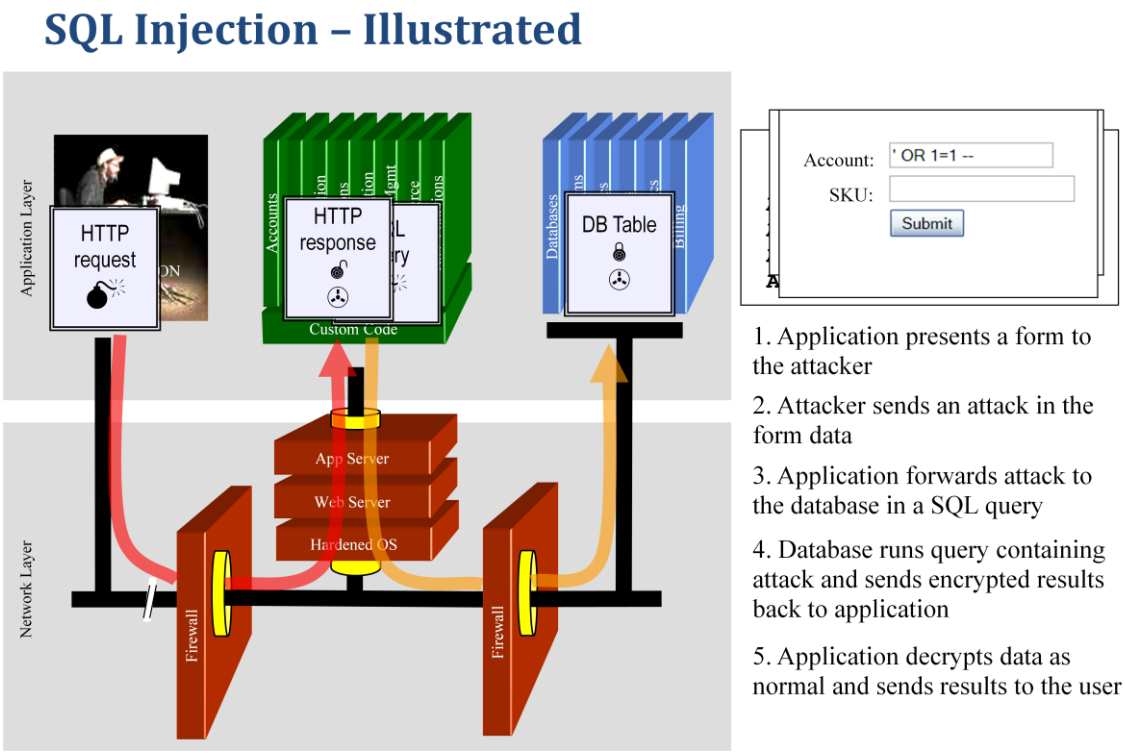
Bezpečnosť vo web aplikáciách

Zoznam bezpečnostných rizík pre web aplikácie (vychádza hlavne z výsledkov a materiálov OWASP).

Obsah

1 INJECTION....2
2 Cross Site Scripting (XSS) 4
3 Broken Authentication and Session Management 7
4 Insecure Direct Object References 10
5 Cross Site Request Forgery (CSRF) 11
6 Security Misconfiguration 13
7 Failure to Restrict URL Access 14
8 Unvalidated Redirects and Forwards 15
9 Insecure Cryptographic Storage....17
10 Insufficient Transport Layer Protection 18
11 Ďalší prehľad nástrojov na testovanie 19

1 INJECTION



SQL injection je hackerská metóda ktorá využíva nedostatky (chyby programátora) v kontrole vstupných dát k tomu aby mohol hacker podstrčiť namiesto očakávaných informácií vlastné a prebrať tak kontrolu nad databázou nad rámec svojich práv. Tým prebraním kontroly je myslená úplná kontrola nad databázou kde hacker môže databázu kopírovať, mazať, pridávať ľubovoľné údaje alebo kradnúť dôležité informácie.

Takýto útok je teda veľmi závažný a je potrebné sa pred ním chrániť. Základom je myslieť na tento typ útokov už pri implementácii. Je dôležité ošetrovať a kontrolovať všetky vstupy od používateľov, ale napr. aj adresu, ktorá sa pri danej stránke zobrazuje v prehliadači (napr. http://www.mojastranka.tld/clanok.php?id=1 **or 1=1**). Články ako sa brániť pred takýmto typom útokov sú napr. :

http://security-portal.cz/clanky/sql-injection

http://msdn.microsoft.com/sk-sk/magazine/cc163917(en-us).aspx

http://interval.cz/clanky/bezpecnost-predevsim-bezpecnejsi-prikazy-sql/

http://www.mikesdotnetting.com/Article/113/Preventing-SQL-Injection-in-ASP.NET

Metódy testovania

Existuje viacero prístupov ako testovať tento útok. Príručka od OWASP prináša podrobný postup pre testovanie (kapitola 4.8.5 – OWASP Testing Guide).

V skratke je tu stále možnosť testovať aplikáciu manuálne, ale existuje už aj mnoho nástrojov, ktoré môžu takéto testovanie zjednodušiť. Otestovať by sa mali všetky vstupy od používateľa rôznymi dátami. Možno ešte lepší prístup nie je testovať výslednú aplikáciu, ale priamo analyzovať zdrojový kód.

Free:

- **http://www.fyxm.net/Source-Code-9704.html** - prechádza ASP kód
- http://www.fyxm.net/SQLFury-29581.html
- http://www.fyxm.net/SQLBrute-70312.html
- **http://www.microsoft.com/downloads/details.aspx?FamilyId=58A7C46E-A599-4FCB-9AB4-A4334146B6BA&displaylang=en** - toto neviem či je free
- **http://sqlmap.sourceforge.net/**
- http://weblogs.asp.net/hosamkamel/archive/2008/06/30/free-sql-injection-vulnerability-scanner-by-hp.aspx
- http://www.sqlpowerinjector.com/download.htm

Licencované:

