Universitatea Tehnică a Moldovei

Aplicație pentru analiza şi depistarea vulnerabilităților populare platformei CMS WordPress

Penetration test tool for CMS WordPress

Student: **Dumbrava Vladislav**

Conducător: lector superior Antohi Ionel

Chişinău 2017

Ministerul Educaţiei al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică şi Microelectronică
Departamentul Ingineria Software şi Automatică

Admis la susţinere
Şef departament: dr.conf.univ. Ciorbă D.
_____
„__”_____ 2017

Aplicație pentru analiza şi depistarea vulnerabilităților populare platformei CMS WordPress

Proiect de licenţă

Student: _____ (V. Dumbrava)
Conducător: _____ (I. Antohi)
Consultanţi: _____ (I. Antohi)
_____ (G.Covdii)

Chişinău 2017

Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică şi Microelectronică
Departamentul Ingineria Software şi Automatică
Specialitatea Securitate Informaţională

Aprob
dr.conf.univ. Dumitru Ciorbă
şef departament
_____
„__”_____ 2017

CAIET DE SARCINI
pentru proiectul de licenţă al studentului
Dumbrava Vladislav

1. Tema proiectului de licenţă: Aplicație pentru analiza şi depistarea vulnerabilităților populare platformei CMS WordPress

confirmată prin hotărârea Consiliului facultăţii de la „ *24*” *octombrie 2016*

2. Termenul limită de prezentare a proiectului *31.05.2017*

3. Date iniţiale pentru elaborarea proiectului *Sarcina pentru elaborarea proiectului de diplomă*

4. Conţinutul memoriului explicativ

Introducere
1. Analiza domeniului de studiu
2. Proiectarea sistemului
3. Realizarea sistemului
4. Documentarea produsului realizat
5. Argumentarea economică
Concluzii

5. Conţinutul părţii grafice a proiectului
Diagrame Use-Case, Interfața principală a programului

6. Lista consultanţilor:

Consultant	Capitol	Confirmarea realizării activităţii	
		Semnătura consultantului (data)	Semnătura studentului (data)
G. Covdii	*Argumentarea economică*		
I. Antohi	*Controlul calităţii* *Standarde tehnologice*		

7. Data înmânării caietului de sarcini *01.09.2016*

Conducător _____

semnătura

Sarcina a fost luată pentru a fi executată

de către studentul _____ *01.09.2016*

semnătura, data

PLAN CALENDARISTIC

Nr. crt.	Denumirea etapelor de proiectare	Termenul de realizare a	Nota
1	*Elaborarea sarcinii, primirea datelor pentru sarcină*	*01.09.16– 30.09.16*	*10%*
2	*Studierea literaturii de domeniu*	*01.10.16– 30.11.16*	*20%*
3	*Alegerea şi pregătirea de lucru a softului*	*01.12.16 – 25.12.16*	*20%*
4	*Realizarea programului*	*16.01.17 – 30.04.17*	*25%*
5	*Descrierea programului, diagramele UML*	*01.05.17 – 15.05.17*	*10%*
6	*Testarea aplicaţiei*	*16.05.17– 28.05.17*	*10%*
7	*Finisarea proiectului*	*29.05.17– 31.05.17*	*5%*

Student _____ *Dumbrava Vladislav ( )*

Conducător de proiect *Antohi Ionel ( )*

Cuprinsul

1. Introducerea……..,3

2. Conținutul……..4

2.1. WordPress security – WPScan pentru WordPress vulnerabilitate scanner…….4

2.2. WordPress și securitatea……...8

2.2.1 Echipa de securitate a WordPress……...8

2.2.2 Riscuri, procese și istoria securității WordPress…….8

2.2.3 Actualizarea automată în culise a versiunilor de Securitate……9

2.2.4 Top 10 2013, OWASP……9

2.2.5 Alte riscuri de securitate și motive de îngrijorare……..…11

2.3. Securitatea modulelor și temelor WordPress …….…...13

2.3.1. Tema implicită……...13

2.3.1. Depozitarele de module și teme de la WordPress.org……...13 2.3.2 Echipa de revizuire a temelor…….13

2.3.3.1 O notă despre WordPress.com și securitatea WordPress ……14

2.4. Statistica…….15

2.5. XSS……16

2.5.1. Diagrama Use-Case…….16

2.5.2 Diagrama de Activitate…….…17

3.Concluzii……...18

4. Bibliografia……..…19

Introducere

WordPress este o platformă de tip sursă deschisă pentru publicarea blogurilor. Platforma WordPress este scrisă în limbajul PHP, folosind pentru gestionarea bazelor de date sistemul MySQL. Dispune un sistem de șabloane scrise în limbajele HTML și CSS. Avantajele majore prezentate de WordPress sunt simplitatea și numeroasele plugin-uri create de către comunitate care pot modifica funcționalitatea WordPress-ului tranformându-l în aproape orice tip de site web. De asemenea interfața poate fi schimbată foarte ușor cu ajutorul multitudinii de teme gratuite sau premium cu doar un clic.

WordPress a apărut prima dată în 2003 ca precursor al b2/cafelog, care era utilizat de 2 000 de bloguri la acea vreme.

De la începuturi și până acum, WordPress a fost etichetat în nenumărate rânduri ca fiind vulnerabil la atacuri ce vizau adăugare de conținut malițios pe blogurile sau site-urile ce foloseau această platformă. Au fost create numeroase programe automate ce căutau pe internet site-uri care foloseau această platformă și, folosindu-se de vulnerabilitățile acesteia, reușeau să introducă în conținut, fără acordul proprietarului, link-uri către site-urile celor ce foloseau aceste programe.

În ultimul timp însă, dezvoltatorii platformei au reușit să aducă WordPress într-o stare în care atacurile de acest tip sunt aproape imposibil de realizat. În momentul de față WordPress este printre cele mai sigure si mai puternice CMS-uri la ora actuală.

2. Conținutul

2.1 WordPress security – WPScan pentru WordPress vulnerabilitate scanner

WP scanare este o scanare de vulnerabilităţi pentru WordPress. Acesta este dezvoltat în Ruby. El este capabil să lista de plugin-uri utilizate şi vă va oferi vulnerabilităţi de securitate asociate. De asemenea, include un modul de forţă brută pentru a aborda interfaţa de administrare WordPress.

Este important să reţineţi că, în ceea ce priveşte majoritatea securitate şi instrumente de scanare, WPscan nu va asigura nu WordPress pentru tine. De asemenea, completarea unui control de securitate cu WPscan care nu este vizibil defecte ai nu înseamnă că WordPress este de a 100% sigur. Este o noţiune care trebuie să fie constant în minte atunci când vorbim despre securitate.

WP scanare este nativ pe distribuţii următoarele : (Windows nu este acceptată)

-BackBox Linux -

Kali Linux -Pentoo

Dacă doriţi să instalaţi manual pe Debian/Ubuntu sau Fedora/CentOS, Iată premisele :

Pe Debian :

```
sudo apt-get install git ruby ruby-dev libcurl4-gnutls-dev make
```

```
git clone https://github.com/wpscanteam/wpscan.git
```

```
cd wpscan
```

```
sudo gem install bundler
```

```
bundle install --without test --path vendor/bundle
```

Pe distribuţii Fedora/CentOS/RHEL :

```
sudo yum install gcc ruby-devel libxml2 libxml2-devel libxslt libxslt-devel libcurl-devel
```

```
git clone https://github.com/wpscanteam/wpscan.git
```

```
cd wpscan
```

```
sudo gem install bundler && bundle install --without test
```

```
ruby wpscan.rb votrecommande ....
```

Putem folosi acum WPscan.

Înainte de orice analiză a actualiza baza de date a wp-scanare. Acest lucru este important, deoarece dacă unul decide la spre a scanda nostru WordPress cu o bază de date de vulnerabilităţi, care nu este până la data, unele vulnerabilitati de securitate legate de tema
WordPress versiunea sau versiunile de plug-in-uri de exemplu, nu ai nu va transporta, există un risc sa pierd critice pentru a asigura. Pentru a actualiza baza de date WPscan :

```
wpscan --update
```

Acum, că baza noastră de date este actualizată, putem începe să scaneze WordPress site-ul nostru :

```
wpscan --URL www.monsite.fr
```

Cu aceasta comanda va şti versiune de WordPress, numele şablonului, lista de plugin-uri...

Atunci când aveţi un semn de exclamare roşu, acest lucru înseamnă că este important pentru a corecta eroarea. Fi actualizarea WordPress în cazul în care plugin-uri fie prin ştergerea fişierului. Majoritatea dintre noi, să fişierul "readme.html". Ştergeţi-l, acesta oferă informaţiile din versiunea WP.

Alte ordine care pot fi interesante si permite scanarea complet un blog WordPress listarea utilizatori, plugin-uri vulnerabile, vulnerabile teme cunoscute...

```
wpscan --url www.monsite.fr --enumerate
```

Aveţi posibilitatea să rafinaţi detectarea de vulnerabilităţi prin specificarea o opţiune suplimentară de exemplu doar plugin-uri vulnerabile (-enumera vp)utilizatorii (-enumera u), … Puteţi vizualiza detalii despre opţiunile la această adresă : http://wpscan.org

Poti face un test al tău password cu următoarea comandă :

```
wpscan --URL www.monsite.fr --wordlist /chemindevotrelistebruteforce.txt --username votreutilisateur
```

Exemplu: listare versiune, thema instalata, pluginuri, vulnerabiliti cunoscute precum si userii existenti.

