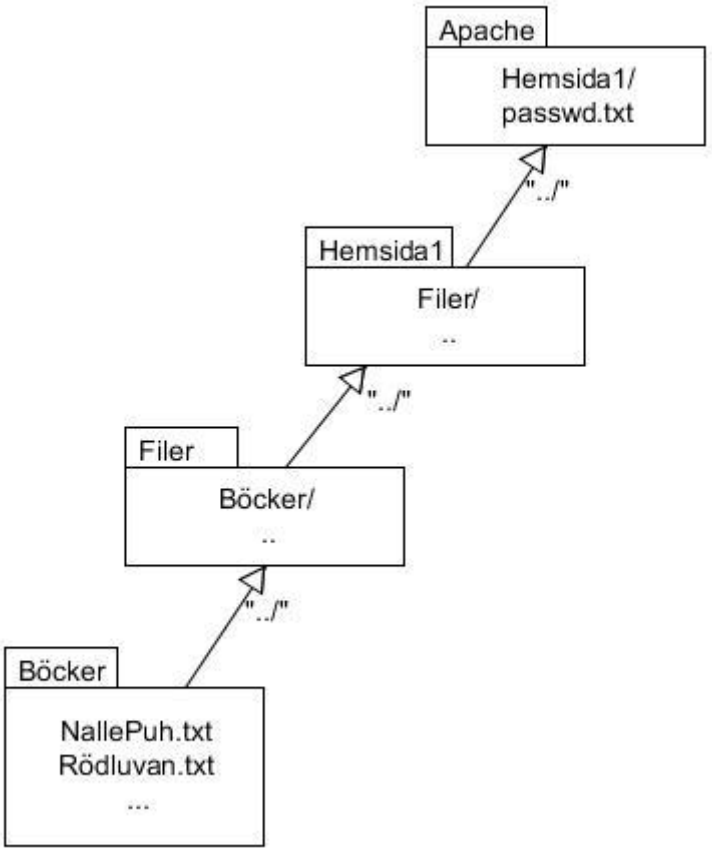
---

[11] http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

Del 1: XPATH

Introduktion

Xpath , även känd som Path traversal, utnyttjar svag åtkomstkontroll. Problemet ligger i att webbapplikationen litar på inmatningen och låter bli att sanitera filnamnet från systemanrop. Om webbapplikationen står i mappen ”apache/hemsida1/filer/böcker/” kan angriparen istället för att mata in namnet på en bok mata in ”../../../passwd.txt”(Se **Error! Reference source not found.**). Det som händer i det här fallet är att filnamnet tolkas av någon komponent för inläsning av filer som då stegar uppåt i filhierarkin tre steg.Vilket betyder att att passwd.txt öppnas istället för en bok. Det som utnyttjas är att komponenten som sköter filinläsningen tolkar varje ”../” [12] som att den måste ta sig upp ett steg i hierarkin.



Figur 3 Visar hur "../" stegar sig uppåt i en filheirarki

Kunskapskrav

-Grundläggande kunskap om unixkommandon.

-Kunskap om hur Webscarab kan användas som I.P.

[12].. (punkt punkt) är ett unixkommando för att stega upp i filhierarkin.

Mål
Att komma åt filer som ligger någon annanstans på servern, t.ex. filen som lagrar användarnamn och lösenord.

Uppgiftskonstruktion
Webgoat besitter en färdig XPATH uppgift som tillåter studenten att utföra det som beskrivs i introduktionen. Den är konstruerad på ett sådant sätt att det verkar som att det inte går att utföra XPATH på den då studenten måste välja en fil i en lista. Studenten måste alltså lära sig att använda verktyget Webscarab(sid 11) för att fånga upp HTTP trafiken. När uppgiften sedan är löst skall studenten presentera ett förslag på hur man kan förhindra detta angrepp. Förslagsdelen är viktig för att få full förståelse för angreppet.

Lösningsförslag
Studenten inspekterar webbapplikationen och finner att filen som skall öppnas går endast att välja i en lista. Studenten skall sedan komma till insikten att de behöver använda sig av Webscarab som en I.P. mellan webbläsaren och webbapplikationen. När studenten sedan ser att variabeln som styr vilken fil som skall väljas skickas i klartext från webbläsaren så byter studenten ut namnet på filen och kan då komma åt lösenordsfilen.

Studenten föreslår sedan en komponent som på serversidan rensar indata från ”../” rekursivt, vilket får anses vara ett godkänt förslag. Ett annat förslag kan vara att komponenten på webbapplikationen kontrollerar att indata motsvarar en fil som ligger i korrekt mapp.