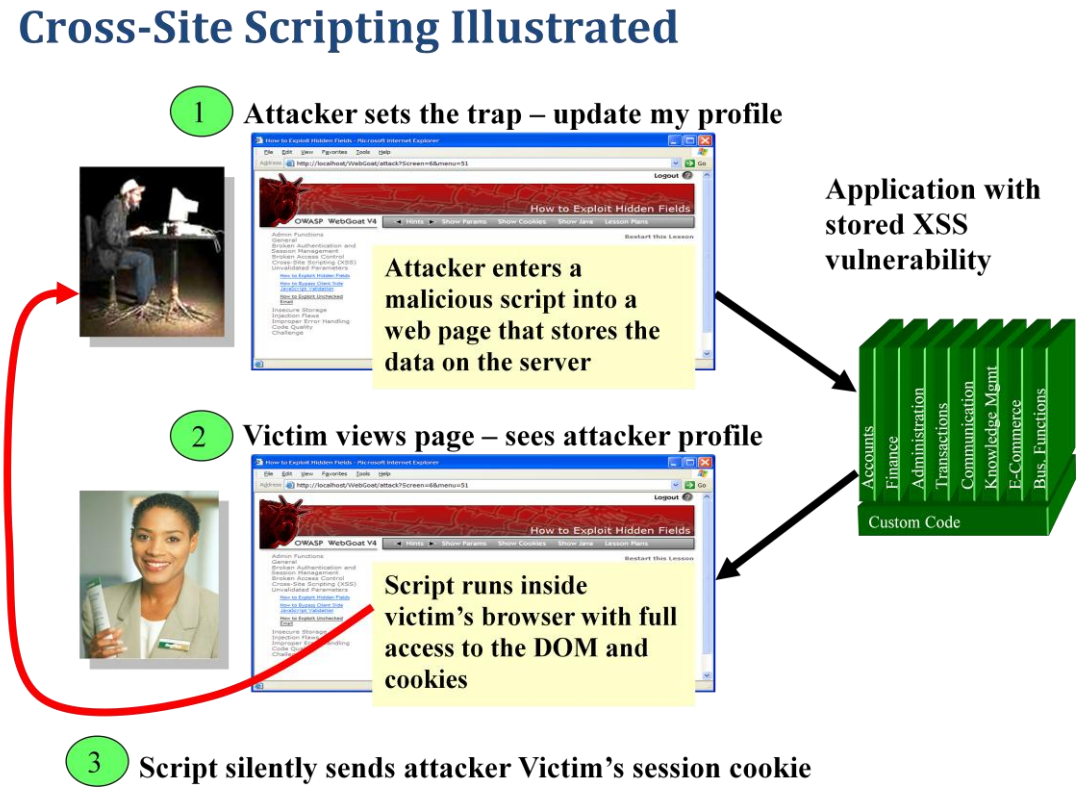
- **http://www.acunetix.com/vulnerability-scanner/sql-injection-scanner.htm?gclid=CJ3VsOGv5Z8CFVSJzAodyi2igQ**
- http://nstalker.com/products
- http://www.fyxm.net/Maui-Security-42417.html
- http://www.fyxm.net/SecurityToolset-80356.html
- **https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-201-200%5E9570_4000_100__**

Okrem týchto produktov som našiel zaujímavú stránku, ktorá mapuje ďalšie nástroje súvisiace s hľadaním sql injections: **http://maestro-sec.com/blogs/2008/10/top-15-sql-injection-scanner/.**

SQL injection je medzi útokmi typu injection asi najpoužívanejší a najznámejší. Existujú ale i ďalšie alternatívy: Blind SQL injection, **Script injection,** LDAP injection, XML Injection...

2 Cross Site Scripting (XSS)

Princíp:



(nasledovné vysvetlenie XSS útoku je prebraté z http://blog.synopsi.com/2007-12-12/najpopularnejsie-utoky-xss-a-csrf-na-vyslni)

XSS alebo Cross-site scripting je typ útoku, kedy sa za použitia client-side skriptov (JavaScript) pozmení časť kódu zobrazovanej webstránky tak, aby bol nebezpečný skript útočníka spustiteľný a vykonal určitú akciu, napríklad získal a odoslal dáta z cookies obete. Útočník tak po nahratí dát z cookies ktoré získal od obete, často získa prístup do aplikácie (webstránky) bez toho aby poznal vaše prihlasovacie údaje!

Existujú tri typy zraniteľností pre XSS. Ich pomenovanie sa líši od dokumentu k dokumentu a tak použijem typové označenie z Wikipedie a teda **Typ 0, 1 a 2** s kodovými menami pre **0 Local** (Lokálny), pre **1 Non-persistent** (nestály) a pre **2 Persistent** (stály).

Typ 0 Local (Lokálny)

Toto je najzákladnejšia forma XSS a taktiež sa označuje ako DOM-based (založené na objektovom modele dokumentu). Pri tejto forme je kód spustený na strane klienta (client-side), kedy sa využíva časť JavaScript kódu namiesto parametra v URL a práve za pomoci JavaScriptu je stránka zmenená (infikovaná) novým kódom (stále na strane klienta), ktorý sa vykoná okamžite po inicializácii browserom obete.

Typ 1 nestály (non-persistant)

Táto forma XSS je asi najmenej nebezpečná a to z dôvodu, že útočník spolu s nebezpečným kódom musí použiť aj tzv. social engineering (sociálne inžinierstvo).

Princíp spočíva v tom, že sa využíva dynamicky generovaný obsah na základe užívateľskej aktivity (AJAX, AJAJ atď.) spolu s predpokladom, že stránka neošetruje dáta premenouHTML kódu na entity, ale umožňuje priame zadanie kódu, ktoré je pri vygenerovaní interpretované a vykonané užívateľovým prehliadačom a takto umožní útočníkovi prístup k citlivými dátam ako sú napríklad Cookies alebo Sessions.

Typ 2 stály (persistant)

Táto forma umožňuje najsilnejší typ útoku (infikuje najväčšie množstvo obetí). Taktiež sa označuje ako HTML injection (infikovanie HTML). Pre tento typ útoku sa využíva uschovanie nebezpečného kódu na strane serveru napríklad v databáze, pričom vstup ani výstup nie je ošetrovaný za pomoci premeny HTML kódu na entity. Najbežnejším príkladom sú tzv. guestbooky, kedy nie je vstup ani výstup chránený proti HTML injection a tak sa nebezpečný kód automaticky interpretuje pri vygenerovaní stránky až do odstránenia majiteľom. Táto metóda sa považuje za najnebezpečnejšiu, pretože sa odhaduje pri jej použití najväčšie množstvo obetí, pričom nebezpečný kód stačí použiť len raz. Aj pri tejto metóde je nutné použiť malé množstvo social engineeringu (vytvorenie zaujímavého odkazu, ktorý pritiahne veľké množstvo ľudí a donúti ich linku otvoriť).

Ďalšou výhodou útoku je, že útočník pre zber dát nepotrebuje vlastnú stránku, stačí mu využiť napríklad logovanie do súboru na infikovanom serveri, email alebo inú formu pre dočasné uloženie výsledkov.

Scénare útoku

Ako by mohli vyzerať útoky za pomoci XSS?

Typ 0

Útočník nájde XSS dieru a pošle upravenú URL linku s nebezpečným kódom Martine (za pomoci emailu, IM alebo inou formou komunikácie)

Martina otvorí linku

Infikovaná web stránka otvorí a načíta za pomoci JavaScriptu útočníkovú stránku so zlým kódom

Stránka spustí tento kód a nebezpečný skript získa práva Martiny

Nebezpečný kód teraz môže spúšťať príkazy s právami Martiny na jej vlastnom PC

Typ 1

Martina navštívi Janovu stránku ktorá jej umožní prihlásiť sa pod menom a heslom a vyplniť osobné údaje

Útočník zistí, že Janová stránka je infikovateľná za pomoci XSS

Útočník aplikuje nebezpečný kód a na Martinin email pošle správu, ktorá vyzerá ako keby prišla z Petrovej stránky (email je spoofnutý)

Martina otvorí linku ktorá bola uvedená v emaili

Linka infikovaná nebezpečným kódom sa z inicializuje a ukradne citlivé dáta (cookies, informácie o kreditných kartách, atď)

Typ 2

Peter má stránku ktorá umožňuje ostatným pridávať textové odkazy (guestbook) a tie následne zobrazuje

Útočník zistí, že Petrova stránka je napadnuteľná XSS typu 2

Útočník pridá správu so zaujímavým obsahom a nebezpečným kódom na ktorú by mohol nalákať najviac obetí (veľmi bežne sa uvádzajú linky na “bezplatné porno”)

V prípade že užívatelia kliknú na tento odkaz, nebezpečný kód začne zbierať citlivé dáta bez vedomia obetí

Útočník si prezie svoje logy ukradnutých dát a využije ich.

