Securitatea aplicațiilor web după modelul OWASP

Rodica BULAI, Alexandr ZIAEV
Technical University of Moldova
rodica.bulai@mail.utm.md, alexandr.ziaev@ati.utm.md

***Abstract* — Aplicațiile web au devenit o vulnerabilitate primară pentru afacerile moderne. Astăzi, activitățile fiecărei întreprinderi depinde de tehnologiile web și Cloud. Potrivit Gartner, 80% din atacuri vizează aplicațiile web. Adesea, aceste atacuri exploatează vulnerabilitățile atenuate în codul aplicației. Cele mai frecvente probleme găsite în codul nativ fiind log forging și cross-site scripting. Vulnerabilitatea aplicațiilor web se datorează mai multor factori: securitatea scăzută a arhitecturii web, care permite ca malware-ul să fie instalat și executat în interiorul browser-ului; creșterea complexității, care ascunde bug-urile; deoarece codul devine mai complex și conține o sintaxă mai obscură, ceea ce face mai dificilă detectarea vulnerabilităților; accesibilitatea publică a internetului, care permite atacatorilor externi să creeze o cale către datele confidențiale și mediile IT ale unei companii.**

I. INTRODUCERE

Fiecare piață de tehnologii are nevoie de o sursă imparțială de informații privind cele mai bune practici, precum și de o comunitate activă care pledează pentru standarde deschise. În spațiul de securitate web, unul dintre aceste grupuri este Proiectul Open Web Application Security Project (OWASP).

Operând ca o comunitate de profesioniști, OWASP are o poziție unică de a furniza informații, practici, instrmente, standarde și recomanadri privind securitatea aplicațiilor web persoanelor, companiilor, universităților, agențiilor guvernamentale și altor organizații din întreaga lume[1].

II. MODELUL OWASP

La fiecare 4 ani, OWAP prezintă un proiect ce include top 10 cele mai critice vulnerabilități pentru securitatea aplicațiilor web, sursele și cauzele apariției lor, precum și modalitățile de securizare contra acestor amenințări [2].

1.**Injection.** Fragmentele de injectare apar atunci când o aplicație trimite date nesigure către un interpret. Fragmentele de injectare sunt foarte răspândite, în special în codul vechi, adesea găsite în interogări SQL, interogări LDAP, interogări Xpath, comenzi OS, argumente de program etc.

Defectele de injecție sunt ușor de descoperit când se examinează codul, dar mai dificil prin testare. Una din opțiunile de prevenire este utilizarea API-rilor sigure, care să evite în totalmente utilizarea interpretului sau care oferă o interfață parametrizată. Daca nu este disponibil un API cu parametri, se recomandă să fie evitate cu atenție caracterele speciale folosind sintaxa specifică de evadare.

2. **Broken authentication and session management.** Dezvoltatorii construiesc frecvent scheme personalizate de autentificare și de gestionare a sesiunilor. Drept urmare aceste scheme personalizate au în mod frecvent defecte cum ar fi logout, gestionarea parolei, timeouts, întrebare secretă, actualizare cont etc. Descoperirea unor astfel de defecțiuni pot fi uneori dificile deoarece fiecare implementare este unică.

Prevenirile acestui risc ar fi, un singur set de controale puternice de autentificare și gestionare a sesiunilor. Acestea controale trebuie să urmărească:

a. îndeplinirea tuturor cerințelor de autentificare și de gestionare a sesiunilor definite în domeniile V2(Authentification) și V3(Managementul sesiunilor) ale OWASP.

b. utilizarea unei interfețe simple pentru dezvoltatori, cum ar fi aplicațiile ESAPI Authentificator și API-urile utilizator.

3. **Cross Site Scripting (XSS).** Este considerată cea mai răspândită vulnerabilitate de securitate a aplicațiilor web. Fragmente XSS apar atunci când o aplicație include date furnizate de utilizator întro-o pagină trimisă browserului fără a valida corespunzător acel conținut. Identificarea vulnerabilităților Server XSS este destul de ușoară prin testarea sau analiza codului. Opțiunea preferata este să scapi în mod corespunzător de toate datele care nu sunt de încredere, bazate pe contextul HTML (atribut, javascript, css, url).

4. **Insecure Direct Object References.** Aplicațiile utilizează frecvent numele sau cheia reală a unui obiect atunci când generează pagini web. Aplicațiile nu verifică întodeauna dacă utilizatorul este autorizat pentru obiectul țintă. Aceasta are ca rezultat o defecțiune de referință direct a obiectelor nesigure. Testele pot manipula cu ușurință valorile parametrilor pentru a defecta astfel de defecțiuni. Analiza codului arată rapid dacă autorizația este corect verificată.

Pentru a preveni riscul, se verifică accesul. Fiecare

utilizare a unei referințe directe la un obiect dintr-o sursă neîncrezătoare trebuie să includă o verificare a controlului accesului pentru a se asigura că utilizatorul este autorizat pentru obiectul solicitat[3].

5. **Security Misconfiguration.** Configurația greșită a securității se poate întâmpla la orice nivel al unui pachet de aplicații, inclusiv platforma, serverul web, serverul de aplicații, baza de date, cadrul și codul personalizat.

Dezvoltatorii și administratorii de sistem trebuie să colaboreze pentru a se asigura că înreaga stivă este configurată corespunzător. Scanerele automate sunt utile pentru detectarea patch-urilor lipsă, confruntări greșite, utilizarea conturilor implicite, servicii inutile etc.