Del2: SQL injection

Introduktion
Den här attacken förekommer oftast där användaren själv får mata in någon form av data som till exempel användarnamn eller sökord. Säkerhetsluckan uppstår då webbapplikationen använder indatan utan att först sanitera den(se nedan).

SQL fråga:
SELECT * FROM users WHERE Username=var1 AND Password=var2;

Innehåll för tabellen ‘users’:

Username	Password	Email
Kamil	Qwerty	khakim@web.se
Admin	Sgh!245Dgl-	admin@web.se

Fall 1(Normal indata):
Indata: var1=(Kamil), var2 = (qwerty)
SQL: SELECT * FROM users WHERE username=’Kamil’ AND password=’qwerty’;
Utdata: Sant

Fall 2(Angrepp):
indata: var1=(foo), var2=(’ or ’1’=1)
SQL: SELECT * FROM users where username=’foo’ AND password=’’ or ’1’=’1’;

Utdata: Sant eftersom ‘1’=’1’ alltid är sant , denna sats skriver även ut hela innehållet i tabellen ’users’.

Då angriparen har tillgång till all information i användardatabasen så kan denne utföra nästa attack, en s.k DoS-attack.Webbapplikationer har en övregräns för hur många användare som kan vara anslutna samtidigt, denna gräns sitter oftast i hårdvaran[13]. När denna gräns är uppnådd slutar webbapplikationen svara. DoS-attacken får förödande konsekvenser för webbapplikationen då attacken utförs genom att angriparen loggar in alla användare som finns i databasen. Vilket snabbt renderar webbapplikationen oanvändbar för samtliga användare.

Kunskapskrav
-Grundläggande kunskap om SQL

Mål
Att först komma åt alla användarnamn och lösenord genom en SQL-injection och sedan utföra en DoS attack mot webb applikationen.

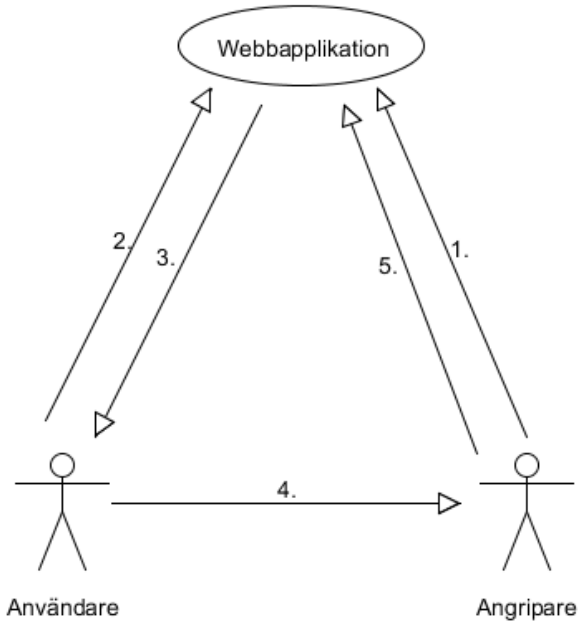
Uppgiftskonstruktion
I Webgoat finns en uppgift som omfattar precis det som beskrivs i introduktionen. Uppgiften behandlar en SQL-injection angrepp som är kombinerad med en DoS-attack. Målet med uppgiften är att utnyttja att webbapplikationen inte saniterar indata. Sedan skall studenten logga in med tre stycken olika användarnamn. Användarnamnen erhålls genom SQL injection attacken. Den andra delen av uppgiften består i att studenten skall presentera ett förslag på hur detta angrepp skall förhindras.

Lösningsförslag
Studenten examinera webbapplikationen genom att skicka ett enkelt citattecken som användarnamn och låter bli att fylla i lösenordsfältet. Detta generar ett felmeddelande som avslöjar att webbapplikationen inte saniterar indata. Studenten sätter nu ihop en attacksträng , ’ or ’1’=’1, och får en utskrift av alla användare och lösenord. Studenten loggar sedan in med 3 av kontona vilket slutför angreppet. Studenten föreslår sedan att detta angrepp kan undvikas genom att använda sig av ett regex som tar bort alla ej tillåtna tecken och ord från indatan.