[root@hacklab /home/admin/tmp/lab/wpscan]#ruby wpscan.rb –url www.crystalmind.ro/wordpress –enumerate u

```
_____________________________________________________________
        __          _______   _____
        \ \        / /__  \ / ____|
         \ \  /\  / /  | |__) | (___   ___  __ _ _ __
          \ \/  \/ /   | |___/ \___ \ / __|/ _` | '_ \
           \  /\  /    | |       ____) | (__| (_| | | | |
            \/  \/     |_|      |_____/ \___|\__,_|_| |_|

      WordPress Security Scanner by the WPScan Team
                  Version 2.8
       Sponsored by Sucuri – https://sucuri.net
  @_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_
_____________________________________________________________
```

[+] URL: http://www.crystalmind.ro/wordpress/

[+] Started: Fri Sep 25 15:47:23 2015

[!] The WordPress 'http://www.crystalmind.ro/wordpress/readme.html' file exists exposing a version number
[+] Interesting header: SERVER: Apache

[+] Interesting header: X-POWERED-BY: PHP/5.3.10-1ubuntu3.19

[+] XML-RPC Interface available under: http://www.crystalmind.ro/wordpress/xmlrpc.php

[+] WordPress version 4.3.1 identified from meta generator

[+] WordPress theme in use: magazine-basic – v3.0.6

[+] Name: magazine-basic – v3.0.6

| Latest version: 3.0.6 (up to date)
| Location: http://www.crystalmind.ro/wordpress/wp-content/themes/magazine-basic/
| Readme: http://www.crystalmind.ro/wordpress/wp-content/themes/magazine-basic/readme.txt
| Style URL: http://www.crystalmind.ro/wordpress/wp-content/themes/magazine-basic/style.css
| Theme Name: Magazine Basic
| Theme URI: https://themes.bavotasan.com/themes/magazine-basic-wordpress-theme/
| Description: Create a truly unique design with Magazine Basic, a lightweight and fully responsive HTML5 theme
…
| Author: c.bavota
| Author URI: http://bavotasan.com

[!] Title: Magazine Basic – wp-content/themes/magazine-basic/view_artist.php id Parameter SQL Injection
Reference: https://wpvulndb.com/vulnerabilities/7521
Reference: http://packetstormsecurity.com/files/118321/
Reference: http://www.securityfocus.com/bid/56664/
Reference: http://osvdb.org/show/osvdb/87838

[+] Enumerating plugins from passive detection …

| 7 plugins found:

[+] Name: addthis – v5.1.2

| Latest version: 5.1.2 (up to date)
| Location: http://www.crystalmind.ro/wordpress/wp-content/plugins/addthis/
| Readme: http://www.crystalmind.ro/wordpress/wp-content/plugins/addthis/readme.txt

[+] Name: captcha – v4.1.5
| Latest version: 4.1.5 (up to date)
| Location: http://www.crystalmind.ro/wordpress/wp-content/plugins/captcha/
| Readme: http://www.crystalmind.ro/wordpress/wp-content/plugins/captcha/readme.txt

[+] Name: fancier-author-box – v1.4 |
Latest version: 1.4 (up to date)

| Location: http://www.crystalmind.ro/wordpress/wp-content/plugins/fancier-author-box/
| Readme: http://www.crystalmind.ro/wordpress/wp-content/plugins/fancier-author-box/readme.txt

[+] Name: mailchimp-for-wp – v2.3.12

| Latest version: 2.3.12 (up to date)
| Location: http://www.crystalmind.ro/wordpress/wp-content/plugins/mailchimp-for-wp/
| Readme: http://www.crystalmind.ro/wordpress/wp-content/plugins/mailchimp-for-wp/readme.txt

[+] Name: popup-maker – v1.3.7
| Latest version: 1.3.7 (up to date)
| Location: http://www.crystalmind.ro/wordpress/wp-content/plugins/popup-maker/
| Readme: http://www.crystalmind.ro/wordpress/wp-content/plugins/popup-maker/readme.txt

[+] Name: social-share-boost – v4.4 |
Latest version: 4.4 (up to date)

| Location: http://www.crystalmind.ro/wordpress/wp-content/plugins/social-share-boost/
| Readme: http://www.crystalmind.ro/wordpress/wp-content/plugins/social-share-boost/readme.txt

[+] Name: yet-another-related-posts-plugin – v4.2.5

| Latest version: 4.2.5 (up to date)
| Location: http://www.crystalmind.ro/wordpress/wp-content/plugins/yet-another-related-posts-plugin/
| Readme: http://www.crystalmind.ro/wordpress/wp-content/plugins/yet-another-related-posts-plugin/readme.txt

[+] Enumerating usernames …

[+] Identified the following 4 user/s:

```
+----+----------+-----------------------------------------+
| Id | Login    | Name                                    |
+----+----------+-----------------------------------------+
| 1  | admin1 |                                         |
| 2  | admin2   | Octavian, Author at Crystal Mind Academy |
| 3  | crystalmind  | Teacher, Author at Crystal Mind Academy |
| 4  | John   | John Doe, Author at Crystal Mind Academy |
+----+----------+-----------------------------------------+
```

[+] Finished: Fri Sep 25 15:48:08 2015

[+] Requests Done: 119

[+] Memory used: 75.609 MB

[+] Elapsed time: 00:00:45

Atentie la faptul de a nu folosi un astfel de instrument pe site-urile web care nu vă sau fără consimţământul proprietarului lor. Există mai multe alte opţiuni pentru utilizarea de WPscan, dar aici am vazut funcţionării sale globale.

Este important să ştiţi cum să lucreze WPscan şi instrumente similare pentru a înţelege că anumite reguli de securitate (şi toate) permite un sigur siguranţă 100%. Înţeleagă mai bine instrumente şi strategii de atacatori pentru a apăra mai bine.

2.2 Documentul alb al securității WordPress

2.2 WordPress și securitatea

1.1 Echipa de securitate a WordPress

Echipa de securitate WordPress e formată din aproximativ 25 de experți incluzând dezvoltatorii șefi și cercetătorii în securitate — aproximativ jumătate din ei angajați ai Automattic (realizatorii lui WordPress.com, prima și cea mai mare platformă de găzduire WordPress de pe web) și un număr care lucrează în domeniul securității pe web. Echipa se consultă cu cercetători de încredere renumiți și cu companii de găzduire[3].

Echipa de securitate WordPress colaborează adesea cu alte echipe de securitate pentru a rezolva probleme de interdependență, cum ar fi rezolvarea vulnerabilității din interpretorul XML al PHP, folosit de API-ul XML-RPC livrat cu WordPress,

în WordPress 3.9.2[4]. Rezolvarea acestei vulnerabilități a fost rezultatul unui efort comun al echipelor de securitate ale WordPress și Drupal.

1.2 Riscuri, procese și istoria securității WordPress

Echipa de securitate WordPress crede în devoalarea responsabilă prin alertarea imediată a echipei de securitate cu privire la orice potențială vulnerabilitate. Potențialele vulnerabilități de securitate pot fi semnalate echipei de securitate direct la adresa de email: security@wordpress.org[5]. Echipa de securitate comunică între membrii săi pe o listă privată de email și lucrează pe un Trac privat dar neînchis (*walled-off*) pentru urmărirea, testarea și rezolvarea erorilor și problemelor de securitate.

Fiecare raport de securitate este confirmat în momentul primirii și echipa lucrează să verifice vulnerabilitatea și să determine gravitatea ei. Dacă e confirmată, echipa de securitate face planificarea

unei reparații (*patch*) ce poate fi asignată unei viitoare versiuni de WordPress sau poate fi imediat lansată o versiune de securitate, în funcție de gravitatea problemei.

Pentru o versiune imediată de securitate, echipa de securitate va publica o recomandare pe situl de știri WordPress.org *News*[6] anunțând lansarea și detaliind modificările. În acest mesaj persoana responsabilă de descoperire va primi o recunoaștere, încurajându-se astfel pe viitor continuarea raportărilor responsabile.

Administratorii WordPress văd în panoul de control al siturilor lor o notificare de actualizare când o nouă versiune e disponibilă, iar mergând pe actualizarea manuală utilizatorii sunt redirectați la ecranul Despre WordPress care detaliază modificările. Dacă administratorii au validată actualizarea automată a noilor versiuni în culise, vor primi un email după ce actualizarea s-a terminat.

1.3 Actualizarea automată în culise a versiunilor de securitate

Începând de la versiunea 3.7, WordPress a introdus actualizarea automată în culise pentru toate versiunile minore[7], cum ar fi 3.7.1 și 3.7.2. Echipa de securitate WordPress poate identifica, repara și împinge automat îmbunătățirile de securitate pentru WordPress fără ca proprietarul sitului să trebuiască să facă ceva din partea sa, actualizarea de securitate instalându-se automat.

Când este publicată o actualizare de securitate pentru versiunea curentă stabilă de WordPress, echipa nucleului va împinge de asemenea actualizările de securitate pentru toate versiunile ce sunt capabile de actualizare în culise (de la WordPress 3.7), astfel că aceste mai vechi dar încă recente versiuni de WordPress să primească și ele aceste îmbunătățiri de securitate.

Proprietarii individuali de situri pot opta pentru înlăturarea actualizărilor automate în culise printr-o simplă modificare în fișierul lor de configurare, dar păstrarea acestei funcționalități este recomandată cu tărie de echipa de bază, precum și cea de a rula ultima versiune stabilă de WordPress.

1.4 Top 10 2013, OWASP

Proiectul de securitate a aplicațiilor web deschise (*Open Web Application Security Project*, OWASP) este o comunitate internet dedicată securității aplicațiilor pentru web. Lista Top10 OWASP[8] se focalizează pe identificarea celor mai serioase riscuri de securitate ale aplicațiilor pentru un larg spectru de organizații. Cele din Top 10 sunt selectate și prioritizate printr-o combinație a estimărilor consensuale privind exploatarea, detectarea și impactul acestora.

Următoarele secțiuni pun în discuție API-urile, resursele și politicile pe care WordPress le folosește pentru a întări softul de bază dar și modulele și temele terților în fața potențialelor riscuri.

1.4.1 A1 – Injecție

Există un set de funcții și API-uri disponibile în WordPress pentru a-i sprijini pe dezvoltatori în a se asigura că injecția de cod neautorizat nu e posibilă, și în ai ajuta să valideze și igienizeze datele. Sunt disponibile documentații și bune

practici[9] despre utilizarea acestor API-uri pentru a proteja, valida, sau igieniza datele de intrare sau ieșire în HTML, URL-uri, antete HTTP și la interacțiunea cu baza de date ori sistemul de fișiere. Administratorii pot în plus să restricționeze tipurile de fișiere ce pot fi încărcate via filtre.

1.4.2 A2 – Autentificare defectuoasă și gestiunea sesiunilor

Softul de bază WordPress gestionează conturile de utilizator și detaliile de autentificare cum ar fi ID-ul utilizatorului, numele și parola pe partea serverului, la fel și cookie-urile de autentificare. Parolele sunt protejate în baza de date folosind tehnici standard de „sărare și întindere” *salting&stretching*. Sesiunile existente sunt distruse după deautentificare începând cu versiunea WordPress 4.0.

