GHID PENTRU SECURIZAREA APLICAŢIILOR ŞI SERVICIILOR WEB

Cuprins

Ghid pentru securizarea aplicațiilor și serviciilor web

Aplicațiile web au cunoscut o dezvoltare uluitoare de-a lungul ultimilor ani. Odată cu intrarea în era WEB 2.0 și dezvoltarea cloud computing-ului o mare parte din activitatea internauților s-a mutat în mediul web.

Dezvoltarea spectaculoasă a aplicațiilor web a fost posibilă datorită inovațiilor tehnologice în domeniu, specifice WEB 2.0, care au transformat simplele pagini web în care erau afișate informații statice, în pagini dinamice, interactive ce permit o interacțiune ridicată a utilizatorului cu aplicația.

Dezvoltarea uluitoare a web-ului a creat premisa apariției vulnerabilităților specifice oricărui produs tehnologic. Numărul acestora este în continuă creștere deși cele mai populare atacuri se bazează pe vulnerabilități identificate pentru prima dată acum câțiva ani buni.

Scopul acestui ghid este să facă o trecere în revistă a principalelor vulnerabilități specifice aplicațiilor web precum și modalitatea de detectare și tratare a acestora. Ghidul este destinat oricărei organizații publice sau private ce dorește securizarea aplicațiilor web proprii, respectiv a site-ului public.

Internetul abundă în numeroase surse de documentație pentru securizarea aplicațiilor web precum și de topuri ale celor mai populare tipuri de atacuri asupra serverelor web. Prezentul ghid are ca sursă de inspirație documentația disponibilă pe owasp.org precum și raportul Trustwave Global Security Report pe 2011.

Ghidul se va concentra pe următoarele tipuri de vulnerabilități specifice aplicațiilor web:

1. SQL Injection
2. Cross-site Scripting (XSS)
3. Verbose Errors
4. Logic Flaw
5. Authorization Bypass
6. Authentication Bypass
7. Vulnerable Third Party Software
8. Session Handling Flaw

9. Cross-site Request Forgery (CSRF)
10. Source Code Disclosure

Tehnicile prezentate sunt simpliste și au scopul de a atrage atenția asupra modului de funcționare al vulnerabilității. Atacurile reale sunt mult mai complexe.

1. SQL Injection

Majoritatea proiectelor web folosesc baze de date pentru a-şi stoca datele. Limbajul SQL este un standard internațional ce stabilește regulile și sintaxa comenzilor ce pot fi transmise unui sistem de gestiune a bazelor de date (SGBD), pentru manipularea datelor stocate. Implementarea acestuia poate să difere de la caz la caz în funcție de producător. În prezent cele mai populare SGBD-uri sunt MySQL, PostgreSQL, Microsoft SQL Server și Oracle.

O vulnerabilitate de tip **SQL injection** (injecţie cu cod sursă SQL) apare atunci când un atacator poate introduce orice date într-o interogare SQL transmisă unei baze de date sau când, prin injectarea sintaxei, logica declaraţiei este modificată în asemenea fel încât să execute o acţiune diferită. Injecţia SQL poate fi crucială pentru sistem dar, în ciuda pericolului pe care îl prezintă, este cea mai frecvent întâlnită vulnerabilitate.

EXEMPLU SQL INJECTION	

Funcția PHP ce interogheaz BD	<?php function autentificare($user,
	$parola)
	{
	$sql=”SELECT * FROM users WHERE unername=’$user’ and parola =’$parola’)”; Return mysql_query($sql); …..
	}
	?>

Ce va introduce atacatorul în câmpul destinat numelui de utilizator	Eu’ OR 1=1

Exemplul de mai sus vizează un formular de autentificare ce are în spate funcția PHP din tabel. Datorită faptului că datele de intrare, respectiv numele utilizatorului și parola nu sunt validate (adică să ne asigurăm ca atacatorul va introduce un nume valid de utilizator și nu alte date) atacatorul va putea insera cod SQL, în câmpul de user, astfel încât sensul interogării SQL se schimbă complet, permițând ocolirea mecanismului de autentificare.

Cea mai bună strategie de apărare împotriva injecţiilor SQL se bazează pe implementarea unor rutine puternice de validare a datelor de intrare, astfel încât atacatorul să nu poată introduce alte date decât cele necesare aplicației.

Între măsurile specifice care pot fi implementate la nivelul bazelor de date şi aplicaţiilor se regăsesc următoarele:

a) Utilizarea de variabile bine definite şi de definiţii ale coloanelor din baza de date

Stocarea şi manipularea numerelor (ID-uri de sesiune, coduri etc.) ca şi numere întregi sau ca alte tipuri numerice potrivite. String-urile (varchars) ar trebui să conţină doar caractere alfanumerice şi să respingă semnele de punctuaţie şi caracterele specifice sintaxei SQL.

b) Atribuirea rezultatelor interogării unei variabile bine definite

Dacă aplicaţia caută valori numerice, atunci atribuiţi rezultatul unui număr întreg, acest lucru împiedicându-i pe atacatori să extragă informaţii din baza de date. De exemplu nu ar trebui să fie posibilă obţinerea şi afişarea numelui unei coloane, dacă variabila ce urmează să fie afişată în browser nu acceptă decât numere întregi. Această tehnică restricţionează sever anumite atacuri.

c) Limitarea lungimii datelor

Toate şirurile de caractere ar trebui să se limiteze la o lungime potrivită scopului lor. Un nume de utilizator, de exemplu, nu este necesar să fie stocat şi manipulat într-o variabilă care utilizează 256 de caractere. Limitarea numărului de caractere, care poate fi introdus într-un câmp, poate împiedica în mod eficient succesul unei injecţii SQL, reducând lungimea şirului de caractere pe care atacatorul îl poate introduce în cod.

d) Evitarea creării de interogări prin concatenarea de şiruri de caractere

Creaţi o funcţie view sau o procedură care operează asupra variabilelor furnizate de aplicaţie. Concatenarea şirurilor de caractere, unde interogarea este formată direct din datele furnizate de utilizatori (de genul: “SELECT something FROM table WHERE” + variable a), este cea mai vulnerabilă la atacurile SQL Injection. Pe de altă parte, o funcţie view sau o procedură particularizată, generează de obicei o eroare dacă primeşte date de intrare incorecte, însă nu îi va permite unui atacator să manipuleze întreaga interogare.

e) Aplicarea separării datelor şi accesul pe baza de rol în interiorul bazei de date

Aplicaţia ar trebui să folosească un cont care are privilegii de acces doar pentru tabelele necesare respectivei funcţii. Tabelele interne ale bazei de date, în special cele legate de managementul conturilor şi variabilele sistemului, nu ar trebui să fie accesibile.