Proti tomuto útoku sa môžu bežní používatelia brániť zakázaním scriptov vo svojich prehliadačov. To však môže ale spôsobiť nesprávne zobrazovanie niektorých stránok. Vývojári by mali kontrolovať všetky vstupy od používateľov, aby nebolo možné do týchto vstupov zadať kúsky kódu s nejakým skriptom.

Existuje viacero modulov/tried ktoré by mali zabezpečovať aplikácie pred takýmito útokmi:

http://debug.sk/ochrana-proti-cross-site-scripting-a-cross-site-request-forgery

http://www.aspnet.sk/News.aspx?Id=101105

https://www.openeco.org/XSS

http://msdn.microsoft.com/en-us/library/ms972967.aspx

Metódy testovania:

V tomto prípade je potrebné testovať, či je možné do vstupov zadávať rôzny škodlivý kód a ako naňho aplikácia reaguje. Aplikácia by rozhodne nemala napr. povoliť zobrazenie obsahu vloženého používateľom bez kontroly a validácie. Podrobnejší popis metód je v OWASP Testing Guide kapitoly 4.8.1,4.8.2,4.8.3,4.8.4.

Softvér pre testovanie xss:

Free:

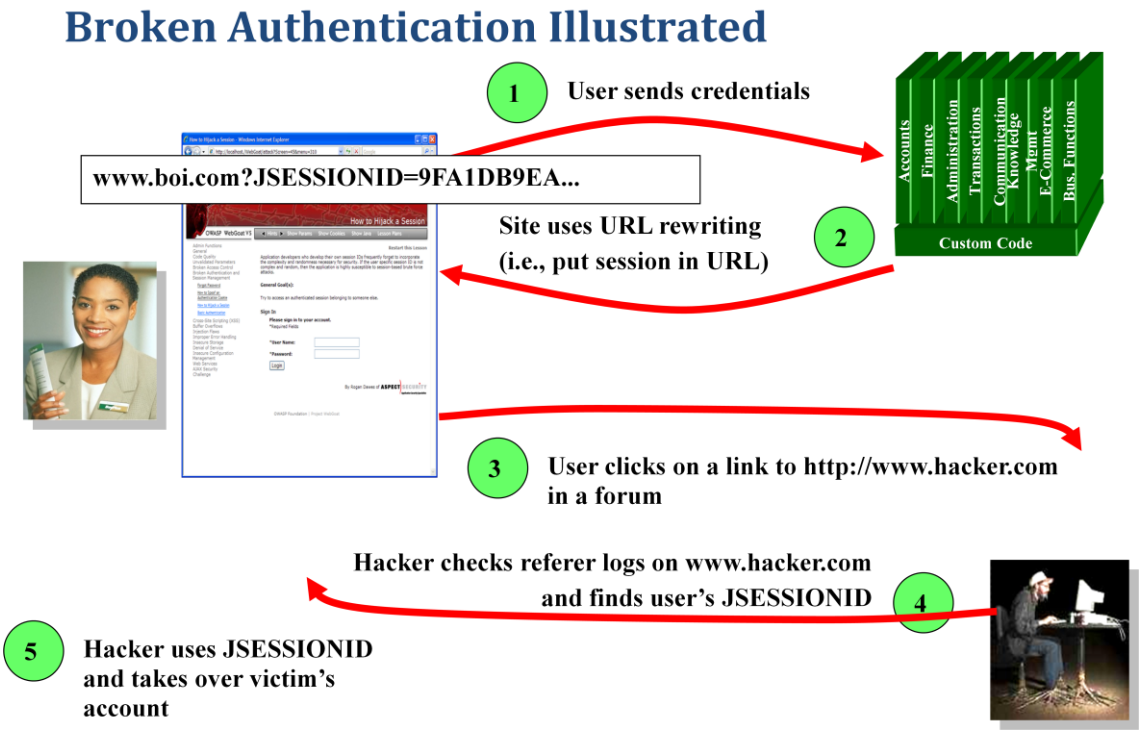
http://www.acunetix.com/cross-site-scripting/scanner.htm - je free iba pre testovanie xss, pre iné typy je licencovaný.

https://addons.mozilla.org/en-US/firefox/addon/7598

http://www.thespanner.co.uk/2009/03/25/xss-rays/

Licencovaný: - nenašiel som nič vhodné.

3 Broken Authentication and Session Management



Vysvetlenie tohto typu útokov je prebraté z prednášky Bezpečnostné zraniteľnosti v nových webových aplikáciách (Pavol Lupták) http://h-k.osiris.ynet.sk/uploads/howknow.2009.06.24.Zranitelnost.v.novych.webovych.technologiach.slides.pdf

Z bezpečnostného hľadiska chybne implementovaný „session management" je hlavný kameň úrazu väčšiny súčasných webových aplikácií. Ako majú aplikácie korektne implementované ošetrovanie vstupu, tak zle majú implementovaný „session anagement".

Zraniteľnosti na „session fixation" útoky

Veľa webových aplikácií kvôli rôznym dôvodom (napríklad nespokojnosť s rýchlosťou „session managementu" poskytovaným samotnými knižnicami daného jazyka) používa vlastný „session management", ktorý je takmer vždy zle naimplementovaný a porušuje niektorú z bezpečnostných zásad:

- útočník dokáže **nainjektovať ľubovoľnú hodnotu** do anonymného „session ID"
- aplikácia uvedenú hodnotu **akceptuje!**
- po úspešnej autentizácii používateľa túto hodnotu **nepregeneruje!**
- po odhlásení používateľa túto hodnotu na strane klienta alebo servera **neinvaliduje** (a následne použije pre ďalšieho používateľa, ktorý sa z daného prehliadača prihlási)

Akceptovanie nainjektovanej hodnoty „session ID" útočníkom znamená, že aplikácia

je zraniteľná na **„Session fixation“ útok**, ktorý je bohužiaľ dosť bežný aj v bankových aplikáciach. „Session fixation“ zraniteľnosť nastáva v situáciách, keď aplikácia vytvára anonymnú „session“ pre každého používateľa, ktorý prvýkrát pristupuje k aplikácii ešte predtým ako sa úspešne autentizuje. Po prihlásení používateľa, funkcia *login* v danej aplikácii len aktualizuje danú anonymnú „session“ na strane servera na autentizovanú. Ten istý „session ID“, ktorý sa používal na anonymné prehliadanie stránky, sa zrazu používa na privilegovaný prístup v kontexte daného autentizovaného používateľa. Aby bolo možné realizovať štandardné ukradnutie „session“ (t.j. realizovať „hijacking“ útok), útočník musí použiť nejaký spôsob na odchytenie „session ID“ daného používateľa (napríklad nájsť XSS zraniteľnosť). V prípade „session fixation“ útoku je to ale podstatne jednoduchšie – útočník najskôr obdrží anonymný „session ID“ priamo z aplikácie a použije nejaký spôsob na podvrhnutie daného tokenu do webového prehliadača svojej obete. Hneď po tom, ako sa používateľ úspešne do aplikácie prihlási, útočník použije preňho už „známy“ token na ukradnutie používateľovho autentizovaného spojenia. Kľúčová fáza v tomto útoku je samozrejme bod, kedy útočník „presvedčí“ obeť o použití „session ID“, ktorý on predtým získal, resp. je mu dopredu známy. Existuje viacero techník (ktoré závisia od mechanizmu, ktorý používa aplikácia na prenos „session ID“) pomocou ktorých útočník dokáže podvrhnúť špecifický „session ID“ svojej obeti.