Del3: XSS

Introduktion
När HTTP protokollet designades hade skaparna i åtanke att det protokollet endast skulle användas för att visa statiska HTML sidor. Vilket betyder att det inte gick att skilja på användarna eftersom det behovet inte fanns, men idag är situationen annorlunda. De flesta statiska hemsidor är utbytta mot dynamiska webbapplikationer där varje användare måste identifieras. Sättet att identifiera en användare idag är via SessionsID som lagras hos klienten i en Cookie. Detta SessionID är högst stöld begärlig då man kan röra sig obehindrat på någon annans konto med det. Det behövs alltså något sätt att stjäla Cookies , det är här XSS kommer in i bilden. Med hjälp av XSS kan angriparen åstadkomma just det. Det finns flerta varianter på XSS attacker som t.ex. Reflected XSS och Stored XSS. Denna säkerhetslaboration omfattar en Stored XSS. En förenklad modell av Stored XSS kan beskrivas i fem steg(se **Error! Reference source not found.**):

[13] Gränsen kan även sitta i mjukvaran i form av en övregräns för hur många anslutningar som får finnas mot databasen samtidigt.



Figur 4 Illustrerar de fem stegen i en stored XSS attack

1. Angriparen lägger in skadlig kod i ett textfält som är synligt för andra användare.
2. En användare observerar textfältet med den skadliga koden.
3. Den skadliga koden laddas ner i användarens webbläsare.
4. Den skadliga koden exekverar, den tar reda på vilken Cookie användaren har och skickar sedan Cookien till angriparen.
5. Angriparen använder sig av Cookien han stal och får tillgång till användarens konto.

Kunskapskrav

-JavaScript

Mål

-Plantera skadlig kod i en profil för användare A och sedan konfirmera att det påverkar användare B.

Uppgiftskonstruktion

Det finns en komplett uppgift om Stored XSS i Webgoat som skall användas för den här delen av laborationen. Målet med uppgiften är att studenten skall logga in som användaren Tom Cat och lägga in skadlig kod i ett fält i Tom Cats profil. Sedan skall studenten logga in som Jerry Mouse och inspektera Tom Cats profil för att verifiera att den skadliga koden Tom Cat har i sin profil också körs i Jerry Mouses webbläsare. Uppgiften i fråga behandlar ej steg 4 och 5 som beskrevs i introduktionen då det gör uppgiften onödigt komplicerad. Studenten skall sedan ge ett förslag på hur det går att undvika säkerhetsluckan.

Lösningsförslag

Studenten loggar in som Tom Cat och klickar på ”Edit profile”. Sedan lägger studenten till <script>alert(’din cookie har blivit stulen!’ + document.cookie);</script> i adressfältet. Studenten loggar sedan ut Tom Cat och loggar sedan in som Jerry Mouse. Som Jerry Mouse väljer studenten först Tom Cat i listan och väljer sedan ”view profile”. Om studenten har gjort rätt bör en ruta komma upp där det står ”din cookie har blivit stulen!” , studenten klickar sedan på ”OK” knappen vilket

slutför laborationen. Studenten föreslår sedan att webbapplikationen integrerar en komponent som rekursivt tar bort <script> taggar. Att komponenten utför det rekursivt är viktigt för annars kan en angripare plantera script taggar i varandra på följande sätt <sc<script>ript>.

Att göra varje lösning unik.

Den här sektionen delas upp i två delar , en som fokuserar på hur en laboration skall göras unik för en årskurs och sen hur de laborationen skall skilja sig från år till år.

Skillnad mellan varje laboration:

Webgoat kräver att studenten loggar in innan denne kan börja laborera. Det första kravet för att förhindra oönskad modifikation(fusk) är att Webgoat är installerad på en server istället för att varje elev laddar ner en egen version från kurshemsidan. Det andra kravet är att varje student har ett eget konto. Förutsatt att båda kraven är uppfyllda kan varje lösning göras unik. Då eleven klarar en del av laborationen får eleven en kod. Koden lämnas in till examinatorn som bevis på att uppgiften är avklarad. Denna kod kan bestå av en permutation av användarnamnet och kurskoden. Examinatorn kan sedan enkelt undersöka hurvida elevens kod är genuin. För att göra det svårare för eleven att försöka finna algoritmen som producerar koden bör den inte visas i klartext. MD5 kan med fördel användas för att dölja koden. MD5 är en kryptografisk hashalgoritm som beräknar en kontrollsumma. Detta används ofta för att kryptera lagrade lösenord på många webbapplikationer, detta erbjuder alltså ett mycket effektivt skydd mot fusk. Detta medför att endast examinatorn kan kontrollräkna en ny MD5 summa för att undersöka om studentens summa är genuin.

Skillnad mellan varje årskurs:

Xpath:

Denna uppgift kan modifieras på följande sätt; filen som är intressant kan flyttas till en annan mapp. Detta medför att eleven inte kan ta en färdig lösning från internet. Fördelen med att flytta på filen är att attackströngen ändras. För att göra ännu en förändring bör man ändra utseendet på uppgiften. Anledningen till att man ändra på utseendet är att man vill ändra angreppspunkten, det går t.ex. att ändra mellan original utseendet(lista) och en sök ruta. Sökrutan kan ha ett skript som saniterar attackströngen på klientsidan så att studenten forceras till att använda en I.P.