2. Cross-site Scripting (XSS)

XSS este o tehnică de atac, folosită pentru a forţa o pagină web să afişeze un cod maliţios (scris de obicei în HTML, JavaScript, ActiveX sau Flash), pe care îl execută ulterior în browser-ul unui utilizator. Acest tip de atac nu are ca ţintă serverul siteului web, codul malware fiind executat direct în browser, deoarece adevărata ţintă a atacului este utilizatorul.

Hackerul va folosi site-ul doar pentru a efectua atacul şi odată ce are control asupra browser-ului utilizatorului, îl va putea folosi pentru a-i fura diferite date: conturi bancare, conturi de utilizator, parole, furtul înregistrărilor din istoricul browser-ului etc.

Sunt mai multe modalităţi prin care un malware scris în JavaScript poate deveni rezident pe o pagină web:

- Proprietarul paginii web îl poate încărca intenţionat;
- Un atacator îl poate injecta în secțiunea publică a unui site profitând de anumite vulnerabilități ale acesteia (vulnerabilitate permanentă).

- Pagina web poate primi un deface folosind o vulnerabilitate a reţelei sau a straturilor sistemului de operare, iar parte din codul introdus să fie malware JavaScript:
- Victima poate accesa un link special pregătit (transmis prin mail sau alte metode) în spatele căruia se ascunde un XSS non-persistent sau bazat pe Document Object Model (DOM).

Un exemplu simplist, dar concludent asupra unui astfel de atac, este reprezentat mai jos:

http://www.website.com/registration.php?idclient=<SCRIPT>alert(document.cookie)</SCRIPT>

Atacul de tip non-persistent: Dacă atacatorul doreşte să atace prin XSS pagina http://vulnerabil.com/ - spre exemplu un site de comerţ electronic, mai întâi trebuie să găsească o vulnerabilitate la XSS. În acest scop caută un parametru de unde utilizatorul poate trimite mesaje la server şi la care primeşte mesaje înapoi (de obicei un câmp pentru căutare). Dacă introduce "test pentru xss" în câmpul de căutare, răspunsul va fi un nou url cu un şir de interogare care conţine "test+pentu+xss" ca şi valoare a parametrului p. Această valoare poate fi schimbată dacă introducem codul HTML/JavaScript: "><script>alert('XSS%20Test'). Ca şi rezultat pagina va afişa o fereastră de dialog inofensivă (după instrucţiunea din cod) care este acum parte din pagină, demonstrând succesul codului care acum face parte din http://vulnerabil.com/. De aici URL-ul poate fi modificat să conţină atacuri XSS mai complexe (ex: furtul de cookie-uri).

Atacul bazat pe DOM: Atacul XSS bazat pe DOM este o formă unică de cross-site scripting (xss), foarte similară cu cel non-persistent, dar fără a fi nevoie să trimitem un mesaj şi să aşteptăm răspuns.

Considerăm pagina de comerţ electronic din exemplul anterior - http://vulnerabil.com/ - având o caracteristică în plus pentru afişarea promoţiilor şi interogările din URL-urile pentru afişarea produselor îşi trag datele direct din backend-ul bazei de date (ex: id_produs) pentru a le afişa utilizatorului.

Putem manipula URL-urile pentru a afişa mesaje diferite sau putem adauga malware la sfârşitul URL-ului în acest fel:

Din: http://vulnerabil/promo?product_id=100&title=Last+Chance!
În: http://victim/promo?product_id=100&title=Foo#<script>alert('XSS%20Testing')</script>

În acest caz, JavaScript-ul de pe partea clientului are încredere suficientă în datele conţinute de URL şi le afişează pe ecran. Ce face acest stil de atac XSS diferit este că nu se trimite codul malware la serverul web, ci fragmentul din URL nou adăugat îi spune browser-ului în ce punct al documentului curent să sară (rămâne în cadrul DOM, de aici şi numele).

Atacul de tip persistent: Atacul XSS persistent sau injecţia cu cod HTML nu necesită link-uri special pentru execuţie, tot ce trebuie hackerul să facă este să adauge codul XSS într-o parte a paginii web care are potenţial mare de a fi vizitată de către utilizatori (comentariile de pe bloguri, posturile de pe forumuri, chat-uri etc.). Odată ce utilizatorul vizitează pagina infectată, execuţia este automată ceea ce face ca acest tip de atac să fie mult mai periculos decât primele două, deoarece nu există cale prin care utilizatorul se poate apăra şi chiar şi utilizatorii care ştiu despre această vulnerabilitate pot fi uşor compromişi.

Metode de prevenire a atacurilor de tip Cross-site scripting (XSS)

Codarea datelor de intrare şi de ieşire au fiecare argumentele lor pozitive şi negative. Partea pozitivă a codificării datelor de intrare oferă un singur punct de acces, în timp ce codarea datelor de ieşire oferă posibilitatea de a face faţă tuturor utilizărilor textului şi poziţionarea acestuia în pagina. Părţile negative sunt că nici codarea datelor de intrare nu poate opri un atac XSS persistent odată ce a fost stocat, iar codarea datelor de ieşire nu poate opri alte forme de atac, cum ar fi injecţia cu cod SQL, deoarece intervine prea târziu.

Există un număr de soluţii de a vă proteja în calitate de client. Nişte idei simple sunt:

- alegerea unui browser securizat;
- folosirea unei maşini virtuale;
- accesarea doar a link-urilor cunoscute;
- grijă la ce informaţii divulgaţi despre conturile dumneavoastră etc.

a) Filtrarea

Filtrarea poate produce rezultate neaşteptate dacă nu monitorizaţi atent datele de ieşire.