1.4.3 A3 – Scripturi inter-situri (XSS)

WordPress asigură o gamă de funcții ce pot ajuta la punerea în siguranță a datelor furnizate de utilizatori[10]. Utilizatorii de încredere, administratorii sau editorii într-o instalare singulară de WordPress, sau doar administratorii de situri în WordPress multi-sit, pot publica HTML sau JavaScript nefiltrat după necesități, cum ar fi în interiorul unui articol sau pagini. Ceilalți utilizatori și elementele de conținut trimise spre publicare sunt implicit filtrate pentru a fi înlăturate entitățile periculoase, folosind biblioteca KSES prin funcția wp_kses.

Ca exemplu, echipa nucleului WordPress a observat că

funcția the_search_query() a fost folosită greșit de majoritatea autorilor de teme, care n-au filtrat pentru siguranță (*escaping*) rezultatul ei pentru a fi folosit în HTML. Într-o foarte rară rupere a compatibilității înapoi, ieșirea funcției a fost schimbată în WordPress 2.3 pentru a fi pre-filtrată pentru securitate (*pre-escaped*).

1.4.4 A4 – Referință directă nesecurizată la obiect

WordPress furnizează destul de des referințe directe către obiecte, cum ar fi identificatori numerici unici de conturi utilizator sau conținut disponibil în URL sau câmpuri de formulare. Cu toate că acești identificatori dezvăluie informații direct din sistem, elaboratul sistem de control al accesului pe care îl are WordPress împiedică cererile neautorizate.

1.4.5 A5 – Configurarea greșită a securității

Majoritatea operațiunilor de configurare a securității WordPress sunt limitate la un singur administrator autorizat. Setările implicite ale WordPress sunt evaluate în mod continuu la nivelul echipei de bază, și aceasta furnizează documentații și exemple de bună practică pentru a întări securitatea configurării serverelor ce rulează situri WordPress[11].

1.4.6 A6 – Expunerea datelor sensibile

Parolele de acces ale utilizatorilor sunt „sărate și secționate” (*salted and hashed*) pe baza cadrului de tratare a parolelor din PHP-ul portabil *Portable PHP Password Hashing Framework*[12]. Sistemul de permisiuni WordPress este folosit pentru a controla accesul la informații private cum ar fi cele de identificare a utilizatorilor, adrese de email ale comentatorilor, conținut publicat privat, etc. În WordPress 3.7, a fost inclus în softul de bază un evaluator de tărie a parolei care să ofere informații suplimentare utilizatorilor care-și stabilesc parola și sfaturi pentru a o întări.WordPress are de asemenea o setare opțională de configurare pentru cerințe HTTPS.

1.4.7 A7 – Lipsă de control a accesului la nivel de funcții

WordPress verifică autorizarea corespunzătoare și permisiunile pentru orice cereri de acces la nivel de funcție înainte de a fi execută acțiunea. Accesul sau vizualizarea unor URL-uri administrative,

meniuri și pagini, fără autentificare corespunzătoare este integrată strâns cu sistemul de autentificare pentru a preveni accesul unor utilizatori neautorizați.

1.4.8 A8 – Falsificare cereri inter-situri (CSRF – Cross Site Request Forgery*)*

WordPress folosește garanți (*tokens*) criptografici, numiți nunici *nonces*[13], pentru a valida intenția unor cereri de acțiuni din partea unor utilizatori autorizați pentru a asigura protecția împotriva potențialelor amenințări CSRF. WordPress oferă un API pentru generarea acestor garanți pentru crearea și verificarea temporară a unicității acestora, ei fiind limitați la un anumit utilizator, o acțiune specifică, un obiect specific, o perioadă specifică de timp și care pot fi adăugați unor formulare și URL-uri după nevoie. În plus, toți nunicii sunt invalidați la deautentificare.

1.4.9 A9 – Folosirea componentelor cu vulnerabilități cunoscute

Echipa de bază WordPress monitorizează îndeaproape cele câteva biblioteci incluse și cadre de lucru (*frameworks*) integrate în WordPress pentru funcționalitățile de bază. În trecut echipa de bază a contribuit la mai multe terțe componente pentru a le face mai sigure, cum a fost și actualizarea pentru repararea unei vulnerabilități inter-sit în TinyMCE în WordPress 3.5.2[14].

Dacă e nevoie, echipa de bază poate decide să ramifice (*fork*) sau să înlocuiască componente externe critice, cum s-a întâmplat când a fost înlocuită oficial biblioteca *SWPUpload* cu *Plupload* în 3.5.2, și când o ramificare securizată a SWFUpload a fost făcută disponibilă de echipa de securitate[15] pentru acele module care au continuat să folosească pe termen scurt SWFUpload.

1.4.10 A10 – Redirectări și trimiteri nevalidate

Sistemul intern de autentificare și control acces al WordPress va asigura protecția împotriva încercărilor de a direcționa utilizatorii spre destinații nedorite sau redirectări automate. Această funcționalitate este de asemenea accesibilă dezvoltatorilor de module printr-un API, wp_safe_redirect()[16].

1.5 Alte riscuri de securitate și motive de îngrijorare

1.5.1 *Atacuri la procesarea XXE (*XML eXternal Entity*) a entităților externe XML*

La procesarea XML, WordPress invalidează încărcarea unor entități XML personalizate pentru a preveni atât atacurile de tip Entitate externă cât și cele de tip Expansiune entitate. Dincolo de funcționalitatea de bază a PHP, WordPress nu asigură un API suplimentar de procesare XML pentru autorii de module.

*1.5.2 Atacuri SSRF (*Server Side Request Forgery*) – Cereri falsificate către server*

Cererile HTTP lansate de WordPress sunt filtrate pentru a preveni accesul la adrese IP private sau de retur (*loopback*). În plus, accesul este permis numai pe anumite porturi HTTP standard.

2.3 Securitatea modulelor și temelor WordPress

2.3.1. Tema implicită

WordPress are nevoie de o temă validată pentru a face conținutul vizibil în față pe sit. Tema implicită, livrată cu nucleul WordPress, (acum „*Twenty Fifteen*„) a fost revizuită viguros și testată din punctul de vedere al securității atât de echipa dezvoltatorilor de teme cât și de echipa de dezvoltare de bază.

Tema implicită poate să servească ca punct de plecare pentru dezvoltarea de teme personalizate, iar dezvoltatorii de situri pot să creeze teme copil care să includă unele personalizări dar să se bazeze pe tema implicită pentru majoritatea funcționalității și a securității. Tema implicită poate fi înlăturată cu ușurință de administrator dacă nu e necesară.

2.3.1. Depozitarele de module și teme de la WordPress.org

Sunt aproximativ 30.000+ de module și 2.000+ de teme listate pe situl WordPress.org. Aceste teme și module sunt înscrise pentru a fi incluse și sunt manual revizuite de voluntari înainte de a deveni disponibile în depozitar.

Includerea modulelor și temelor în depozitare nu garantează că acestea nu au vulnerabilități de securitate. Există ghiduri pentru dezvoltatorii de module pe care aceștia să le consulte înainte de la depune pentru includerea în depozitar a unui modul[17], iar pe WordPress.org este disponibilă și o documentație extensivă despre cum se dezvoltă o temă WordPress[18].

Fiecare modul și fiecare temă au posibilitatea de a fi dezvoltate în mod continuu de către proprietarul modulului sau temei respective, iar orice următoare reparații sau facilități adăugate pot fi încărcate în depozitar și făcute disponibile utilizatorilor ce au instalat acel modul sau temă împreună cu o descriere a modificărilor.

Administratorii de situri sunt notificați despre modulele ce trebuie să fie actualizate prin panoul de control din administrare.

Când este descoperit un modul cu vulnerabilități de către echipa de securitate WordPress, aceasta contactează autorul modulului și lucrează împreună să-l corecteze și să lanseze o nouă versiune sigură. Dacă răspunsul autorului modulului întârzie și vulnerabilitatea e severă, tema/modulul sunt scoase din directorul public, iar în unele cazuri, reparate și actualizate de către echipa de securitate.

2.3.2 Echipa de revizuire a temelor

Echipa de revizuire a temelor este un grup de voluntari, conduși de membri cheie consacrați ai comunității WordPress, care revizuiesc și aprobă temele depuse pentru a fi incluse în directorul oficial de teme WordPress. Echipa ține la zi ghidul oficial de revizuire a temelor[19], datele unității de testare a temei[20] și modulul de verificare temă[21], și încearcă să angajeze și să educe comunitatea dezvoltatorilor de

teme WordPress cu privire la cele mai bune practici de dezvoltare. Includerea în grup e moderată de comiteri de bază ai echipei de dezvoltare WordPress.

Rolul furnizorului de găzduire web în securitatea WordPress

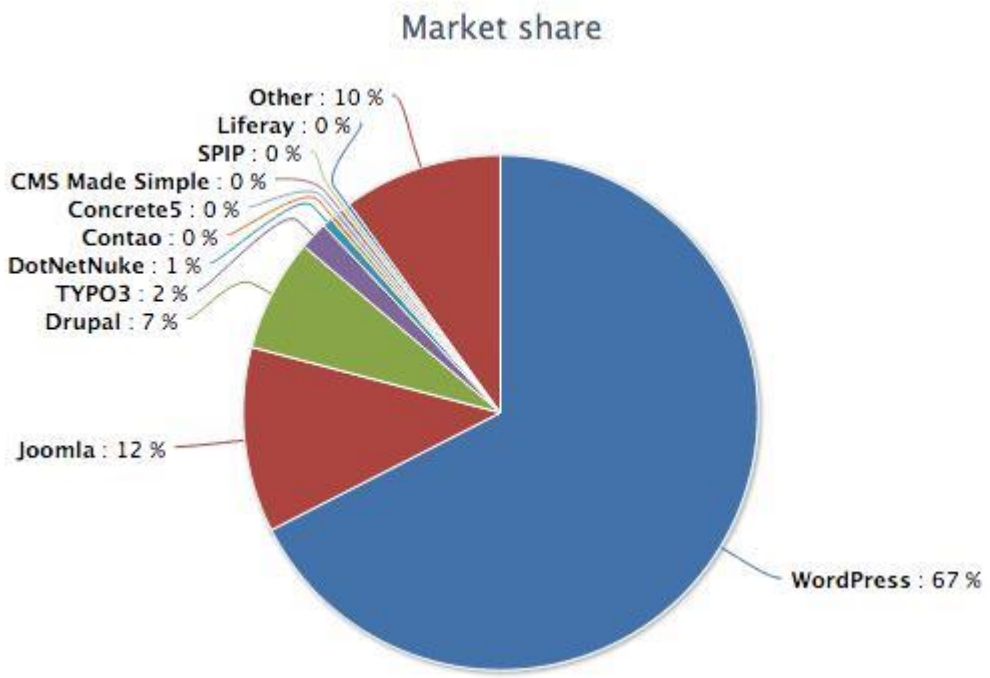
WordPress poate fi instalat pe o mulțime de platforme. Cu toate că nucleul software al WordPress asigură multe posibilități de a opera o aplicație web sigură, trecute în revistă în acest document, configurarea sistemului de operare și a serverului web de dedesubt care îl găzduiește este la fel de importantă pentru a menține securizate aplicațiile WordPress.