Prevenirea acestei amenințări poate fi realizată prin utilizarea unei arhitecturi puternice de aplicații care asigură o separare eficientă și sigură între componente. Se recomandă, de asemenea, luarea în considerare a rulării scanărilor și efectuarea periodică de audit pentru a ajuta la detectarea eventualelor confruntări greșite sau a lipsei de patch-uri.

6. **Sensitive Data Exposure.** Cel mai mare neajuns este pur și simplu să nu criptezi datele sensibile. Când se utilizează criptarea, generarea și gestionarea slabă a cheilor, precum și folosirea slabă a algoritmilor sunt comune, în special tehnicile de hashing cu parolă slab compusă. Slăbiciunile browserului sunt foarte frecvente și ușor de detectat, dar greu de exploatat pe scară largă. Atacatorii externi au dificultăți în detectarea defectelor de pe server datorită accesului limitat.

Un mod de prevenire ar fi să nu fie stocate inutil datele sensibile. Să fie șterse cât mai curînd posibil. Datele pe care nu le aveți nu pot fi furate. Asigurativă că sunt utilizați algoritmi puternici de criptare și că sunt folosite chei puternice, cu un management adecvat al acestora. Luați în considerare utilizarea modulelor criptografice validate FIPS 140. Asigurativă că parolele sunt stocate cu un algoritm special conceput pentru protecția prin parolă, cum ar fi bcrypt, PBKDF2 sau scrypt.

7. **Mising Function Level Acces Control.** Aplicațiile nu protejează întodeauna funcțiile aplicației în mod corespunzător. Uneori protecția la nivel de funcție este gestionată prin configurație, iar sistemul este configurat greșit. Uneori, dezvoltatorii trebuie să includă verificările corespunzătoare ale codului. Detectarea unor astfel de defecțiuni este ușoară. Partea cea mai grea este identificarea paginilor (URL – urilor) sau a funcțiilor care există pentru a ataca.

Pentru a asigura protecția împotriva unor astfel de amenințări este nevoie ca mecanismul (dispozitivele) de executare să refuze accesul în mod implicit, necesitând subvenții explicite pentru anumite roluri de acces la fiecare funcție. Dacă funcția este implicată întru-un flux de lucru, să se verifice dacă condițiile sunt corespunzătoare pentru a permite accesul.

8. **Cross-Site Request Forgery (CSRF).** CSRF profită de faptul că majoritatea aplicațiilor web permit atacatorilor să prezică toate detaliile unei anumite acțiuni. Deoarece browserele trimit automat acreditări precum cookie-urile de sesiune, atacatorii pot crea pagini web rău intenționate, care generează cereri forțate, care nu pot fi distinse de cele legitime. Detectarea defectelor CSRF este destul de ușoară prin testarea penetrării sau prin analiza codului.

Opțiunea recomandată este să includeți un token unic întru-un câmp ascuns. Acest lucru determină trimiterea valorii în corpul solicitării HTTP, evitând includerea acesteia în URL. Solicitarea ca utilizatorul să se re-autentifice sau să demonstreze că nu este un robot (de exemplu, printr-un CAPTCHA) poate, de asemenea să protejeze împotriva CSRF.

9. **Using Components with Known Vulnerabilities.** Practic, fiecare aplicație are aceste probleme, deoarece majoritatea echipelor de dezvoltare nu se concentrează pe asigurarea actualității componentelor/bibliotecilor. În multe cazuri, dezvoltatorii nu cunosc nici măcar toate componentele pe care le folosesc, nu contează versiunile lor. Componentele dependente fac lucrurile și mai rele.

Pentru prevenire se recomandă monitorizarea securizării acestor componente în bazele de date publice, listele de corespondență a proiectelor și listele de corespondență de securitate, menținerea actualizărilor. Să fie create politici de securitate care guvernează utilizarea componentelor, cum ar fi solicitarea anumitor practici de dezvoltare a software-ului, trecerea testelor de securitate și licențe acceptabile.

10. **Unvalidated Redirects and Forwards.** Aplicațiile frecvent redirecționează utilizatorii către alte pagini sau utilizează funcțiile interne înainte într-un mod similar. Uneori, pagina țintă este specificată într-un parametru nevalidat, permițând atacatorilor să aleagă pagina de destinație. Detectarea redirecționărilor necontrolate este ușoară. Căutați redirecționări unde puteți seta adresa URL completă. Verificările nu sunt ușoare, deoarece vizează paginile interne.

Ca recomandări de prevenire poate servi evitarea utilizării redirecționărilor. Dacă parametri de destinație nu pot fi evitați, să se asigure că valoarea furnizată este validă și autorizată pentru utilizator.

III. CONCLUZII

Vulnerabilitățile pe scară largă necesită o transformare a modului în care securitatea aplicațiilor este concepută și a modului în care dezvoltarea de software implică soluțiile de securitate, de la definirea cerințelor până la implementare și operațiuni.

BIBLIOGRAFIE:

[1] Despre Open Web Application Security Project, https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project

[2] Noi orientări pentru scrierea codului securizat https://opensource.com/business/14/5/new-guidelines-writing-secure-code

[3] Cele 10 riscuri majore pentru securitatea app web hhttps://www.checkmarx.com/glossary/owasp-top-10/)