Folosirea unei bucle poate reduce riscurile asociate cu filtrarea de conţinut. Doar filtrarea, fără folosirea altor metode, poate introduce noi riscuri prin crearea unor noi tipuri de atac, aşadar, este important să înţelegeţi în ce ordine trebuie aplicate filtrele şi cum interacţionează unul cu celălalt.

b) Codarea și validarea datelor de intrare

Această tehnică poate crea un singur punct de intrare a datelor pentru toate codările.

Vă poate proteja împotriva vulnerabilității XSS, dar şi de injecţii cu cod SQL şi injecţii de comandă, care pot fi verificate înainte de a stoca informaţii în baza de date. Nu poate opri atacurile persistente de tip XSS odată stocate.

c) Codarea datelor de ieşire

Această tehnică este mai detaliată şi poate lua în considerare şi contextul. Este posibil ca dezvoltatorii să trebuiască să efectueze codarea de mai multe ori pentru aceeaşi locaţie unde este trimisă informaţia.

d) Securitatea browser-ului web

Evitaţi URL-urile prea lungi sau prea complexe, acestea sunt cel mai probabil să conţină vulnerabilităţi. Nu accesaţi URL-uri necunoscute primite prin e-mail, dacă este posibil.

Alegeţi un browser sigur, actualizat la zi şi personalizaţi-vă setările de securitate pentru a reduce riscul de atac.

3. Mesaje de eroare prea detaliate (verbose error messages)

Nu sunt un tip de atac în sine, însă mesajele de eroare cu scop informativ pot conţine adresele complete şi numele fişierelor, descrieri ale tabelelor SQL, erori ale bazei de date sau alte erori legate de aplicaţie şi mediul în care rulează. Un formular tipic de autentificare îi cere utilizatorului să introducă două informaţii (nume de utilizator şi parolă), alte aplicaţii cer mai multe informaţii (data naşterii, un cod PIN). Când un

proces de autentificare dă greş, poţi, în mod evident, să îţi dai seama că una din informaţiile introduse nu au fost corecte, însă uneori aplicaţia te anunţă care din ele a fost greşită.
Acest lucru poate fi folosit pentru a diminua eficienţa mecanismului de autentificare. În cel mai simplu caz, unde autentificarea cere nume de utilizator şi parolă, aplicaţia poate răspunde la o autentificare nereuşită prin identificarea motivului (nu a recunoscut numele de utilizator sau parola este greşită).
Atacul: Într-un asemenea caz, puteţi folosi o metodă automată de atac, care să parcurgă o listă mare de nume comune de utilizatori pentru a afla care din ele sunt valide, deşi în general numele utilizatorilor nu sunt considerate a fi secrete, identificarea lor îi dă atacatorului şanse mai mari de a compromite aplicaţia folosindu-se de timp, abilitate şi efort. O listă de nume de utilizatori enumerați poate fi folosită ulterior pentru diverse metode de atac incluzând: ghicirea parolelor, atacuri asupra datelor utilizatorilor, sesiuni sau inginerie socială. În procesele de autentificare mai complexe, unde aplicaţia cere utilizatorului să introducă mai multe informaţii sau să treacă prin mai multe etape, mesajele de eroare verbose sau alţi discriminatori pot ajuta un atacator să treacă prin fiecare etapă a autentificării, crescându-i şansele de a obţine acces neautorizat.

Metode de prevenire a atacurilor bazate pe Verbose Errors

a) Utilizaţi validările pe partea clientului doar pentru performanţă, nu şi pentru securitate

Mecanismele de verificare a datelor introduse pe partea clientului previn erorile de introducere şi de tipar nevinovate să ajungă la server, acest pas de anticipare a validării putând reduce solicitarea serverului, împiedicând datele introduse greşit în mod neintenţionat să ajungă la acesta. Totuși atacatorul poate opri execuția scripturilor locale și poate avea acces la mesajele de eroare ale bazei de date.

b) Normalizaţi datele de intrare

Multe atacuri folosesc o multitudine de codări diferite bazate pe seturi de caractere şi reprezentări hexadecimale. Datele de intrare ar trebui canonizate înainte de

verificarea de securitate şi validare, altfel o bucată de cod poate trece prin filtre şi să fie decodată şi descoperită ca fiind maliţioasă doar mai târziu.

c) Aplicaţi validarea pe partea serverului

Toate datele de la browser pot fi modificate cu conţinut arbitrar, aşadar, validarea datelor introduse ar trebui făcută de server, unde evitarea funcţiilor de validare nu este posibilă.

d) Restrângeţi tipurile de date care pot fi introduse

Aplicaţia nu ar trebui să conţină tipuri de date care nu îndeplinesc tipul de bază, formatul şi lungimea cerute.

e) Utilizaţi codarea securizată a caracterelor şi validarea datelor de ieşire

Caracterele utilizate în formatele HTML şi SQL ar trebui codate în aşa măsură încât să împiedice aplicaţia să le interpreteze greşit. Acest tip de validare a datelor de ieşire sau de reformatare a caracterelor reprezintă un nivel adiţional de protejare împotriva atacurilor prin injectare HTML. Chiar dacă un cod maliţios reuşeşte să treacă de un filtru de intrare a datelor, efectele acestuia vor fi neglijate în momentul în care ajunge în faza de ieşire.

f) Utilizaţi white lists şi black lists

Utilizaţi expresii obişnuite pentru a căuta dacă datele fac parte din conţinut autorizat sau neautorizat - white lists conţin tiparele de date acceptate, iar black lists conţin tipare de date neacceptate sau maliţioase.

g) Aveţi grijă cu mesajele de eroare

Indiferent de limbajul folosit pentru a scrie aplicaţia, erorile ar trebui să urmărească conceptele de încercare, descoperire, în final când vine vorba de tratarea excepţiilor. Încercaţi o acţiune, descoperiţi excepţiile specifice care pot fi cauzate de acea acţiune; în final închideţi aplicaţia dacă nimic altceva nu funcţionează. De asemenea, creaţi un mesaj de eroare custom sau o pagină specială de erori, care nu dezvăluie nicio informaţie despre sistem.

h) Solicitaţi autentificare

În unele cazuri s-ar putea să fie necesar să configuraţi serverul, în aşa măsură încât să fie solicitată autentificarea la nivelul de director pentru toate fişierele din interiorul acelui director.