Absencia „secure“ a „httponly“ atribútu

Veľa testovaných aplikácii stále nepoužívalo pre „cookies“ atribút „secure“. V praxi to znamená, že „cookies“ platné pre https spojenie dokáže klient poslať aj v nešifrovanom http spojení. Takmer žiadna z testovaných aplikácií nepoužívala pre „cookies“ atribút „httponly“, čo v praxi znamená, že injektovaný javascript (cez XSS zraniteľnosť) dokáže všetkým používateľom načítať a ukradnúť „cookies“. Je dôležité zdôrazniť, že „httponly“ atribút je neúčinný pokým cieľový web server podporuje HTTP metódu TRACE (štandardne zapnutá v Apache web serveri), nakoľko útočník (napríklad cez AJAX) dokáže sformulovať TRACE žiadosť a v jej odpovedi získať validné „cookies“ ku ktorým by sa pri zapnutom atribúte „httponly“ cez „*document.cookie*“ z javascriptu nedostal.

Podpora simultánnych spojení

Podpora simultánnych spojení (t.j. možnosť viacnásobne sa prihlásiť pod jedným používateľom do aplikácie z jednej alebo viacerých odlišných IP adries) bol **bezpečnostný problém väčšiny testovaných aplikácií.** Takmer v každej bežnej aplikácii obvykle neexistuje dôvod, prečo by jeden užívateľ mal mať viac ako jedno aktívne prihlásenie v danom čase. Samozrejme, že je celkom bežné, že užívateľ ukončí svoje doterajšie aktívne prihlásenie tým, že sa opätovne prihlási – napríklad, že zatvorí okno svojho prehliadača alebo sa presunie na iný počítač. Ak je ale používateľ simultánne prihlásený viackrát (z rôznych adries), obvykle to znamená, že nastalo bezpečnostné kompromitovanie používateľa: buď používateľ zverejnil svoje prihlasovacie údaje ďalšej strane alebo útočník získal dané prihlasovacie údaje iným spôsobom (napríklad využitím XSS zraniteľnosti). V oboch prípadoch je povolenie simultánnych prihlásení nežiadúce, nakoľko umožňuje používateľom používať z bezpečnostného hľadiska nežiadúce praktiky a útočníkovi umožňuje využívať používateľské prihlasovacie údaje bez rizika jeho odhalenia samotným kompromitovaným užívateľom. Ako riešenie odporúčame používateľské prihlásenie („session“) zviazať buď s jeho IP adresou alebo IP rozsahom (v prípade, že používateľ používa proxy server s viacerými verejnými IP adresami).

Možnosť útoku hrubou silou na „session management“

Všetky testované webové aplikácie boli zraniteľné na **útoky hrubou silou na „session management**“.

Ak „session ID“ využívané aplikáciou je kratšie ako 128 bitov alebo je ľahko determinovateľné (automatizovane je to možné zistiť buď pomocou analýzy „session ID“ vo WebScarabe alebo pomocou Stompy), hrozí riziko útokov hrubou silou na „session management“. Útočník sa môže pre tento druh útoku rozhodnúť, keď bitová zložitosť prideleného „session ID“ je podstatne menšia, ako zložitosť používaných hesiel. Tento útok je podstatne ťažšie zastaviteľný ako útok hrubou silou na prihlasovacie mená a hesla, nakoľko cieľová aplikácia nemá hrubou silou skúšané HTTP žiadosti nijako asociované s používateľom, ktorého by mohla jednoducho zablokovať. Nakoľko je na aplikáciu vyskúšané v krátkom čase veľké množstvo http žiadostí s rôznym „session ID“, aplikácia to dokáže detekovať a v prípade, že uvedený útok nastane, tak zdrojovú IP adresu útočníka zablokovať na definovanú dobu (ktorej dĺžka môže závisieť od agresivity útoku). Bohužiaľ žiadna z našich testovaných webových aplikácií nebola chránená voči tomuto druhu útoku.

Metódy testovania

Dôležité v tomto prípade je skontrolovať celý mechanizmus zabezpečenia a riadenia pripojenia používateľa. Či sú použité štandardné metódy pre prácu so session, či sú správne nastavené potrebné parametre spojenia... Podrobné postupy pre testovanie sú v OWASP Testing Guide kapitoly 4.5.1 až 4.5.4.

Softvér:

Pre tento typ útokov som nenašiel softvér, ktorý by priamo otestoval aplikáciu a overil jej zabezpečenie. Našiel som ale niekoľko podporných nástrojov, s ktorými je toto testovanie efektívnejšie:

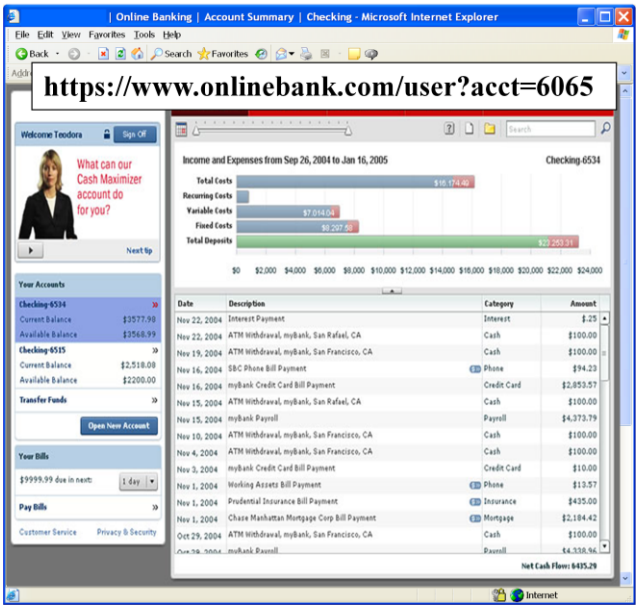
http://www.acunetix.com/blog/acunetix-how-to/tutorial-on-how-to-test-for-broken-authentication-using-acunetix-wvs-tools/

http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

http://www.securiteam.com/tools/5VP0O2AKAG.html

4 Insecure Direct Object References

Insecure Direct Object References Illustrated



- Attacker notices his acct parameter is 6065

  ?acct=6065

- He modifies it to a nearby number

  ?acct=6066

- Attacker views the victim's account information

Základom tohto útoku je zobrazenie interne implementovaných objektov používateľom. Príklad takéhoto zobrazenia je uvedený na ilustrácii hore. Útočník môže modifikovať tieto objekty a získať tak prístup k nim. Dva hlavné typy tohto útoku sú:

Open Redirect – v tomto prípade aplikácia využíva parameter, pomocou ktorého presmeruje používateľa na konkrétnu stránku bez akejkoľvek kontroly prístupu. Táto metóda sa využíva hlavne pri phisingových útokoch nato, aby útočník presmeroval používateľa na svoju škodlivú stránku.

Directory Traversal – v tomto prípade aplikácia umožňuje priamy prístup k súborom na lokálnom počítači/ serveri. Ak aplikácia neoveruje, kto a akým spôsobom k súborom pristupuje, útočník môže získať prístup k interným súborom. Napr. ak máme stránku http://misc-security.com/file.jsp?file=report.txt, útočník môže zmeniť jej parameter na http://misc-security.com/file.jsp?file=../../../etc/shadow, čím získa prístup k heslám do aplikácie.