2.3.3.1 O notă despre WordPress.com și securitatea WordPress

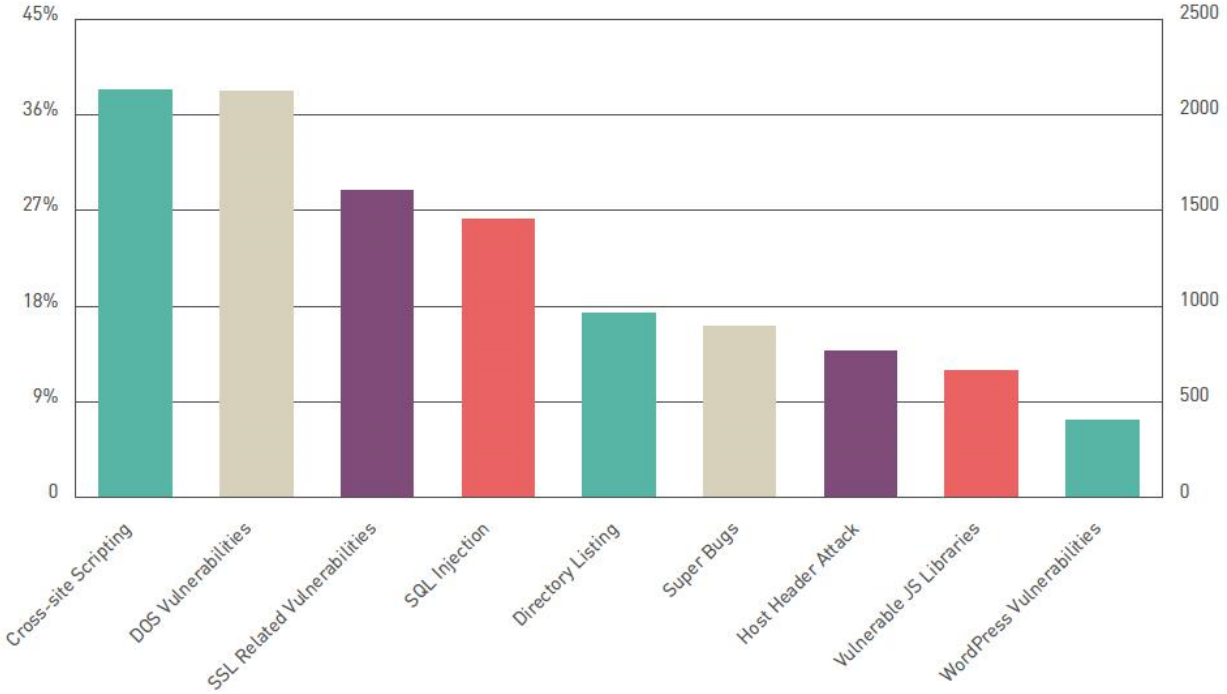
WordPress.com este cea mai mare instalare WordPress la nivel global, ce aparține și este gestionată de Automattic, Inc., fondată de Matt Mullenweg, co-creatorul proiectului WordPress. WordPress.com deși rulează softul de bază WordPress, are propriile procese de securitate, de risc și de soluționare[22]. Acest document se referă la securitatea softului descărcat de la WordPress.org pentru o platformă auto-găzduită pe orice server din lume.

2.4. Statistica

Mai mult de 2 milioane de site-uri sunt alimentate de WordPress și care deține poziția numărul unu, cu 67% din cota de piață în CMS mondială.



Raport de vulnerabilitate recent de Acunetix arată că aproximativ 8% din vulnerabilități găsite în site-uri sunt legate de WordPress.



Tehnologii
Angular 2/4

SPA-uri

Dacă cumva n-ai fost prin zonă în ultimii 5 ani, astăzi scriem SPA-uri (Single Page Applications), nu doar site-uri. De ce? Pentru că utilizatorii vor viteză. Nu au timp să aștepte după *server roundtrips* sau după pagini HTML servite una câte una. În concluzie, nu ne permitem să servim 20 de fișiere JS într-o pagină. Nicidecum. Astăzi, servim unul singur, care conține toată aplicația noastră. El se numește *bundle*. Dar acest *bundle* nu conține doar JS. El poate conține tot: atât HTML și CSS, cât și JS.

Bundle-uri, Webpack si ES2015

"Nu sună rău," vei zice, "dar cine știe să îmi adune toate fișierele și să le pună în acest *bundle*?" Există mai multe *tooluri*, dar unul de top este **Webpack**. Webpack este un **module bundler**. Ce înseamnă asta? Un *module bundler* e un *tool* căruia îi servești modulele pe care le-ai scris. Apoi el rezolvă dependințele dintre **module**, urmând ca rezultatul să consiste dintr-un fișier: **bundle**-ul tău. Acel *bundle* creează toate elementele HTML necesare în momentul în care e rulat în browser. De asemenea, tot *bundle*-ul aplicației e cel care injectează CSS-ul în aplicația ta la *runtime*. Magic!

"Dar ce sunt acelea module?" Excelentă întrebare! Toată lumea râdea de JavaScript până nu demult. Și principalul motiv era faptul că JS îți permitea să declari **variabile globale** cu ușurință. Adică, orice zonă a aplicației tale putea să aibă acces la variabila creată de tine și s-o modifice. Și de aici problema: Devine tot mai greu să urmărești starea aplicației tale în timp. Cine modifică variabila și de ce? Aici intervin modulele. În loc să fie totul expus în câmp deschis, cum era în cazul variabilelor globale, un modul e ca și un gard în jurul curții tale. Orice faci în curtea ta rămâne acolo. Dacă vrei să îi arăți vecinului ce faci, iei ce ai și **exporți** peste gard. În acest fel, vecinul poate folosi din ce ai tu în curtea ta, dar tu controlezi exact ce anume. Așadar, aplicația ta poate fi considerată un cartier. Un cartier care e format din mai multe curți închise care comunică în mod controlat între ele.

Deci, un **modul** poate **importa** elemente din alte module și poate **exporta** în aceeași manieră către alte module. Ceea ce înseamnă că aplicația devine un graf format din dependințe între module. Cine ne oferă acestă funcționalitate? Chiar JavaScript, prin noua versiune ES2015 (denumită și ES6, nume care vine de la grupul care defineste standardele JS: ECMAScript, pe scurt — ES)! Lucrurile s-au schimbat destul de mult de la era jQuery încoace, nu-i așa? Și nu te plafona în faptul că avem nevoie să convertim codul ES6 în ES5. Această problemă e temporară. Producătorii de browsere sunt mai convinși ca niciodată că trebuie să țină pasul mai îndeaproape cu noile standarde ECMAScript, așadar, în câțiva ani nu vom mai avea această problemă.

Web Components

Dacă încă citești aceste rânduri, poate te întrebi de ce nu am zis nimic de Angular 2 până acum. Bună întrebare! Felul în care scriam JavaScript în trecut (inclusiv Angular 1) e diferit de felul în care scriem Angular 2. De altfel, schimbarea majoră a venit odată cu… React. Da, ai citit bine! În 2013, când apare React în lumea JS, acesta introduce un nou concept care avea să influențeze *web development*-ul considerabil: Web Components. Adică elemente customizate, create de către *developer* care să permită reutilizarea lor pe parcurs. Cu alte cuvinte, o parte de aplicație care primește o informație și știe ce să facă cu ea. O parte încapsulată. Modulele și clasele introduse de către ES2015 ne pun la dispoziție tocmai vocabularul de care aveam nevoie pentru a putea scrie aceste Web Components. Știm că React e performant tocmai datorită felului în care e transmisă informația între componente, permițând optimizarea operațiilor costisitoare de UI. Deci Angular 2, în mod natural, devine mult mai performant decât versiunea inițială tocmai din acest motiv. Faptul că există un astfel de nivel de deschidere care să permită lucrul în echipă între diferite proiecte din lumea JS e un avantaj pentru fiecare din noi.

Zones

Acum că avem noțiuni de bază despre Web Components, probabil ne întrebăm cum comunică acestea între ele. **Change Detection** e complet rescris în Angular 2. Acest nou mecanism de administrare a schimbărilor care apar în structura informației a fost încapsulat într-o nouă librărie: **Zone.js**. Această librărie oferă posibilitatea execuției codului într-un mediu delimitat (denumit Zone), în care schimbările asupra variabilelor sunt detectate chiar și asincron. Datorită performanței și utilității lor, Zones au fost propuse pentru a fi incluse într-o versiune viitoare de JavaScript.

Data Binding

O altă schimbare intervine în felul în care circulă informația în cadrul aplicațiilor Angular 2. Dacă în versiunea precedentă eram obișnuiți să modificăm date într-o parte și schimbările să fie reflectate peste tot pe unde se folosesc acele date (concept numit **Double Data Binding**), aceasta s-a dovedit a nu fi cea mai bună abordare din cauza problemelor de performanță. Așadar, cum am amintit mai sus, Angular 2 pășește din nou pe urmele lui React și adoptă ca stil *default* de lucru **Single Data Binding**. Dacă înainte aveam o stradă cu două sensuri pentru circularea datelor în aplicații, acum avem *by default* o stradă cu sens unic. Informația întotdeauna pleacă dintr-o parte și circulă către *layerul* UI. Ca urmare a acestei schimbări, performanța lui Angular 2 e vizibil îmbunătățită față de predecesorul său.