4. Eludarea restricţiilor de autentifiare (authentication bypass)

Autentificarea dovedeşte, într-o oarecare măsură, identitatea unei persoane sau entităţi. De exemplu, toţi folosim parole pentru a ne autentifica în conturile personale de e-mail. Paginile web folosesc certificate Secure Socket Layer (SSL) pentru a valida faptul că traficul provine într-adevăr de la domeniul solicitat de către site, acest lucru neasigurându-ne că site-ul este cel adevărat şi nu o copie.

Atacatorul are două opţiuni pentru a sparge un sistem de autentificare: utilizarea unei parole furate sau evitarea verificării autentificării. Pentru a identifica şi monitoriza activitatea unui utilizator pe o pagina web, acestuia i se atribuie un token de sesiune unic, de obicei sub formă de cookie-uri. Odată autentificat, utilizatorul respectiv este identificat doar după cookie-ul de sesiune, deci dacă un atacator îl compromite, ghicindu-i valoarea sau furându-l, reuşeşte să treacă cu succes de mecanismul de autentificare a paginii respective şi să îi ia locul victimei. Cookie-urile de sesiune pot fi compromise prin mai multe metode:

a) Cross-site scripting (XSS): dacă atributul HttpOnly nu este setat JavaScript poate accesa obiectul document.cookie. Cea mai simplă formă de atac este de forma:

**

Acest cod trimite numele cookie-ului unui site unde atacatorul poate vedea traficul venit din exterior.

b) Cross-site request forgery (CSRF): atacatorul exploatează indirect sesiunea unui utilizator, pentru asta victima trebuie să fie deja autentificată pe site-ul ţintă. Atacatorul plasează o pagină capcană pe un alt site, iar când victima vizitează pagina infectată, browser-ul face în mod automat o cerere către pagina ţintă folosind cookie-ul de sesiune al victimei.

c) SQL Injection: unele aplicaţii web stochează cookie-urile de sesiune într-o bază de date, în loc să le stocheze într-un sistem de fişiere sau spaţiu de memorie al serverului web, iar dacă un atacator sparge baza de date poate fura cookie-urile de sesiune.

d) Network snifffing: HTTPS criptează traficul dintre browser şi pagina web pentru a oferi confidenţialitate şi integritate comunicaţiilor dintre ele, majoritatea formularelor de autentificare sunt trimise prin HTTPS, însă majoritatea aplicaţiilor web folosesc HTTP pentru restul paginilor, HTTPS protejează parola utilizatorului, în timp ce HTTP expune cookie-ul de sesiune în văzul tuturor, mai ales prin reţelele wireless din locurile publice (cafenele, aeroporturi, şcoli, etc.).

5. Eludarea restricţiilor de autorizare (authorization bypass)

Vulnerabilităţile ce ţin de autorizare şi acces pot apărea oriunde în aplicaţia web şi se referă la ce se întâmplă atunci când un atacator are acces la o resursă la care în mod normal au acces doar utilizatorii autentificaţi sau care deţin anumite privilegii în acele aplicaţii.

Cele mai folosite tehnici pentru eludarea restricţiilor referitoare la autorizare:

- Traversarea de directoare (directory traversal);
- Evitarea unor mecanisme de autorizare; - Escaladarea privilegiilor.

Serverele restricţionează utilizatorii care navighează pe un site la documentul rădăcină al acestuia, a cărui localizare depinde de sistemul de operare instalat pe server, şi adiţional în funcţie de permisiunile de citire/scriere/execuţie pe care le are utilizatorul respectiv asupra fişierelor de pe server.

Subminarea unui script de execuţie pentru a traversa directoarele serverului şi a citi fişiere protejate cum ar fi /etc/passwd este cunoscută ca atac cu traversare de directoare.

Cum aproape toate aplicaţiile web folosesc roluri (ex: utilizator neautentificat / utilizator autentificat / administrator) care pot avea diferite nivele de acces, un atacator poate reuşi să acceseze un privilegiu restricţionat nivelului lui de acces şi să

evite mecanismul de autorizare, acest lucru se face de obicei prin schimbarea rolului într-unul superior.

Atacul

Avem sursa:

http://www.exemplu.com/index.php?page=login.

Un atac de tip traversare de directoare al cărui scop este de a afişa fişierul /etc/passwd poate fi realizat prin a schimba URL-ul în:

http://www.exemplu.com/index.php?page=../../../../../../../etc/passwd.

Luaţi în considerare o cerere HTTP făcută unui administrator pentru a reseta parola unui utilizator:

Post / admin / resetPassword.jsp HTTP/1.1
Realizator: www.examplu.com
[HTTP Headers]
utilizator = admin & newpassword = parolă

Dacă atacatorul poate face o cerere identică şi aplicaţia web resetează parola unui cont de administrator, atacatorul evită mecanismul de autentificare, deoarece această funcţie era gândită pentru a fi folosită doar de administratorii aplicației, întrun sens este o creştere a privilegiilor deoarece un non-administrator poate folosi funcţia de resetare a parolelor şi obţine acces la contul de administrator cu noua parolă.

Metode de prevenire a atacurilor de tip Authorisation Bypass

Cele mai simple metode de protecţie împotriva acestui tip de atac sunt validarea datelor introduse de utilizatori şi urmărirea metodelor de proiectare sigură. Este important să fie identificată din timp orice parte a aplicaţiei web care poate fi folosită într-un eventual atac informatic, acest lucru nereferindu-se doar la câmpurile în care utilizatorul poate introduce date, ci şi la orice valoare pe care utilizatorul o poate modifica şi trimite prin intermediul unui proxy, cum ar fi datele din cookie-uri, câmpurile ascunse etc.

Aceste date ar trebui validate corespunzător, înainte de a putea trece mai departe. Utilizaţi de asemenea principiul de a da cu atât mai puţine privilegii utilizatorului, cu cât scad şansele de a putea duce la capăt un atac asupra aplicaţiei web.