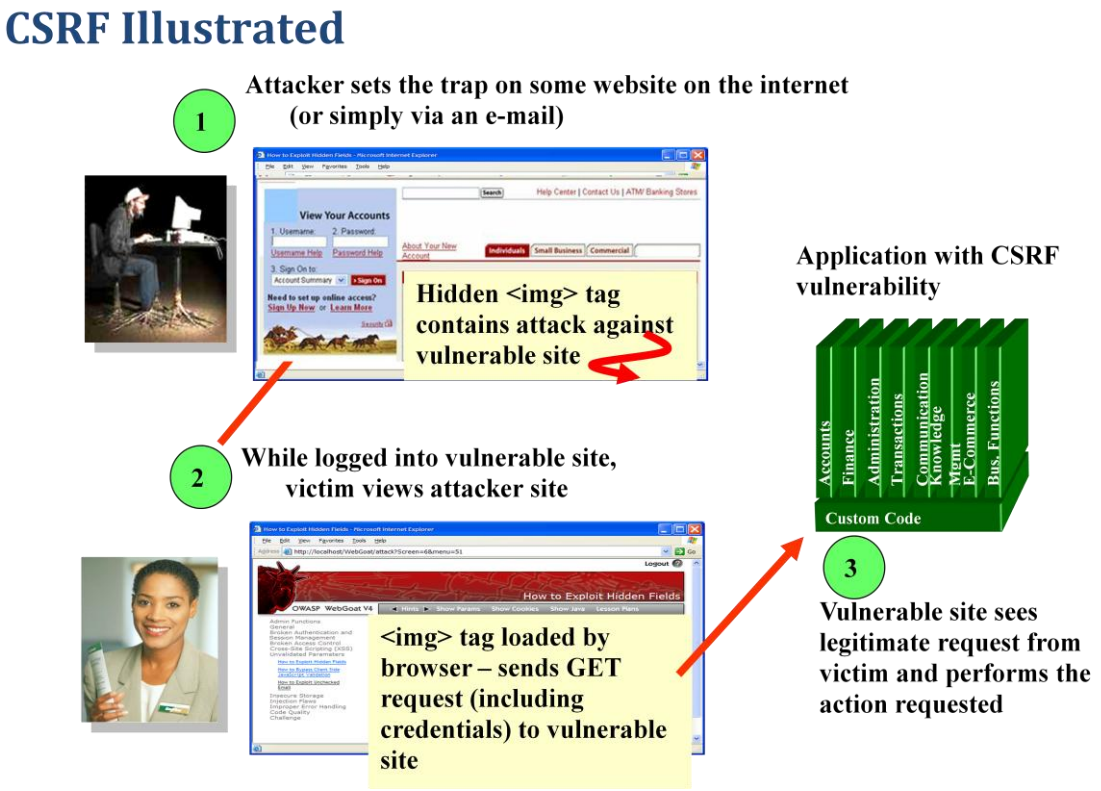
Ako ochrana pred takýmito útokmi slúži nezobrazovanie interných parametrov používateľom, mapovanie takýchto parametrov na virtuálne hodnoty –

a kontrolovanie prístupu k objektom na základe prihláseného používateľa.

Metódy testovania

Pre tento typ útokov som nenašiel žiadne automatizované nástroje. Testovanie teda spočíva v manuálnom prezretí kódu a spôsobu, ako aplikácia narába s internými parametrami.

5 Cross Site Request Forgery (CSRF)



(vysvetlenie prebraté z http://blog.synopsi.com/2007-12-12/najpopularnejsie-utoky-xss-a-csrf-na-vyslni)

CSRF/XSRF alebo Cross-site request forgery je veľmi podobný predošlému XSS útoku. U CSRF útoku je známa jedna vec. Pre predanie argumentov stránke obeti využíva metódu GET, čo znamená že parametre sú posielané priamo v url. Tento typ útoku v postate nie je chybou ani prehliadača ani užívateľa a je podstatou samotného protokolu. HTTP protokol je veľmi jednoduchý, server na jeho základe nedokáže identifikovať návštevníka, na to slúžia Cookies. Metóda POST je rovnako postihnutá, je síce zložitejšia ale nie nemožná, to treba mať na zreteli! Celý problém spočíva v neoveriteľnej transakcii, ktorú browser vykoná. Server nie je schopný identifikovať užívateľa na základe HTTP protokolu a preto jeho jediná možnosť je použiť Cookies. Keďže však obeť Cookies pošle a zhodujú sa z uloženými dátami o užívateľovi, je takto možné vykonať prakticky akúkoľvek zmenu. Najjednoduchšie to bude zobraziť na príklade.

Máme stránku A, ktorá na základe parametru v URL odhlási užívateľa.

GET http://gmail.com/?logout=true HTTP/1.1

Gmail však nevie overiť od koho táto požiadavka prišla na základe HTTP protokolu a tak to urobí na základe Cookies ktoré má uložené užívateľ v prehliadači. To znamená, že v prípade že užívateľ takejto stránky navštívi práve tú kde bude nebezpečný kód, dojde k odhláseniu. Nič úžasné, maximálne otravné. Čo však ak na tomto základe útočník zmení identifikačné dáta obete?

CSRF je určite veľmi nebezpečný typ útoku, je veľmi ťažké sa proti nemu brániť. Posledne sa ako najlepšia ochrana osvedčili tokeny inicializované pri prvom načítaní z

login page a ďalej sa s nimi už pracuje. Po vypršaní session je nutné reinicializovať token a tak sa zabráni použitiu tohoto typu útoku. Predstavte si iný prípad.

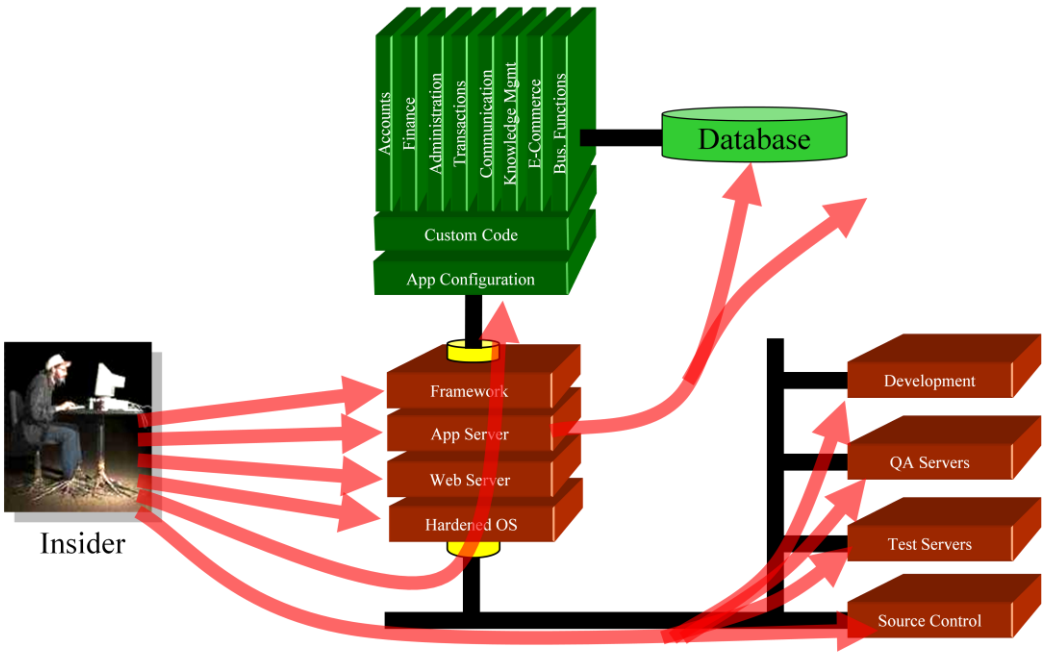
Užívateľ príde na stránku kde je CSRF škodlivý kód a ten mu podstrčí dva, tri, štyri obchody odkiaľ by mohol niečo kúpiť. Ak má útočník šťastie, užívateľ už v jednom z nich prihlásený je a ak nebodaj v obchode, ako Amazon, ich fantastická funkcia o ktorej patent sa snažili “1-Click ordering” odošle objednávku na pozmenenú adresu a to práve útočníka. Škoda môže byť obrovská a nenapraviteľná