TypeScript

Dacă deja simți că se lasă tăcerea în bucătărie, acum e momentul să spui "Dar stați, că n-am terminat!" Angular 2 recomandă să folosim TypeScript. Ce e Typescript? E o extensie a limbajului JavaScript care permite anotarea tipurilor de date. Știu, e controversat. Acum mai bine de un an, când am văzut prima dată TypeScript, ideea nu mi-a surâs foarte tare. Dar între timp, am văzut că există două feluri de a folosi TypeScript: ca pe un *tool* care să ajute la viteza de *development*, respectiv ca pe o povară, din cauză că vrei ca totul să fie anotat până la ultimul punct și virgulă. Pe scurt, recomandarea mea e pentru a folosi TypeScript în măsura în care e util și ne ajută să fim mai eficienți. Dar am observat din proprie experiență că sunt mult mai rapid folosindu-l.

Alte funcționalități

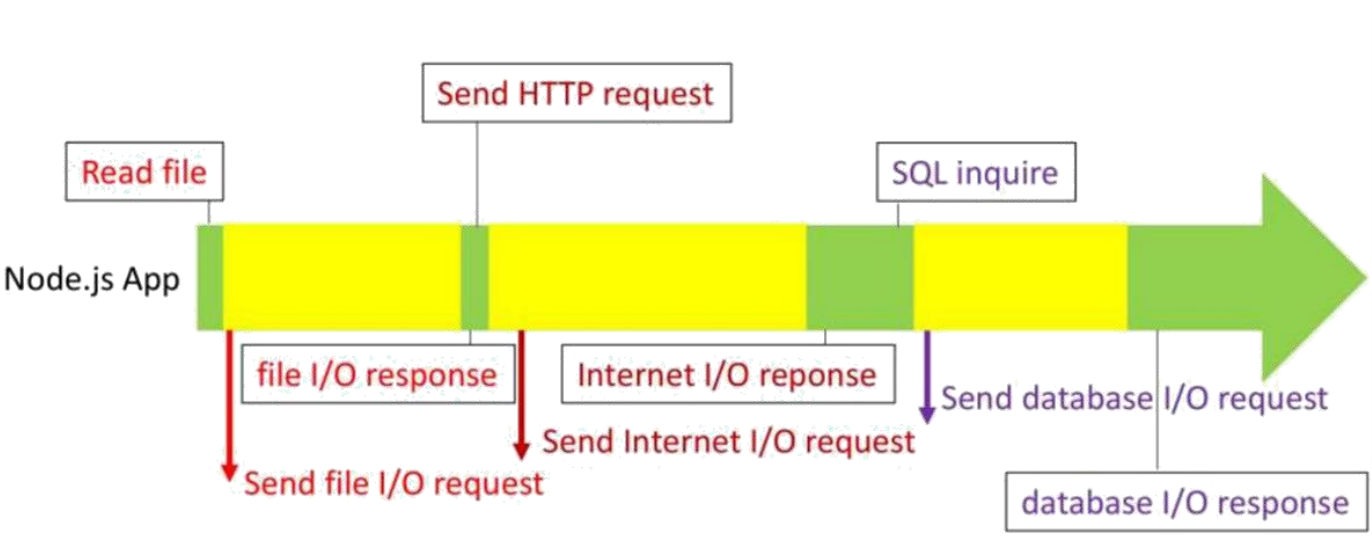
E imposibil să acoperim totul despre Angular 2 într-un articol. Deja am sentimentul că trebuie să scriu mai mic, ca să fiu sigur că încape tot. Noua versiune a *frameworkului* ne oferă și alte funcționalități. **Server-Side Rendering** permite generarea primului *view* direct pe server și servirea lui în variantă statică -- totul în numele vitezei sporite. Faptul că nu e gândit doar pentru browser, ci și pentru server (ajungând chiar și în ecosistemul mobile), face din Angular 2 o platformă, nu doar un *framework*. **Ahead of Time Compilation (AoT)** înseamnă că *template*-urile HTML pot fi compilate direct pe server si apoi trimise direct către browser pentru a fi folosite. Vechea metodă, Just in Time Compilation (JiT) presupune că browserul trebuie să facă toată munca de *randare* a *template*-urilor. De asemenea, se poate folosi acum și **Tree Shaking**, o metodă care presupune analizarea statică a *bundle*-ului generat și eliminarea codului care nu e necesar. Scopul acestei tehnici este să elimine cât mai mult cod din *framework* cu putință. Lista nu se termină aici, dar acestea sunt cele mai importante noutăți.

Concluzie

În încheiere, sper că subiectele abordate mai sus să te ajute data viitoare când te găsești în mijlocul unei conversații despre lumea *frontend*. Angular 2 e un *tool* excelent, care va fi cu noi… încă preț de câteva luni. Pentru că Google pregătește deja lansarea Angular 4 în prima parte a lui 2017 (au trecut pe Semantic Versioning). Saltul peste versiunea 3 se datoreaza faptului că vor să păstreze constistența cu librăria de routing pentru Angular, ea însăși ajunsă la versiunea 4. În mod evident, diferențele nu vor fi așa dramatice ca și trecerea de la versiunea 1 la 2, dar schimbări tot vor fi. Și acesta nu e rău, chiar dimpotrivă, pentru că nu e doar datoria *toolurilor* pe care le folosim să progreseze, ci și a noastră să ținem pasul cu ele. În loc să ne plângem de afluxul de librării și *tooluri* din lumea JS, mai bine să fim mulțumitori pentru toate jucăriile care ne sunt puse la dispoziție în lumea *connected* în care trăim. Pentru că, până la urmă, să fii JS Dev in 2016 e ca și cum ai fi un copil mic scăpat liber într-un magazin de jucării, cum ar spune Eric Elliott. E o lume dură în bucătăriile firmelor. Vreau să cred că și tu și eu avem acum o armă în plus care să ne ajute să supraviețuim.

NodeJS

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem – npm – is the largest ecosystem of open source libraries in the world."

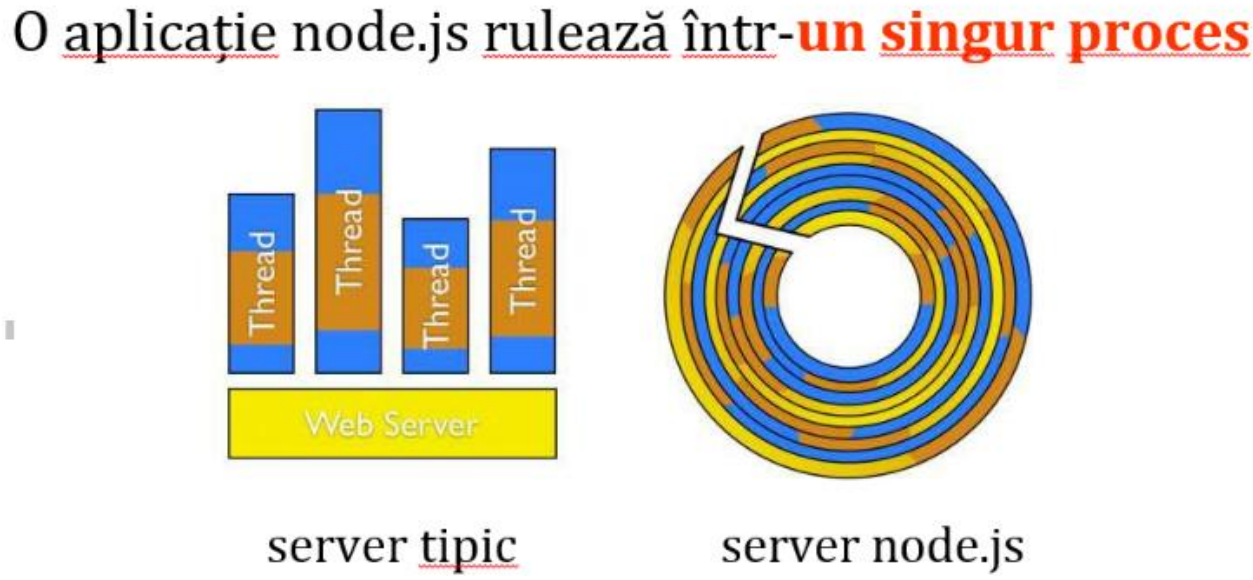


Operațiile de intrare/ieșire sunt asincrone fiecare cerere (operație) adresată aplicației

– e.g., acces la disc, la rețea, la alt proces – poate avea atașată o funcție de tratare a unui eveniment specific evented I/O

cod JS executat de client (*browser* Web)		**cod JS rulat pe partea de server (node.js)**
așteaptă și tratează evenimente de interacțiune (onclick, onmouseover, onkeypressed,...)	**procesare bazată pe evenimente** *evented/ event-based*	așteaptă și tratează cereri (evenimente) provenite de la client(i)
programul trebuie să fie responsiv atunci când așteaptă încărcarea datelor de pe rețea (*e.g.*, JSON, XML, imagini, video) via Ajax/Comet ori *socket*-uri Web	**asincronism** (*e.g.*, operatii neblocante)	programul trebuie să fie responsiv atunci când așteaptă încărcarea datelor locale/externe (preluate din baze de date, fișiere, servicii Web, API-uri,...)

Dr. Sabin Buraga



Securitatea aplicaţiilor web

1. SQL Injection
2. Logic Flaw
3. Authorization Bypass
4. Cross-site Scripting (XSS)
5. Authentication Bypass
6. Vulnerable Third Party Software
7. Session Handling Flaw
8. Cross-site Request Forgery (CSRF)
9. Verbose Errors
10. Source Code Disclosure

1. SQL Injection

Majoritatea proiectelor web, folosesc baze de date pentru a-şi stoca şi ordona datele. Structured Query Language (SQL) este folosit pentru a accesa informaţiile dintr-o bază de date, sintaxa acesteia poate să difere puţin în cazul diferitelor servere de bazele de date, însă, SQL este un limbaj universal potrivit pentru toate bazele de date.
O vulnerabilitate numită injecţie cu cod sursă SQL (pe scurt injecţie SQL) apare atunci când un atacator poate introduce orice date într-o interogare SQL sau când, prin injectarea sintaxei, logica declaraţiei, să fie 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 una din cele mai frecvente vulnerabilităţi.