6. Vulnerabilităţi în managementul sesiunilor

Autentificarea şi managementul sesiunilor includ toate aspectele ce ţin de manipularea datelor de autentificare ale utilizatorului şi managementul sesiunilor active ale acestuia. Autentificarea este un proces critic al acestui aspect, dar până şi cel mai solid proces de autentificare poate fi subminat de erori ale funcţiilor pentru verificarea credenţialelor, incluzând: schimbarea parolelor, funcţia de recuperare a parolelor uitate, funcţia de amintire a parolelor de către aplicaţia web, update-uri ale conturilor şi alte funcţii legate de acestea. Pentru a evita astfel de probleme, pentru orice fel de funcţii legate de managementul conturilor, ar trebui să ceară reautentificarea utilizatorului, chiar dacă acesta are un id de sesiune valid.

Autentificarea utilizatorilor pe internet, de obicei, necesită un nume de utilizator şi o parolă. Există metode mai bune de autentificare pe piaţă de tip hardware şi software bazate pe token-uri criptate şi biometrie, însă acestea nu sunt foarte răspândite datorită costurilor mari de achiziţionare. O gamă largă de erori legate de conturi şi managementul sesiunilor rezultă în urma compromiterii conturilor utilizatorilor sau celor de administrare a sistemului.

Echipele de dezvoltare, de cele mai multe ori, subestimează complexitatea necesară pentru a proiecta o metodă de autentificare şi management al sesiunilor care să protejeze corespunzător credenţialele în toate aspectele aplicaţiei web. Paginile web au nevoie de sesiuni pentru a putea monitoriza valul de cereri venit de la fiecare utilizator în parte, iar, cum protocolul HTTP (stateless) nu poate face acest lucru, fiecare aplicaţie web trebuie să şi-l facă singură. De cele mai multe ori, mediul aplicaţiilor web oferă asemenea capabilităţi, însă mulţi dezvoltatori preferă să-şi creeze propriile token-uri de sesiune.

Dacă toate credenţialele de autentificare şi identificatorii de sesiune nu sunt protejate corespunzător (prin SSL), protejate împotriva divulgării şi alte tipuri de

erori, cum ar fi vulnerabilitatea la cross-site scripting, un atacator poate fura sesiunea unui utilizator şi să îşi asume identitatea acestuia.
Toate serverele web, serverele de aplicaţii şi mediile aplicaţiilor web cunoscute sunt susceptibile la problemele legate de evitarea mecanismelor de autentificare şi de management al sesiunilor. Acest gen de vulnerabilitate se bazează mult pe eroare umană şi tehnologii care nu îndeplinesc standardele de securitate necesare.

Metode de prevenire a vulnerabilităţilor legate de managementul sesiunilor

a) **Complexitatea parolelor**

Parolele ar trebui să aibă restricţii care cer un număr minim de caractere şi de complexitate; de asemenea ar trebui să li se ceară utilizatorilor să îşi schimbe periodic parola şi să le fie interzis să refolosească o parolă veche.

b) **Utilizarea de parole**

Utilizatorilor ar trebui să le fie limitat numărul de logări pe care le pot încerca într-o anumită unitate de timp iar tentativele eşuate de autentificare ar trebui logate, însă parolele introduse nu ar trebui înregistrate, deoarece acest lucru poate expune parola utilizatorului oricui reuşeşte să obţină accesul la loguri.
De asemenea, sistemul nu trebuie să indice motivul pentru care procesul de autentificare nu a reuşit, iar utilizatorul să fie informat cu privire la data ultimei autentificări reuşite şi numărul de autentificări nereuşite de atunci.

c) **Comenzile de schimbare a parolelor**

Ar trebui folosit un singur mecanism de schimbare a parolelor indiferent de circumstanţele în care acest lucru se întâmplă. Utilizatorul trebuie să scrie întotdeauna vechea parolă şi noua parolă de fiecare dată. Dacă parolele uitate sunt trimise utilizatorului prin e-mail, sistemul ar trebui să-i ceară utilizatorului să se reautentifice atunci când îşi schimbă adresa de e-mail, altfel un atacator care are acces la token-ul de sesiune temporar al utilizatorului, poate pur şi simplu să schimbe adresa la care să fie trimisă parola uitată.

***d)* Stocarea parolelor**

Toate parolele trebuie criptate sau sub formă de hash-uri indiferent de locul unde sunt stocate.

***e)* Protejarea ID-ului de sesiune**

În mod ideal, întreaga sesiune a utilizatorului ar trebui protejată prin SSL (în acest mod cookie-ul de sesiune nu ar putea fi furat).

***f)* Liste de conturi**

Sistemele ar trebui proiectate în aşa fel încât să nu permită accesul utilizatorilor la lista de conturi înregistrate pe site. Dacă este imperativ să fie prezentată o listă de acest gen se recomandă folosirea pseudonimelor în locul numelor reale. În acest fel, pseudonimul nu poate fi folosit pentru logare în cont în timpul unei încercări de autentificare a unui atacator pe site.

***g)* Relaţionări bazate pe încredere**

Arhitectura paginii dumneavoastră ar trebui să evite relaţiile implicite de încredere între componente ori de câte ori este posibil acest lucru. Fiecare componentă în parte ar trebui să se autentifice faţă de o alta cu care interacţionează.

7. Cross-site Request Forgery (CSRF)

Cross-site Request Forgery (CSRF sau XSRF) este o formă de atac asupra aplicaţiilor web care se foloseşte de relaţiile de încredere existente între aplicaţiile web şi utilizatorii autentificaţi prin a forţa acei utilizatori să facă tranzacţii sensibile în numele atacatorului. Această vulnerabilitate, deşi mai puţin cunoscută ca XSS, este mult mai periculoasă decât cross-site scripting, deoarece îşi are rădăcinile în natura

lipsită de stare (stateless) ale specificaţiilor HTTP-ului, care cer ca un token de autentificare să fie trimis cu fiecare cerere a utilizatorului.
În mod obişnuit, vulnerabilităţile web apar ca urmare a unor greşeli făcute de dezvoltatorii paginilor web în timpul proiectării şi dezvoltării acestora sau de către administratori, în timpul utilizării acestora. Spre deosebire de restul, vulnerabilităţile de tip XSRF apar atunci când dezvoltatorii omit un mecanism de prevenire a XSRF din aplicaţia lor.