SQL injektion:

Uppgiften kan modifieras på ett sådant sätt att den saniterar olika delar av attackströngen. Eleven måste alltså experimentera sig fram för att finna en attacksträng som fungerar. Vilket medför att en attacksträng som fungerade ett år inte fungerar nästa år.

XSS:

Denna uppgift kan göras knepig då examinatorn kan förändra uppgiften på många olika sätt, främst i hur indata saniteras. Det går t.ex. att sanitera bort alla ”<script>” taggar, detta för att forcera eleven att använda en annan teckenkodning på för att dölja dem.

Slutsats

Rapportens frågeställning var huruvida det gick att konstruera en säkerhetslaboration. Laborationen skulle använda sig av Webgoat. Uppgifterna skulle även kunna modifieras för att undvika fusk.

Resultatet pekar på att en sådan laboration går att framställa. För att den skall uppfylla kraven måste vissa förhållanden vara uppfyllda:

- Examinatorn måste vara värd för Webgoat servern.
- Examinatorn måste även se till att uppgifterna modifieras varje år.

Anledningarna till att det bör finnas en Webgoat server istället för att varje elev laddar ned en egen kopia av Webgoat är följande:

- Om eleven laddar ned en egen kopia av Webgoat så får eleven tillgång till källkoden, vilket förenklar fusk.
- Då Webgoat är byggd för att vara osäker så är det lämpligt att den som håller i webbsäkerhetskursen har god insikt i användandet av Webgoat.

Att uppgifterna måste modifieras varje år är ett viktigt krav. Detta måste redan uppfyllas det första året då det finns färdiga lösningar till alla Webgoat uppgifter på internet.

Då webbapplikationssäkerhet är ett väldigt brett område finns det många sätt att utvidga denna rapport. En tänkbar fortsättning är hur en mer avancerad laboration utvecklas. Denna laboration behöver inte använda sig av Webgoat utan kan betraktas som ett projekt. Målet med projektet kan vara att studenterna skall konstruera en säker webbapplikation. En annat tänkbart område inom webbsäkerhet kan vara att skapa en sårbarhetsscanner i form av datoriserad domare. Tanken är att den datoriserade domaren skall fungera som Kattis[14]. Det går även att göra en liknande uppsats som fokuserar mer på de didaktiska aspekterna snarare än de tekniska.

[14] https://kattis.csc.kth.se/about

Referenser

1. **Stuttard, Daffyd and Pinto, Marcus.** *The Web Application Hacker's Handbook, Discovering and Exploiting Security Flaws.* Indianapolis : Wiley Publishing, 2008. p. 736. 978-0-470-17077-9.

2. **Wheeler, David A.** Secure Programming for Linux and unix HOWTO. *http://www.dwheeler.com/secure-programs/.* [Online] Mars 3, 2003. [Cited: April 10, 2010.] http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html.

3. **Gordeychik, Sergey.** Web Application Security Statistics. *The Web Application Security Consortium.* [Online] The Web Application Security Consortium, Februari 1, 2010. [Cited: April 9, 2010.] http://projects.webappsec.org/Web-Application-Security-Statistics.

4. **OWASP.** HTTPOnly. *OWASP.* [Online] Januari 25, 2010. [Cited: April 10, 2010.] http://www.owasp.org/index.php/HTTPOnly.

5. **Wichers, Dave.** OWASP top ten project. *OWASP.* [Online] November 13, 2009. [Cited: April 10, 2010.] http://www.owasp.org/index.php/OWASP_Top_Ten_Project.

6. **OWASP.** Webgoat. *OWASP.* [Online] Mars 9, 2010. [Cited: April 10, 2010.] http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project.

7. **Yu, H, et al.** *Teaching a Web Security Course to Practice Information Assurance.* Department of Computer Science, North Carolina A&T State University. New York : ACM, 2006. 0097-8418.

8. **Goyvaerts, Jan.** Regular Expression. *Regular Expression.* [Online] Mars 19, 2010. [Cited: April 17, 2010.] http://www.regular-expressions.info/.

9. **KTH.** Om Kattis. *KTH Automated Teaching Tool.* [Online] Mars 24, 2009. [Cited: April 22, 2010.] https://kattis.csc.kth.se/about.

10. *Can you find the XSS vurnability ?* [Online] [Cited: April 28, 2010.] En enkel webbapplikation i form av ett mycket enkelt Wargame.. http://xss.progphp.com/xss1.html.