Atacul: Metode de prevenire a atacurilor de tip SQL injection:
Metodele de apărare împotriva atacurilor ar trebui să fie adresate direct tipurilor de atac specifice bazelor de date şi să poată minimiza impactul unui breşe în cazul în care una din metodele de apărare s-a dovedit inadecvată. Cea mai bună metodă de apărare împotriva injecţiilor SQL se

bazează pe rutine puternice de validare a datelor de intrare.
Există măsuri specifice care pot fi luate în cadrul bazei de date şi la nivelul aplicaţie:
• Utilizaţi variabile bine definite şi definiţiile coloanelor bazei de date: Stocaţi şi manipulaţi numerele (ID-uri de sesiune, coduri poştale, date de naştere) ca şi numere întregi sau că 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.
• Atribuiţi rezultatele interogării unei variabile bine definite: dacă aplicaţia caută valori numerice atunci atribuiţi rezultatul unui număr întreg, acest lucru îi împiedică pe atacatori să extragă informaţii din baza de date. Nu este posibil să fie obţinut şi afişat numele unei coloane dacă variabila ce urmează să fie afişată în browser nu acceptă decât numere întregi. Această tehnică restricţionează sever anumite atacuri.
• Limitaţi lungimea datelor: toate şirurile de caractere ar trebui să se limiteze la o lungime potrivită scopului lor. Un nume de familie, de exemplu, nu este necesar să fie stocat şi manipulat într-o variabila 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.
• Evitaţi crearea de interogări prin concatenarea de şiruri de caractere: creaţi o funcţie view sau o procedura, 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” + variablea), este cea mai vulnerabilă la atacurile SQL Injection. Pe de altă parte, o funcţie view sau o procedură particularizată, de obicei generaza o eroare dacă primeşte date de intrare incorecte, însă, nu îi va permite unui atacator să manipuleze întreagă intrerogare.
• Aplicaţi separarea 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 ei. Cataloagele interne ale bazei de date, în special cele legate de managementul conturilor şi variabilele sistemului, nu ar trebui să fie accesibile.

2. Logic Flaw

Toate aplicaţiile web folosesc logica pentru a avea funcţionalitate. A scrie cod într-un limbaj de programare, nu înseamnă nimic altceva decât descompunerea unui process complex în paşi mici şi simplii, pentru, a aduce ceea ce este pe înţelesul oamenilor la nivelul la care poate fi executat de către computer iar atunci când un număr mare de programatori şi designer 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, de multe ori, însă, sunt trecute cu vedrea deoarece, rareori, poat fi scanate cu programe specializate în scanarea vulnerabilităţilor, nu au o semnătură specializată ca şi vulnerabilităţile la injecţiile SQL 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 asupra aplicaţiei codul scris de el, luând în considerare 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, ele, prezintă un interes foarte mare pentru atacatori.

Defectele logice nu pot fi definite şi învăţate prin terorie, deci, cea mai bună metodă de explicare a lor este prin exemplu concret.

Exemple:
• Păcălirea funcţiei pentru schimbarea parolei: autorii au descoperită această eroare de logică într-o aplicaţie web utilizată de către o societate de servicii financiare şi, de asemenea, în aplicaţia AIM Enterprise AOL Gateway.
Functionaliatea: 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 administrator, prin care aceştia puteau schimba parolă 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 a□□ 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 carerea 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 for furniza intotdeuna vechea parolă.

?
```
1       String existingPassword = request.getParameter(a□□existingPassword”);
2       if (null == existingPassword)
3       {
4       trace("Old password not supplied, must be an administrator”);
5       return true;
6       }
7       else
8       {
9          Trace("□□Verifying usera□□s old password”);
10         ...
11      }
```
Atac : 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, 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 resteteze parola oricărui alt utilizator preluând, astfel, controlul asupra contului acestuia.

• Ştergerea unei piste de audit: acest defect logic a fost descoperit într-un centru de telefonie.
Funcţionalitate: 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, inclusive crearea de conturi şi schimbarea de parole, aşadar, aplicaţia a menţinut o gestiune complete de audit, înregistrând fiecare acţiune efectuată şi identitatea utilizatorului care a responsabil. Aplicaţia includea o funcţie care le permitea administratorilor să sterga anumite înregistrări de audit, însă pentru a evita exploatarea în scopuri rău-voitoare, orice utilizare a funcţiei de ştergere era la rândul ei înregistrată, pentru a se putea şti utilizatorul responsabil de utilizarea acestei funcţii.

Presupunerea: designerii aplicaţiei au crezut că nimeni nu poate şterge înregistrările de audit fără a lasă măcar o urmă (această presupunere a fost testată de către administratorii lor, intotdeuna rămânând ultima înregistrare a persoanei care a şters datele).
Atac: presupunerea designerilor a fost greşită, există o cale de a accesa datele, a produce modificări asupra lor şi de a îţi şterge urmele în întregime. Paşii sunt următorii:
Autentificarea cu contul propriu, şi creaţi un nou cont de utilizator.
Atribuiţi toate privilegiile dumneavoastră noului cont.
Utilizaţi noul cont pentru a duce atacul la capăt.
Utilizaţi un 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, dar cum cel care i-a dat privilegii noului cont 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. Crima perfectă.

Prevenirea apariţiei defectelor logice:
Nu există nici o reţetă perfectă de a evita erorile logice, ci doar o serie de sfaturi care va poat ajuta să evitaţi aceste erori:
• Documentaţi toate aspectele legate de designul aplicaţiei suficient de detaliat pentru a putea fi uşor înţelese de cineva din exterior.
• Lăsaţi comentarii în codul sursă care să includă următoarele informaţii:
Scopul şi funcţionalitatea fiecărei bucăţi de cod.
Presupunerile făcute de fiecare componentă în legătură cu elementele care nu se află direct sub controlul ei.
Adăugaţi comentarii pentru fiecare bucată de cod care să facă trimitere la toate componentele care îl folosesc.

• 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.
• 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 component ale codului.

3. Authorization Bypass

Procesul prin care se stabileşte dacă un utilizator, care foloseşte o anumită identitate, are permisiunea de a accesa o resursă sau nu, şi după verificare acordându-i-se accesul la acea resursă, în concordanţă cu politicile sistemului, indiferent de identitalea lui/ei reală (aici ne referim la aplicaţii web unde în general utilizatorii nu îşi folosesc numele reale ci pseudonime) poartă numele de autorizaţie.
Vulnerabilităţile ce ţin de autorizaţii 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 autentificati sau care deţin anumite privilegii în acele aplicaţii.
Vulnerabilităţi comune sunt:
• Traversarea de directoare
• Evitarea unor mecanisme de autorizare

• Creştere a 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 le are 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, oricând un atacator a reuşit să acceseze un privilegiu restricţionat nivelului lui/ei de acces, a reuşit să evite mecanismul de autorizare, acest lucru se face de obicei prin shimbarea rolului într-unul superior.
Atacul:
Avem sursa: http://www.exemplu.com/index.php?page=login. Un atatc 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 aplictiei, într-un 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 nu se referă doar la câmpurile în care utilizatorul poate introduce date, ci şi la orice valoare pe care utilizatorul o poate modifică 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 cât mai puţine privilegii utilizatorului, cu cât acesta care mai puţine privilegii cu atât scad şansele de a putea duce la capăt un atac asupra aplicaţiei web.

4. Cross-site Scripting (XSS)

XSS este o tehnică de atac, folosit pentru a forţa o pagină web să afişeze un cod maliţios (scris, de obicei, în HTML (Hypertext Markup Language)/ JavaScript (numit malware)), pe care îl execută ulterior în browser-ul unui utilizator. Acest tip de atac, nu are ca ţintă serverul site-ului web, (acesta este doar o gazdă), ci codul malware este executat direct în browser, deoarece, adevărata ţintă a atacului este utilizatorul. Hackerul va folosi doar site de încredere pentru a efectua atacul , şi odată ce are control asupra browser-ului utilizatorului, îl va putea folosi pentru a îi: fura un cont victimei, înregistrarea tastelor apăsate de la tastatură victimei, furtul înregistrărilor din istoricul browser-ului, etc.
Pentru a infecta un browser, trebuie să vizitaţi o pagină care conţine malware JavaScript.

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.
• 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.
• Poate fi folosită o vulnerabilitate permanentă la XSS, iar malware-ul să fie injectat într-o zonă publică a paginii web.
• Victima poate accesa un link special pregătit în spatele căruia se ascunde un XSS non-persistent sau bazat pe Document Object Model (DOM).

Atacul:
Non-persistent:
Dacă atacatorul doreşte să atace prin XSS pagina http://vulnerabil/, un site de comerţ electronic mai întâi trebuie să găsească o vulnerabilitate la XSS. Pentru asta acesta 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 a□□test pentru xss" în câmpul de căutare răspunsul va fi un nou url cu un şir de interogare care conţine testez+pentu+xss că şi valoare a parametrului p. Această valoare poate fi schimbată dacă introducem codul HTML/JavaScript: „><script>alert('XSS%20Testing'). Ca şi rezultat pagina va afiza o fereastră de dialog inofenseiva (după instrucţiunea din cod) care este acum parte din pagină, demonstrând succesul codului care acum face parte http://vulnerabil/. De aici URL-ul poate fi modificat să conţină atacuri XSS mai complexe (ex: furtul de cookie-uri).

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 următor, doar că are o caracteristică în plus pentru afişarea promoţiilor şi că interogările din URL-urile pentru afiseare 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 adaugă 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 orbeşte 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).
Persistent:
Atac XSS persistent sau injecţie 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, chaturi, etc.). Odată ce utilizatorul vizitează pagina infectată, execuţia este automată ceea ce face a acest tip de atac să fie mult mai periculor decât primele două, deoarece, nu există cale prin care utilizatorul se poate apăra, şi până, şi utilizatorii care ştiu despre această vulnerabilitate pot fi uşor compromişi.