Atacul. Un exemplu clasic este cel al unei aplicaţii bancare care le permite utilizatorilor să transfere fonduri dintr-un cont în altul folosind o cerere simplă GET prin HTTP. Presupunem că aplicaţia foloseşte următoarea modalitate de a transfera fondurile:

http://xsrf.bancavulnerabila.com/transferFonduri.aspx?Incontul=12345&fonduri=1000.00&valuta=euro

Continuând cu exemplul de mai sus, presupunem că un atacator creează o pagina HTML maliţioasă pe un sistem care se află sub controlul lui şi care conţine următorul cod JavaScript:

```
<script type a??text/javascript”>
Var i document.createElement(a??image”);
i.src="http://xsrf.bancavulnerabila.com/transferFonduri.aspx?"Incontul=ATACATOR&fonduri=1000.00&valuta=euro”;
</script>
```

Efectul acestui cod este de a crea un tag de imagine dinamic în HTML şi să seteze sursa ca fiind cea a transferului de fonduri din aplicaţia vulnerabilă a băncii. Browser-ele clienţilor autentificaţi pe pagina băncii respective, care accesează pagina atacatorului, o să execute codul JavaScript al acestuia şi o să creeze în fundal o cerere HTTP GET legată la sursă imaginii dinamice iar acţiunea va fi executată ca şi cum utilizatorul ar fi făcut-o în mod voluntar.

Metode de prevenire a vulnerabilităţilor de tip Cross-site Request Forgery

a) Cookie-uri postate de două ori

Această metodă de apărare constă în introducerea unui câmp de introducere a datelor secrete care să conţină valoarea actuală a ID-ului de sesiune a utilizatorului

sau o altă valoare securizată generată aleator într-un cookie al clientului pentru orice formular folosit la transmiterea datelor sensibile. Când formularul este postat, serverul aplicaţiei va verifica dacă valoarea cookie-ului din formular coincide cu cea din antetul HTTP al cererii, în caz contrar cererea va fi ignorată ca şi invalidă şi va fi înregistrată în fișierele log ca potenţial atac. Această metodă se bazează pe faptul că atacatorul nu ştie valoarea cookie-ului de sesiune al utilizatorului, însă dacă prin altă metodă acesta reuşeşte să afle valoarea, această strategie de apărare nu va avea succes.

b) **Nonce unic pentru formular**

Este probabil cea mai folosită metodă de apărare împotrivă CSRF şi constă în construirea fiecărui formular folosind un câmp ascuns care conţine un nonce (number used once) obţinut folosind un generator pseudo-aleator de numere securizate prin criptare, pentru a nu fi vulnerabil la atac. Când serverul aplicaţiei primeşte valorile parametrilor formularului ca făcând parte dintr-o cerere HTTP POST, va compara valoarea nonce-ului cu valoarea stocată în memorie şi va ignora cererea dacă valorile acestora diferă sau dacă valoarea nonce-ului a expirat.

c) **Cererea credenţialelor de autentificare**

Această metodă le cere utilizatorilor autentificaţi să reintroducă parola corespunzătoare sesiunii în care sunt autentificaţi ori de câte ori fac o tranzacţie sensibilă. Această strategie este des întâlnită în aplicaţiile web în cadrul cărora tranzacţiile de o natură sensibilă se întâmplă rar (cel mai adesea fiind schimbări ale informaţiilor de pe profilul utilizatorului).

8. Diseminarea codului sursă

Divulgarea codului sursă este o eroare de codare, foarte des întâlnită, în aplicaţiile web, care pot fi exploatate de către un atacator pentru a obţine codul sursă şi

configurarea fişierelor prin intermediul HTTP, acest lucru oferindu-i atacatorului o înţelegere mai profundă a logicii aplicaţiei web.

Multe pagini web oferă utilizatorilor fişiere pentru download folosind pagini dinamice specializate. Când browser-ul cere pagina dinamică, mai întâi serverul execută fişierul şi apoi returnează rezultatul în browser, deci paginile dinamice sunt, de fapt, coduri executate pe serverul web. Dacă această pagină nu este codată suficient de securizat, un atacator o poate exploata pentru a descărca codul sursă şi chiar fişierele de configurare.

Folosind un atac de tip divulgarea codului sursă, atacatorul poate obţine codurile sursă pentru aplicaţiile de pe server, cum ar fi: ASP, PHP şi JSP. Obţinerea codului sursă al aplicaţiilor de pe server îi oferă atacatorului o imagine mai bună asupra logicii aplicaţiei, modul în care aplicaţia gestionează cererile şi parametrii lor, structura bazei de date, vulnerabilităţile codului şi comentariile introduse în el. Odată ce are codul sursă şi posibil un duplicat al aplicaţiei pe care să poată face teste, atacatorul se poate pregăti pentru un atac asupra aplicaţiei.

Atacul: Poate fi făcut prin mai multe metode:

- Folosind vulnerabilităţi cu divulgare a codului sursă cunoscute;
- Exploatarea unei vulnerabilităţi din aplicaţie care s-ar putea să permită divulgarea codului sursă;
- Exploatarea erorilor detaliate care uneori pot include codul sursă;
- Utilizând alte tipuri de vulnerabilităţi cunoscute care se pot dovedi utile pentru divulgarea codului sursă (cum ar fi traversarea directoarelor).