Metódy testovania

Podrobný návod je v OWASP Testing Guide kapitola 4.5.5. Daný útok je potrebné opäť zrejme testovať len manuálne. Okrem nasledovného projektu som nenašiel žiadne nástroje, ktoré by testovali takýto útok automatizovane.

http://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project

6 Security Misconfiguration

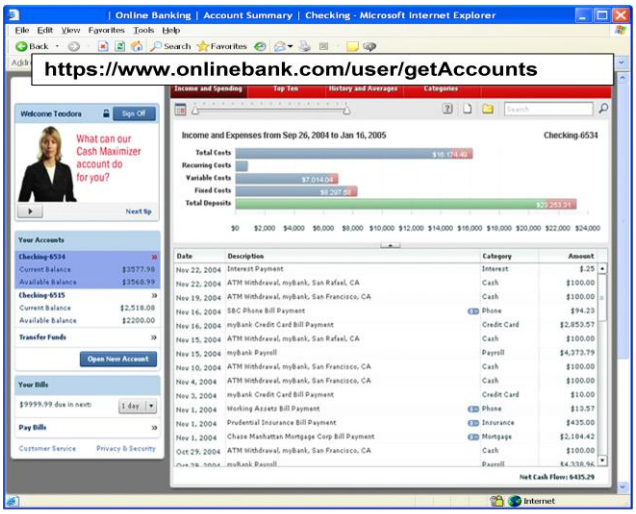


K tejto kapitole som našiel len minimum zdrojov. V princípe ide o celkovú konfiguráciu a zabezpečenie prostredia, kde sa aplikácia nachádza. Útočník teda môže využiť chyby v sieti, na serveri alebo chyby operačného systému, kde je aplikácia nasadená. Takýmto spôsobom môže prekonať aj dobre zabezpečenú aplikáciu.

Proti takýmto útokom treba podrobiť bezpečnostnému testovaniu celé prostredie, kde bude aplikácia nasadená. Skontrolovať pravidelnú aktualizáciu prvkov systému, overiť bezpečnostný manažment prostredia...

7 Failure to Restrict URL Access

Failure to Restrict URL Access Illustrated

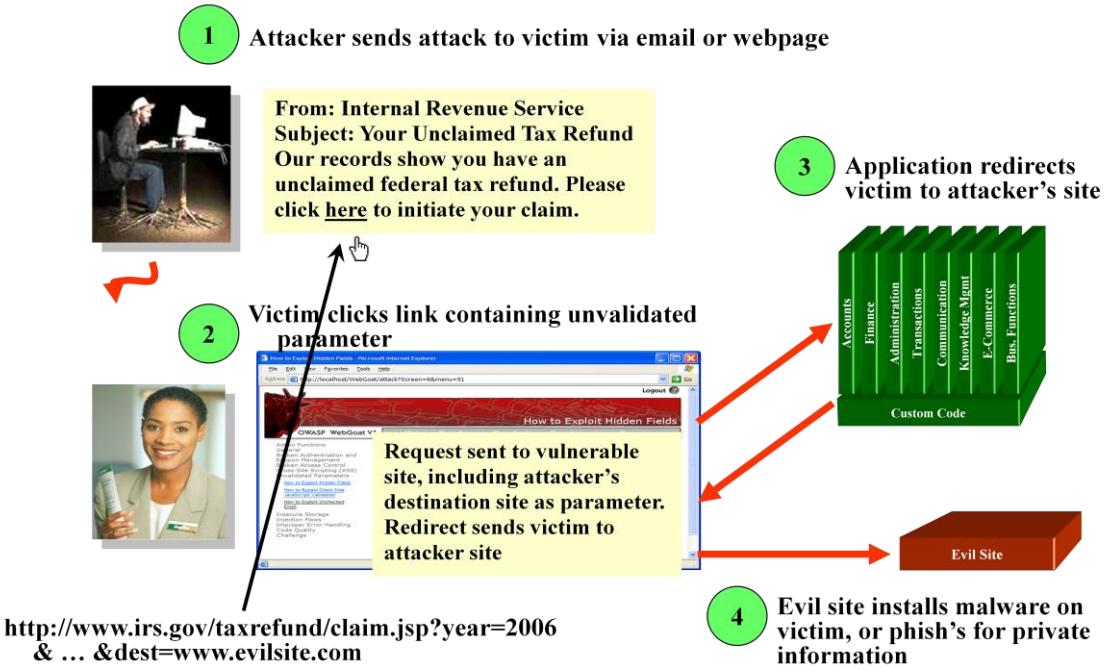


- Attacker notices the URL indicates his role
  /user/getAccounts
- He modifies it to another directory (role)
  /admin/getAccounts, or
  /manager/getAccounts
- Attacker views more accounts than just their own

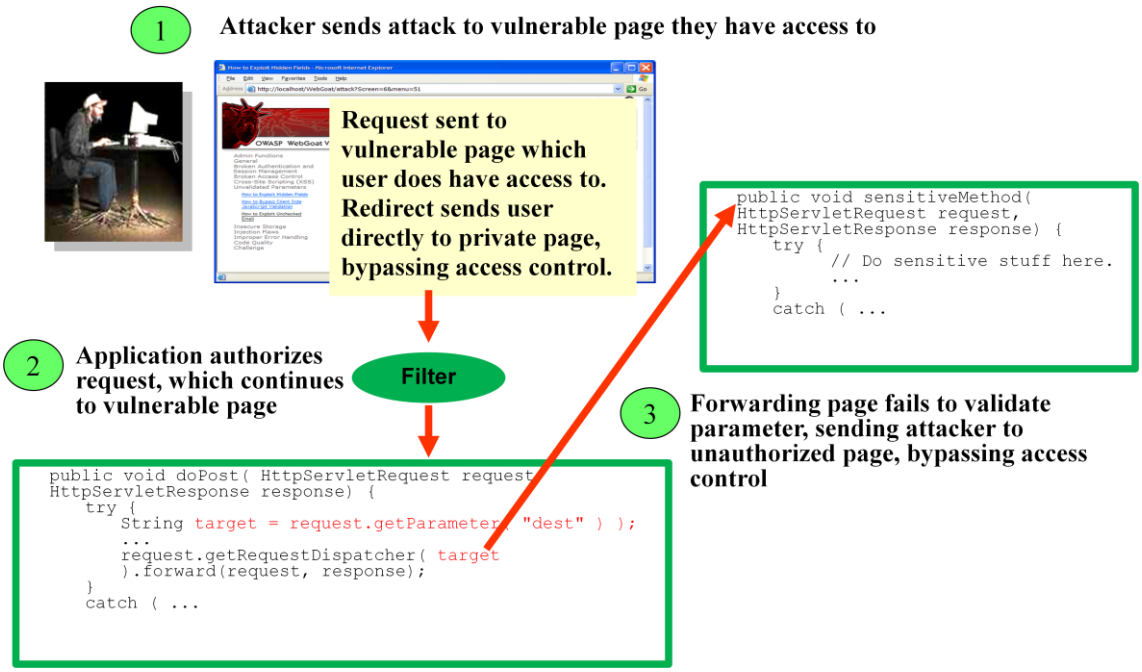
Tento útok spočíva v neoverovaní prístupu užívateľom k stránkam, ktorých adresu zadá útočník priamo do prehliadača. Takto môže útočník získať napr. administrátorský prístup k stránkam alebo prístup k bežne nedostupným stránkam. Ochrana proti takýmto útokom spočíva v overovaní autorizácie pri prístupe ku konkrétnej stránke. Je vhodné mať maticu zobrazujúcu, aká rola má prístup k akým stránkam (príp. aj s konkrétnymi právami). Tento typ útokov možno zrejme tiež testovať len manuálne.