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

Codificarea datelor de intrare şi de ieşire au fiecare argumentele lor pozitive şi negative. Partă pozitivă a codificării datelor de intrare vă oferă un singur punct de acces, în timp ce, codarea datelor de ieşire va ofere 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, de a accesă doar link-urile cunoscute, şi a avea grijă la ce informaţii divulgaţi depre conturile dumneavoastră, aceste precauţii pot face o mare diferenţă.

• 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 ordinte trebuie filtrele aplicate şi cum interacţionează unul cu celalat.

• Codarea datelor de intrare:

Poate crea un singur punct de intrare a datelor pentru toate codarile.
Vă poate proteja împotriva la mai mult decât de vulnerabilitatea la XSS, va poate proteja, de asemenea, de injecţii cu cod SQL şi injecţii de comandă care pot fi verificate înainte de a stoca informaţii în bază de date.
Nu poate opri atacurile persistente de tip XSS odată stocate.

• Codarea datelor de ieşire:

Este mai detaliat şi poate lua şi contextul în considerare.
Se poate că dezvoltatorii să trebuiască să efectueze codarea de mai multe ori pentru aceeaşi locaţie acolo unde este trimisă informaţia.

• 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 şi personalizaţi-va setările de securitate pentru a reduce riscul de atac.

5. Authentication Bypass

Autentificarea dovedeşte, într-o oarecare măsură, identitatea unei persoane sau entităţi. De exemplu, toţi folosim parole pentru a ne loga în conturile personale de e-mail. Prin asta ne dovedim identitea. Paginile web folosesc certificate Secure Socket Layer (SSL) pentru a valida că traficul provine într-adevăr de la domeniul solicitat de către site, acest lucru ne asigura că site-ul este cel adevărat şi nu o copie. Atacatorul are două opţiuni pentru a sparge un sitem de autentificare: utilizarea unei parole furate sau evitarea verificării autentificării. Pentru indentifică şi monitoriza activitatea unui utilizator pe o pagina web, acestuia îi se atribuie un token de sesiune unic de obicei sub formă de cookie-uri. Acest lucru ajută site-ul web să îşi diferenţieze utilizatorii între ei şi li se atribuie

utilizatorilor atunci când accesează pagină web, înainte că aceştia să se autentifice (odată autentificati site-ul atribuie cookie-ul utilizatorului respectiv.
Odată autentificat utilizatorul respective este identificat doar după cookie-ul de sesiune, deci dacă un atacator îl compromite, ghicindu-i valoarea sau furându-l, reuşeşte să treacă cu success de mecanismul de autentificare a paginii respective şi să îi ia locul victimei.

Atacul:
Cookie-urile de sesiune pot fi compromise prin mai multe metode:

• Cross-site scripting (XSS): dacă atributul HttpOnly nu este setat JavaScript poate accesă obiectul document.cookie. Cea mai simplă formă de atac acest cod trimite numele cookie-ului=valoare unui site unde atacatorul poate vedea traficul venit din exterior.
• Cross-site request forgery (CSRF): atacatorul exploatează indirect sesiunea unui utilizator, pentru asta victimă trebuie să fie deja autentificată pe site-ul ţintă. Atacatorul plasează o pagină capcană pe un alt site, când victimă vizitează pagină infectată, browser-ul face în mod automat o cerere către pagină ţintă folosind cookie-ul de sesiune al victimei.
• 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 spaţiul de memorie al serverului web, dacă un atacator sparge bază de date, poate fură cookie-urile de sesiune.
• Network snifffing: HTTPS incripteaza traficul dintre browser şi pagină 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 parolă 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.).

Metode de aparare impotriva atacurilor de tip authentication bypass:
Autentificarea dovedeşte, într-o oarecare măsură, identitatea unei persoane sau entităţi. De exemplu, toţi folosim parole pentru a ne loga în conturile personale de e-mail. Prin asta ne dovedim identitea. Paginile web folosesc certificate Secure Socket Layer (SSL) pentru a valida că traficul provine într-adevăr de la domeniul solicitat de către site, acest lucru ne asigura că site-ul este cel adevărat şi nu o copie. Atacatorul are două opţiuni pentru a sparge un sitem de autentificare: utilizarea unei parole furate sau evitarea verificării autentificării. Pentru indentifică şi monitoriza activitatea unui utilizator pe o pagina web, acestuia îi se atribuie un token de sesiune unic de obicei sub formă de cookie-uri. Acest lucru ajută site-ul web să îşi diferenţieze utilizatorii între ei şi li se atribuie utilizatorilor atunci când accesează pagină web, înainte că aceştia să se autentifice (odată autentificati site-ul atribuie cookie-ul utilizatorului respectiv.
Odată autentificat utilizatorul respective este identificat doar după cookie-ul de sesiune, deci dacă un atacator îl compromite, ghicindu-i valoarea sau furându-l, reuşeşte să treacă cu success de mecanismul de autentificare a paginii respective şi să îi ia locul victimei.

Atacul:
Cookie-urile de sesiune pot fi compromise prin mai multe metode:

• Cross-site scripting (XSS): dacă atributul HttpOnly nu este setat JavaScript poate accesă obiectul

document.cookie. Cea mai simplă formă de atac acest cod trimite numele cookie-ului=valoare

unui site unde atacatorul poate vedea traficul venit din exterior.
• Cross-site request forgery (CSRF): atacatorul exploatează indirect sesiunea unui utilizator, pentru asta victimă trebuie să fie deja autentificată pe site-ul ţintă. Atacatorul plasează o pagină capcană pe un alt site, când victimă vizitează pagină infectată, browser-ul face în mod automat o cerere către pagină ţintă folosind cookie-ul de sesiune al victimei.
• 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 spaţiul de memorie al serverului web, dacă un atacator sparge bază de date, poate fură cookie-urile de sesiune.
• Network snifffing: HTTPS incripteaza traficul dintre browser şi pagină 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 parolă 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.).

Mecanismele de sesiune şi autentificare a unui site trebuie să fie folosite în cadrul unor practici bune de securitate, deoarece fără măsuri bune de contraatac, o slăbiciune într-o parte a aplicaţiei web poate cu uşurinţă compromite o altă parte a acestuia.

6. Vulnerable Third Party Software

Când vine vorba de aplicaţii care provin de la alte 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 securiate ale aplicaţiei web. Multe aplicaţii web provenite din terţe părţi sunt nesigure şi de multe ori interfeţele acestor programe vin cu un nume de utilizator implicit şi parolă a□□admin". Această este o breşă gravă de securitate deoarece, fiind atât de uşor de a□□ghicit" numele de utilizator şi parolă, un atacator poate accesa orice parte a aplicaţiei web, inclusive cea a consolei de comandă, în care poate introduce date cu care poate manipulă direct sistemul de operare pe care se bazează serverul aplicaţiei respective, aşa poate obţine date privilegiate, şterge/modifică bază de date a aplicaţiei, poate schimbă rolurile utilizatorilor, etc.
De asemenea, aplicaţiile web provenite din terţe surse, pot fi vulnerabile la toate atacurile la care pot fi vulnerabile şi aplicaţiile web făcute de noi (XSS, SQL injection, etc.)
Pentru a va proteja de breşe de securitate, oricând adăugaţi un nou software aplicaţiei dumneavoastră web, nu faceţi presupuneri că sunt sigure, sau că au fost testate amănunţit înainte de a fi scoase pe piaţă şi toate problemele rezolvate, ci testatile dumneavoastră cât puteţi de amănunţit pentru a va asigura că nu veţi avea probleme mai târziu. O eroare apărută într-un program vă pot pune în pericol întreagă aplicaţie web.

Raportul Verizon pentru 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 caliatate a securităţii inacceptabilă: cele mai grave probleme au fost descoperite la programele de asistentă pentru clienţi (82% inacceptabile), uramate de programele de securitate (72%).
• Dezvoltatorii au nevoie de mai mult train-ing 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 remediati. Nu faceţi presupuneri când este vorba de securitate.

7. Session Handling Flaw

Autentificarea şi managementul de sesiune includ toate aspectele ce ţin de manipularea datelor de autentificare ale utilizatorului şi managemnetul sesiunilor active ale acestuia. Autentificarea este un process critic al acestui aspect, dar până şi cel mai solid proces de autentificare poate fi subminat de erori ale funcţiilor de management pentru verificarea credentialelor, incluzând: schimarea 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 dezvoltate, de cele mai multe ori, subestimează complexitatea necesară pentru a proiecta o metodă de autentificare şi management al sesiunilor care să protejeze corespunzător credentialele, 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, cum HTTP 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 îşi creeze propriile token-uri de sesiune.
Dacă toate credentialele de autentificare şi identificatorii de sesiune nu sunt protejate corespunzător, prin SSL în permanenţă, 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 evitatea 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:
• 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.

• 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. Sistemul, de asemenea 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.
• Comenzile de schimbare a parolelor: ar trebui folosit un singur mecanism de schimbare a parolelor indiferent de circumstanţele în care acest lucru se inatmpla. Utilizatorul să trebuiască întotdeauna să scrie 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 a□□uitata".
• Stocarea parolelor: toate parolele trebuie criptate sau sub formă de hash-uri indiferent de locul unde sunt stocate. Este de preferat stocarea sub formă de hash-uri deoarece acestea nu sunt reversibile.
• 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).
• 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.
• 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. Dacă o relaţie de încredere este absolut necesară, atunci ar trebui că această nu poată fi , prin mecanisme procedurale de , care protejeze chiar cadrul unei dezvoltări timp.

8. Cross-site Request Forgery (CSRF)

Cross a□□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 autentificati prin a forţa acei utilizatori să facă tranzacţii sensibile în numele atacatorului. Această vulnerabilitate, deşi mai puţin cunoscută că XSS, este mult mai periculoasă decât cross-site scripting, deoarece, îşi are rădăcinile în natură lipsită de stare ale specificaţiilor HTTP-ului, care cer că un token de autentificare să fie trimis cu fiecare cerere a utilizatorului.
În mod obişnuit, vulnerabilităţile web apăr că 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, apăr atunci când dezvoltatorii omit un mecanism de prevenire a XSRF din aplicaţia lor.