Metode de prevenire a atacurilor de tip Source Code Disclosure

a) **Verificaţi folderul de unde este cerut fişierul care urmează să fie descărcat (menţineţi un white list cu numere directoarelor de unde este permisă download-area fişierelor şi validaţi cererile pe bază acestuia).**

b) **Verificaţi tipul de fişiere care sunt cerute de utilizatori.**

c) **Indexaţi fişierele care pot fi descărcate şi afişaţi doar numărul lor din index ca şi parametru al URL-ului.**

9. Erori logice

Toate aplicaţiile web folosesc logica pentru a avea funcţionalitate. A scrie cod întrun limbaj de programare, nu înseamnă nimic altceva decât descompunerea unui proces complex în paşi mici şi simpli, pentru a aduce ceea ce este pe înţelesul oamenilor la nivelul la care poate fi executat de către computer.
Atunci când un număr mare de programatori şi designeri diferiţi lucrează în paralel la aceeaşi aplicaţie, există şanse foarte mari să apară erori. Până şi pentru dezvoltarea celor mai simple aplicaţii web este nevoie o cantitate mare de logică pentru fiecare etapă. Această logică prezintă oportunitatea pentru erori logice, iar erorile logice oferă o bază foarte mare şi variată de atac, însă de multe ori sunt trecute cu vederea deoarece rareori pot fi scanate cu programe specializate în identificarea vulnerabilităţilor, nu au o semnătură specializată ca şi vulnerabilităţile de tip SQL injection sau cross-site scripting şi sunt mai greu de recunoscut şi caracterizat.
Erorile logice apar atunci când programatorul sau dezvoltatorul aplicaţiei web nu se gândeşte la toate efectele pe care le poate avea codul asupra aplicaţiei, luând în considerare numai un anumit efect (cel pe care el intenţionează să-l implementeze în primul rând) şi omiţând alte efecte posibile (secundare). Deoarece, în general, sunt mai greu de înţeles şi nu sunt apreciate ca şi vulnerabilităţi atât de grave, erorile logice prezintă un interes foarte mare pentru atacatori.
Defectele logice sunt greu de explicat, definit şi învăţat prin teorie, astfel că în cele ce urmează sunt prezentate câteva exemple concrete.

9.1. Ocolirea funcţiei pentru schimbarea parolei

Această eroare de logică a fost descoperită într-o aplicaţie web utilizată de către o societate de servicii financiare.
Funcţionalitatea: aplicaţia constă într-o funcţie pentru schimbarea parolei de către utilizatorii finali în care trebuiau completate câmpurile: nume, parola actuală, noua parolă şi confirmarea noii parole. De asemenea, venea cu o funcţie care permitea schimbarea parolei de către administratori, prin care aceştia puteau schimba parola

oricărui utilizator fără a li se cere să introducă vechea parolă. Cele două funcţii au fost puse în aplicare în cadrul aceluiaşi script pe partea serverului.

Presupunerea: Interfeţele afişate clienţilor şi administratorilor difereau într-un singur aspect, acela că cea afişată administratorului nu avea câmpul pentru introducerea vechii parole. Aşadar, când serverul procesa o schimbare de parolă utiliza prezenţa sau absenţa vechii parole pentru a determina dacă cererea a fost făcută de un administrator sau de un utilizator obişnuit. Cu alte cuvinte, eroarea logică a constat în faptul că programatorii au presupus că utilizatorii obişnuiţi vor furniza întotdeauna vechea parolă.

```
String existingPassword = request.getParameter (“existingPassword”);
if (null == existingPassword)
{
trace("Old password not supplied, must be an administrator”); return
true;
}
else
{
Trace("Verifying usera□□s old password”;
....
}
```

Atacul: Odată ce ipoteza a fost formulată în mod explicit în acest mod, eroarea logică devine evidentă, din moment ce utilizatorii controlează fiecare aspect al cererilor pe care le emit, astfel pot alege să completeze sau nu câmpul care cere vechea parolă. Acest defect logic s-a dovedit a fi devastator pentru aplicaţie, deoarece permitea unui atacator să reseteze parola oricărui alt utilizator preluând, astfel, controlul asupra contului acestuia.

9.2. Ştergerea unei piste de audit

Acest defect logic a fost descoperit într-un centru de telefonie.

Funcţionalitatea: aplicaţia a implementat diverse funcţii care permiteau personalului de la biroul de asistenţă şi administratorilor să gestioneze o bază de date de mari dimensiuni. Multe din aceste funcţii se refereau la informaţii securizate, inclusiv crearea de conturi şi schimbarea de parole, aşadar aplicaţia a menţinut o gestiune

completă de audit, înregistrând fiecare acţiune efectuată şi identitatea utilizatorului responsabil de acea acţiune. Aplicaţia includea o funcţie care le permitea administratorilor să şteargă anumite înregistrări de audit, însă pentru a evita exploatarea în scopuri rău-voitoare, orice utilizare a funcţiei de ştergere era la rândul ei înregistrată.

Presupunerea: designerii aplicaţiei au crezut că nimeni nu poate şterge înregistrările de audit fără a lăsa măcar o urmă (această presupunere a fost testată de către administratorii lor, întotdeauna rămânând ultima înregistrare a persoanei care a şters datele).

Atacul: presupunerea designerilor a fost greşită, existând o cale de a accesa datele, a produce modificări asupra lor şi de a şterge urmele în întregime. Paşii erau următorii:

- Autentificarea cu contul propriu, şi crearea un nou cont de utilizator;
- Atribuirea tuturor privilegiilor noului cont;
- Utilizarea noului cont pentru a duce la îndeplinire atacul;
- Utilizarea unui nou cont pentru a şterge toate înregistrările generate de primele trei etape.

Fiecare din acţiuni generează înregistrări în jurnalul de audit, dar pentru că în ultima fază ultimul cont şterge toate datele legate de intrările precedente, în jurnalul de audit este indicat doar faptul că ce-l de-al doilea cont nou creat a şters toate datele. Deoarece cel care i-a dat privilegii celui de-al doilea cont nou creat este primul cont nou creat, şi cum înregistrările legate de cine a creat acest cont au fost şterse, nu există nimic care să poată lega identitatea persoanei care deţine contul care a dus la capăt atacul, de contul iniţial al persoanei care a pornit atacul.