8 Unvalidated Redirects and Forwards

Unvalidated Redirect Illustrated



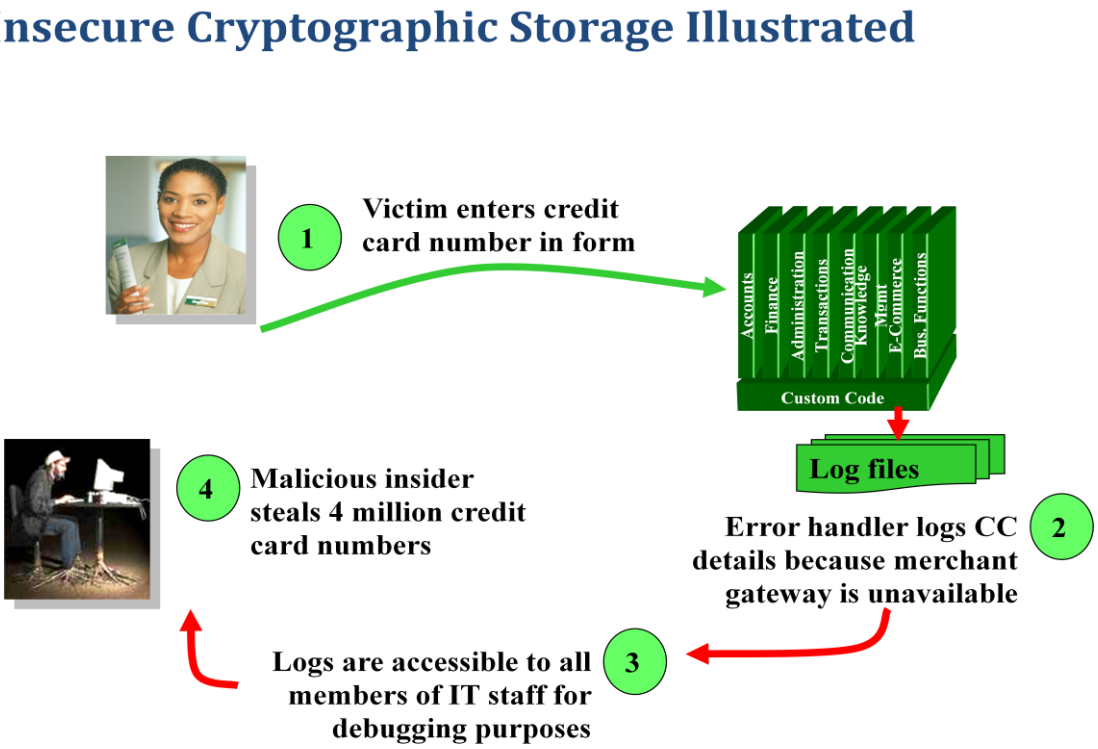
Unvalidated Forward Illustrated



Pri týchto útokoch využíva útočník odkazy a presmerovania na externé stránky, pričom tieto odkazy môže modifikovať tak, aby sa používateľ dostal na útočníkovu stránku, kde môže zanechať dôverné údaje (napr. prihlasovacie). Je preto vhodné vyhýbať sa

takýmto presmerovaniam, nepoužívať v nich parametre, a hlavne kontrolovať nielen platnosť parametrov ale i cieľ presmerovania. Možno využiť nasledovné API (ESAPI http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API - SecurityWrapperResponse.sendRedirect(URL)). Tieto útoky zrejme možno testovať len manuálne.

9 Insecure Cryptographic Storage

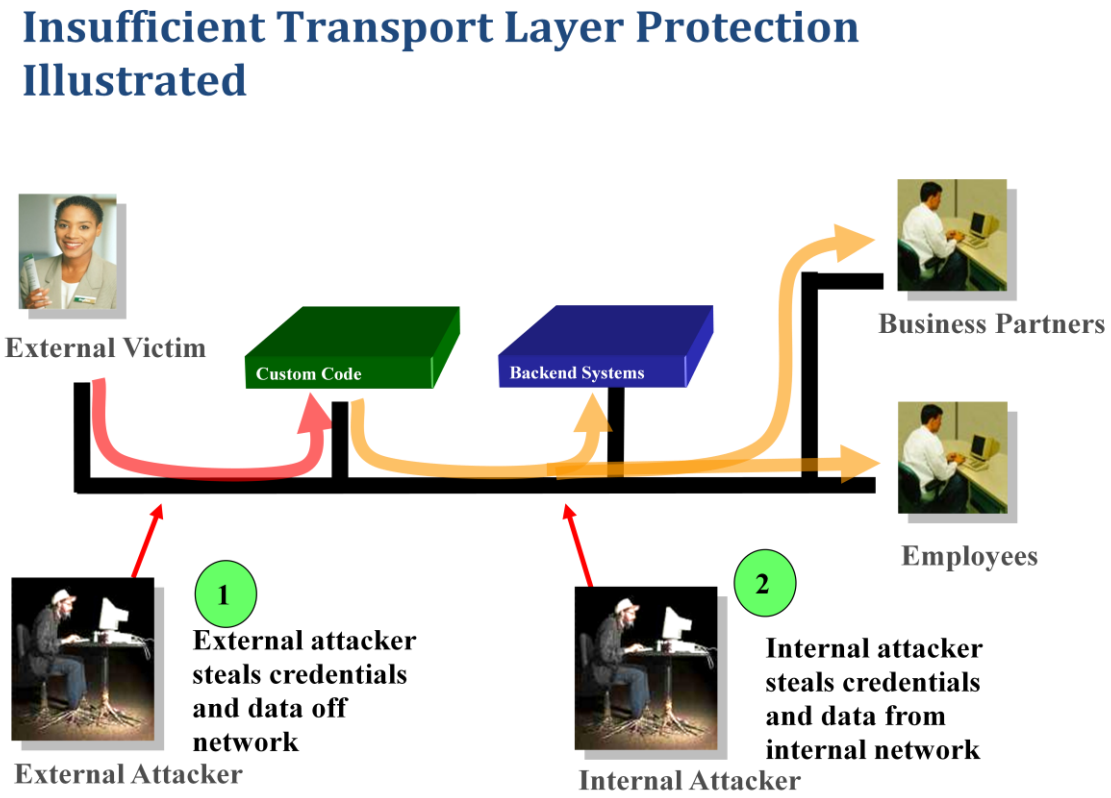


Tento útok spočíva v nedostatočnom zabezpečení všetkých citlivých dát. Ako vidno na obrázku, aj keď priame citlivé údaje sú čísla kariet, tieto je možné vyčítať z logov. Tie sú potom taktiež citlivými dátami, ktoré je treba chrániť.