Nu există o reţetă perfectă pentru evitarea erorilor logice, ci doar o serie de bune practici care pot ajuta în acest sens, între care se regăsesc:

***a)* Documentaţi toate aspectele legate de designul aplicaţiei suficient de detaliat pentru a putea fi uşor înţelese de cineva din exterior**

***b)* Lăsaţi comentarii în codul sursă care să includă următoarele informaţii:**

- Scopul şi funcţionalitatea fiecărei porţiuni de cod;

- Presupunerile făcute de fiecare componentă în legătură cu elementele care nu se află direct sub controlul ei;
- Adăugaţi comentarii pentru fiecare porţiune de cod care să facă trimitere la toate componentele care folosesc acel cod.

c) **În timpul recenziilor de securitate referitoare la cod, plecaţi de la ideea pe care a avut-o designerul şi mergeţi înspre modul în care ar putea influenţa utilizatorii aplicaţia.**

d) **În timpul verificării de securitate, puneţi accent pe modul în care se comportă aplicaţia în momentul în care sunt introduse date eronate şi efectele produse asupra dependenţelor şi interoperabilităţilor dintre diversele componente ale codului.**

10. Software vulnerabil din partea altor producători

Când vine vorba de aplicaţii care provin de la diferite companii, majoritatea designerilor şi proprietarilor de aplicaţii web presupun că acestea sunt sigure şi nu le mai testează înainte de implementare ceea ce poate duce la breşe grave de securitate ale aplicaţiei web. Multe aplicaţii web provenite din terţe părţi sunt nesigure şi de multe ori acestea vin cu un nume de utilizator şi parolă implicit.

Aceasta este o breşă gravă de securitate deoarece, dat fiind faptul că o parolă de administrare ”default” ghicită poate oferi acces la multiple opțiuni de configurare ale aplicației inclusiv la toate datele stocate în baza de date. Așadar parolele și conturile implicite trebuie întotdeauna schimbate.

De asemenea, aplicaţiile web, provenite din terţe surse, pot fi vulnerabile la toate atacurile specifice aplicaţiilor web în general (XSS, SQL injection, etc.)

Pentru a vă proteja de breşe de securitate, oricând adăugaţi un nou software aplicaţiei web a dumneavoastră, nu faceţi presupuneri că sunt sigure sau că au fost testate amănunţit, înainte de a fi scoase pe piaţă şi toate problemele rezolvate, ci testaţi-le dumneavoastră cât puteţi de amănunţit, pentru a vă asigura că nu veţi avea probleme mai târziu. O eroare apărută într-un program vă poate pune în pericol întreaga aplicaţie web.

Un raport al companiei Verizon pentru anul 2011 cu privire la investigarea furturilor de date arată că:

- 66% din aplicaţiile făcute de industria de software prezintă un nivel inacceptabil de scăzut al securităţii la eliberarea pe piaţă;
- 72% din produsele dedicate securităţii şi programele de service au o calitate a securităţii inacceptabilă: cele mai grave probleme au fost descoperite la programele de asistență pentru clienţi (82% inacceptabile), urmate de programele de securitate (72%);
- Dezvoltatorii au nevoie de mai mult training în legătură cu securitatea programelor: mai mult de jumătate din dezvoltatorii care au dat examenul de principii de bază ale securităţii aplicaţiilor au luat 5 sau mai puţin;
- Între programele publice şi cele private ale furnizorilor de software s-au găsit foarte puţine diferenţe;
- Industria de software se mişcă rapid pentru a remedia erorile: 90% din programe au atins nivele acceptabile de securitate în 30 de zile de la lansarea pe piaţă;
- Vulnerabilitatea la injecţiile cu cod SQL scade încet;
- Construirea de software bine securizat nu trebuie să consume mult timp.

Nu există nici o aplicaţie care este 100% lipsită de vulnerabilităţi, însă puteţi încerca să reduceţi cât mai mult aceste probleme, dar acest lucru nu se va întâmpla decât dacă testaţi amănunţit întreaga aplicaţie web pentru a descoperi punctele ei slabe şi a încerca să le remediaţi. Nu faceţi presupuneri când este vorba de securitate.

11. Detectarea vulnerabilităților specifice aplicațiilor web

Plaja de vulnerabilități specifice aplicațiilor web este foarte vastă, cele prezentate în cadrul acestui ghid fiind totuși cel mai des întâlnite. După cum am mai menționat, tehnicile prezentate sunt simpliste, dar scopul acestui ghid este de a oferi noțiunile generale despre problemele aplicațiilor web.
O etapă esențială pentru orice dezvoltator de aplicații web este identificarea acestor vulnerabilități și tratarea acestora corespunzător. În acest sens, există numeroase

instrumente software automate pentru detectarea vulnerabilităților, care, de cele mai multe ori, dau și recomandări pentru rezolvarea problemelor identificate.

Scanarea vulnerabilităților este o etapă de bază, ce trebuie realizată înainte de punerea în producție a oricărei aplicații web.

NESSUS este un scanner de vulnerabilități ce cuprinde și modul specific aplicațiilor web. Așadar o scanare cu această unealtă poate identifica și vulnerabilitățile de la nivelul rețelei interne. Scannerul poate fi folosit și de către administratorii de rețea, pentru identificarea resurselor interne (echipamente etc.) ale unui LAN. Acesta va identifica echipamentele din cadrul rețelei alături de lista vulnerabilităților specifice pentru fiecare.

Nessus are și o variantă free și poate fi descărcat de la adresa http://www.tenable.com/products/nessus.

OpenVas este altă aplicație de management al vulnerabilităților. Aceasta este opensource și se trage din vechea versiunea gratuită a NESSUS (înainte de a fi cumparat de Teenable). Produsul poate fi descărcat de la adresa http://www.openvas.org/

Dacă se consideră că o versiune gratuită nu este de ajuns, sunt disponibile și unelte comerciale de scanare, specifice mediului web, precum HP WEB Inspect sau Acunetix.

Bibliografie

Acest material a fost realizat prin documentare din următoarele surse:

1. OWASP Top 10 Most Critical Web Application Security Risks, disponibil la https://www.owasp.org/index.php/Top_10
2. OWASP Application Security Verification Standard 2009, disponibil la https://www.owasp.org/images/4/4e/OWASP_ASVS_2009_Web_App_Std_Release.pdf
3. TrustWave Global Security Report 2011, disponibil la https://www.trustwave.com/downloads/Trustwave_WP_Global_Security_Report_2011.pdf?u=6JY2rq4LWNktaCGRVZPjTn1SL1A3PJh3Apv8WGjZV5TcRZa96rB6W2nPb5grXHUYewwAmZ0pb0296kKCaL8hXEtf9T6