Pre ochranu citlivých dát existuje viacero možností. Je potrebné obmedziť prístup k takýmto dátam len pre vybrané osoby, tieto dáta je možné kryptovať rôznymi algoritmami (možno kryptovať databázy, súbory...)

V tomto prípade je možné testovať návrh architektúry celého systému, riadenie prístupov, skontrolovanie, či sú kryptované všetky dáta, ktoré majú byť chránené a či tieto algoritmy fungujú správne. Tieto prípady je možné testovať zrejme len ručne. Možno však využiť nástroje napr. pre brute force attack (http://freeworld.thc.org/thc-hydra/) , ktorý by mohol odhaliť slabé kryptovanie dát.

10 Insufficient Transport Layer Protection



Je síce dôležité chrániť samotnú aplikáciu alebo systém, ale pokiaľ nie sú chránené prenosové kanály medzi jednotlivými časťami alebo medzi používateľom a aplikáciou, útočník môže odchytiť alebo pozmeniť dáta tečúce cez transportnú vrstvu (internet, vnútorná sieť...) a tak získať prístup k cenným informáciám.

Ako ochranu je možné využívať SSL/TLS protokoly pre šifrovanie dát. V príručke OWASP Testing Guide je napr. celá kapitola 4.3.1 venovaná testovaniu SSL/TLS.

Celkom zaujímavý komerčný nástroj ktorý okrem iného testuje aj tento typ útoku je http://www.toptechwire.com/article/Core_Security_Launches_Professional_Services/71171/index.html

11 Ďalší prehľad nástrojov na testovanie

(prebratý z OWASP Test Guide, len kvôli prehľadu, koľko je rôznych nástrojov)

OPEN SOURCE BLACK BOX TESTING TOOLS

General Testing

- OWASP WebScarab
- OWASP CAL9000: CAL9000 is a collection of browser-based tools that enable more effective and efficient manual testing
efforts. Includes an XSS Attack Library, Character Encoder/Decoder, HTTP Request Generator and Response Evaluator, Testing
Checklist, Automated Attack Editor and much more.
- OWASP Pantera Web Assessment Studio Project
- SPIKE - http://www.immunitysec.com
- Paros - http://www.parosproxy.org
- Burp Proxy - http://www.portswigger.net
- Achilles Proxy - http://www.mavensecurity.com/achilles
- Odysseus Proxy - http://www.wastelands.gen.nz/odysseus/
- Webstretch Proxy - http://sourceforge.net/projects/webstretch
- Firefox LiveHTTPHeaders, Tamper Data and Developer Tools - http://www.mozdev.org
- Sensepost Wikto (Google cached fault-finding) - http://www.sensepost.com/research/wikto/index2.html
- Grendel-Scan - http://www.grendel-scan.com

TESTING FOR SPECIFIC VULNERABILITIES

Testing Flash

- OWASP SWFIntruder - http://www.owasp.org/index.php/Category:SWFIntruder, http://www.mindedsecurity.com/swfintruder.html

Testing AJAX

- OWASP Sprajax Project

Testing for SQL Injection

- OWASP SQLiX
- Multiple DBMS SQL Injection tool - SQL Power Injector
- MySQL Blind Injection Bruteforcing, Reversing.org - [sqlbftools]
- Antonio Parata: Dump Files by SQL inference on Mysql - [SqlDumper]
- Sqlninja: a SQL Server Injection & Takeover Tool - http://sqlninja.sourceforge.net
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - http://sqlmap.sourceforge.net
- Absinthe 1.1 (formerly SQLSqueal) - http://www.0x90.org/releases/absinthe/
- SQLInjector - http://www.databasesecurity.com/sql-injector.htm
- bsqlbf-1.2-th - http://www.514.es

Testing Oracle

- TNS Listener tool (Perl) - http://www.jammed.com/%7Ejwa/hacks/security/tnscmd/tnscmd-doc.html
- Toad for Oracle - http://www.quest.com/toad

Testing SSL

- Foundstone SSL Digger - http://www.foundstone.com/resources/proddesc/ssldigger.htm

Testing for Brute Force Password

338

- THC Hydra - http://www.thc.org/thc-hydra/
- John the Ripper - http://www.openwall.com/john/
- Brutus - http://www.hoobie.net/brutus/
- Medusa - http://www.foofus.net/~jmk/medusa/medusa.html

Testing for HTTP Methods

- NetCat - http://www.vulnwatch.org/netcat

Testing Buffer Overflow

- OllyDbg - http://www.ollydbg.de

o "A windows based debugger used for analyzing buffer overflow vulnerabilities"

• Spike - http://www.immunitysec.com/downloads/SPIKE2.9.tgz
o A fuzzer framework that can be used to explore vulnerabilities and perform length testing
• Brute Force Binary Tester (BFB) - http://bfbtester.sourceforge.net
o A proactive binary checker
• Metasploit - http://www.metasploit.com/projects/Framework/
o A rapid exploit development and Testing frame work
Fuzzer
• WSFuzzer
Googling
• Foundstone Sitedigger (Google cached fault-finding) -
http://www.foundstone.com/resources/proddesc/sitedigger.htm

COMMERCIAL BLACK BOX TESTING TOOLS

• Typhon - http://www.ngssoftware.com/products/internet-security/ngs-typhon.php
• NGSSQuirreL - http://www.ngssoftware.com/products/database-security/
• Watchfire AppScan - http://www.watchfire.com
• Cenzic Hailstorm - http://www.cenzic.com/products_services/cenzic_hailstorm.php
• SPI Dynamics WebInspect - http://www.spidynamics.com
• Burp Intruder - http://portswigger.net/intruder
• Acunetix Web Vulnerability Scanner - http://www.acunetix.com
• ScanDo - http://www.kavado.com
• WebSleuth - http://www.sandsprite.com
• NT Objectives NTOSpider - http://www.ntobjectives.com/products/ntospider.php
• Fortify Pen Testing Team Tool - http://www.fortifysoftware.com/products/tester
• Sandsprite Web Sleuth - http://sandsprite.com/Sleuth/
• MaxPatrol Security Scanner - http://www.maxpatrol.com
• Ecyware GreenBlue Inspector - http://www.ecyware.com
• Parasoft WebKing (more QA-type tool)
• MatriXay - http://www.dbappsecurity.com
• N-Stalker Web Application Security Scanner - http://www.nstalker.com

SOURCE CODE ANALYZERS - OPEN SOURCE / FREEWARE

• OWASP LAPSE
• PMD - http://pmd.sourceforge.net/
• FlawFinder - http://www.dwheeler.com/flawfinder
• Microsoft's FxCop
• Splint - http://splint.org
OWASP Testing Guide v3.0
339
• Boon - http://www.cs.berkeley.edu/~daw/boon
• Pscan - http://www.striker.ottawa.on.ca/~aland/pscan
• FindBugs - http://findbugs.sourceforge.net

SOURCE CODE ANALYZERS - COMMERCIAL

• Fortify - http://www.fortifysoftware.com
• Ounce labs Prexis - http://www.ouncelabs.com
• Veracode - http://www.veracode.com
• GrammaTech - http://www.grammatech.com
• ParaSoft - http://www.parasoft.com
• ITS4 - http://www.cigital.com/its4
• CodeWizard - http://www.parasoft.com/products/wizard
• Armorize CodeSecure - http://www.armorize.com/product/
• Checkmarx CxSuite - http://www.checkmarx.com