Atacul:
Un exemplu classic este cel al unei aplicaţii bancare, carré 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:

```
?
1    http://xsrf.bancavulnerabila.com/transferFonduri.aspx?
```

2 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, care conţine următorul cod JavaScript:

?

```
1 &lt;script type a□□text/javascript”>
2 Var i document.createElement(a□□image”);
3 i.src="http://xsrf.bancavulnerabila.com/transferFonduri.aspx?"Incontul=ATACATOR&fonduri=1000.00&
4 &lt;/script>
```

Efectul acestui cod este de a creea 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 autentificati 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 că şi cum utilizatorul ar fi făcut-o în mod voluntar.

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

• Cookie-uri postate de două ori: această metodă de apărare constă în introducerea unui câmp de introducere a datelor secret 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 se va loga ca potenţial atac. Această metodă se bazează pe faptul că atacatorul nu ştie valoarea cookie-ului de sesiune al utilizatorului, dacă prin altă metodă acesta reuşeşte să afle valoarea aceasta, strategia de apărare nu va avea success.

• Nonce unic pentru formular: este probabil cea mai folosită metodă de apărare împotrivă CSRF şi consata în construirea fiecărui formular folosind un câmp ascuns care conţine un nonce (number used once) obţinut folosind un generator pseudoaleator de numere securizate prin încriptare, pentru a nu fi vulnerabil la atac. Când serverul aplicaţiei primeşte valorile parametrilor formularului că 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 valaoarea nonce-ului a expirat.

• Cererea credentialelor de autentificare: această metodă le cere utilizatorilor autentificati să reintroduca parola corespunzătoare sesiunii în care sunt autentificati ori de câte ori fac o tranzacţie sensibilă. Acesta strategie este des întâlnită în aplicaţiile web în cadrul cărora tranzatiile de o natură sensibilă se întâmplă rar (cel mai adesea fiind schimbări ale informaţiilor de pe profilul utilizatorului).

9. Verbose Errors

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 process de autentificare dă greş, poţi în mod evident să îţi dai seama că una din informaţiile introduse nu au fost corecte, însă uneori, apicatia, 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 lista mare de nume de utilizatori commune pentru a află care din ele sunt valide, deşi în general numele utilizatorilor nu sunt considerate a fi secrete, identificarea lor îi da atacatorului şanse mai mari de a compromite aplicaţia folosindu-se de timp, abilitate şi efort. O lista de nume de utilizatori enumerate poate fi folosită ulterior pentru diverse metode de atac incluzând: ghicirea parolelor, atacuri asupra datelor utilizatorilor sau sesiunilor, 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, crescandu-i şansele de a obţine acces neautorizat.

Metode de prevenire a atacurilor bazate pe Verbose Errors:
• Utilizaţi validările pe partea clientului doar pentru performanţă, nu şi pentru securitate: macanismele 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 pot reduce solicitarea severului, impiedicatnd datele introduse greşit în mod neintenţionat să ajungă la acesta.
• Normalizati 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 scuritate şi validare, altel o bucată de cod poate trece prin filtre şi să fie decodată şi descoperită că a fi maliţioasă doar mai târziu.
• Aplicaţi validarea pe partea serverului: doate 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.
• Restrangeti 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.
• 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 validară a datelor de ieşire sau de reformatare a caracterelor reprezintă un nivel additional de protejare împotrivă atacurilor prin injectare HTML, chiar dacă un cod maliţios reuşeşte să treacă de un filtru de intrare a datelor, efectele acestuia vor fii neglijate în momentul în care ajunge în faza de ieşire.
• 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.
• Aveţi grijă cu mesajele de eroare: indiferent de limbajul folosit pentru a scrie aplicaţia erorile ar trbui să urmărească conceptele de incerarca, descoperă, în final când vine vorba de tratarea excepţiilor. Incearcati 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 mesajde eroare politicos care însă nu dezvăluie nici o informaţie despre sistem.
• 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 fisierede din interiorul acelui director.

10. Source Code Disclosure

Divulgarea codului sursă este o eroare de codare foarte des întâlnită în aplicaţiile web, care poate fi exploatate de către un atacator pentru a obţine codul sursă şi configurearea fişierelor prin intermediul HTTP, acest lucru îi oferă 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, structură bazei de date, vulnerabilităţile codului şi comentariile introduse în el. Odată ce are codul sursă şi posibul un duplicat al aplicaţiei pe care să poate face teste, atacatorul se poate pregăti pentru un atac asupara aplicaţiei.

Atacul:
Poate fi făcut prin mai multe metode:

• Folosind vulnerabilităţi cu divulgare a codului sursă cunoscute
• Exploatarea unei vulnerabulitati 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)

De exemplu, luăm în considerare un site web pe care rulează Microsoft Internet Information Server (IIS). Trimitem urmarorul URL serverului web:
http://www.vulnerabil-iis.com/exemplu.%61%73%70
Atacatorul poate obţine codul sursă al acestui exemplu, deoarece există o vulnerabiliatate în serverele IIS când vine vorba de gestionarea fişierelor .asp, care îi permite să obţină codul sursă al fişierelor .asp de la distanţă. Dacă IIS este instalat pe o partiţie FAT şi atacatorul trimite o cerere codată în Unicode pentru a obţine un fişier .asp (%61%73%70 este codul Unicode pentru a□□asp”), serverul IIS nu îl va recunoaşte că şi fişier ASP aşadar nu îl va executa, ci va trimite codul sursă ASP direct browserului.

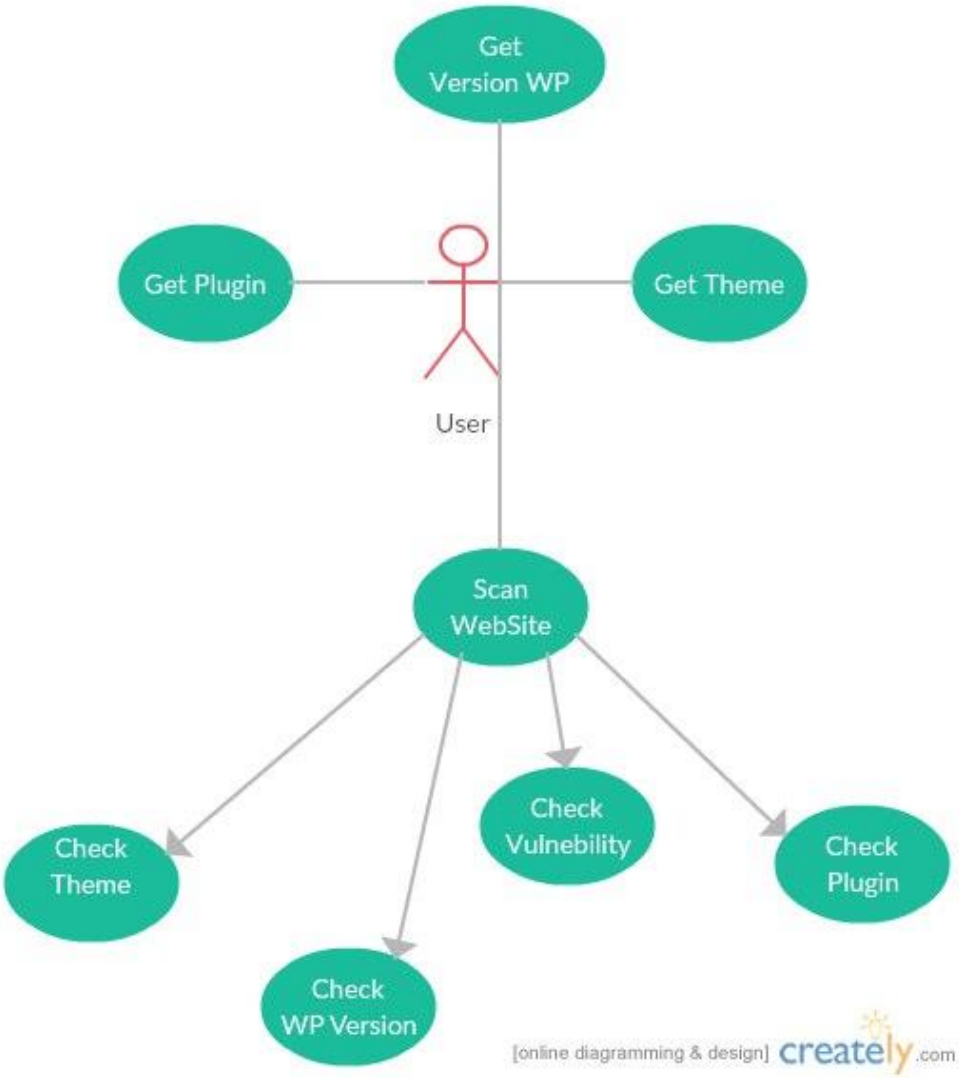
Metode de prevenire a atacurilor de tip Source Code Disclosure:
• Verificaţi folderul de unde este cerut fişierul care urmează să fie descărcat (menţineţi un white list cu numere directoarelor de unde este permisă download-area fişierelor şi validaţi cererile pe bază acestuia).
• Verificaţi tipul de fişiere care sunt cerute de utilizatori.
• Indexati fişierele care pot fi descărcate şi afişaţi doar numărul lor din index că şi parametru al URL-ului.

După cum am exemplificat mai sus, atacurile cibernetice sunt pe cât de reale, pe atât de periculoase, mai ales într-o lume în care informaţia a ajuns să fie cea mai preţioasă marfă din toate, pierderea de

informaţii poate duce la pierderi catastrofale atât financiare cât şi de imagine ale paginii web. Poate una din cele mai bune investiţii într-o afacere este cea făcută pentru protejarea datelor pe care această le deţine.

Diagramele

1. **Digrama use-case**



Code:

```
getPluguin(pluguin_name) {
    return this._http.get(`${this._url}/plugins/${pluguin_name}`)
      .map((response: Response) => response.json())
      .catch(this._errorHandler);
  }

  getTheme(theme_name) {
    return this._http.get(`${this._url}/themes/${theme_name}`)
      .map((response: Response) => response.json())
      .catch(this._errorHandler);
  }

  getVersion(version_name) {
    return this._http.get(`${this._url}/wordpresses/${version_name}`)
      .map((response: Response) => response.json())
      .catch(this._errorHandler);
  }
  checkURL(url) {
    const headers = new Headers({
      'Content-Type': 'application/json'
    });
    return this._http.post('http://localhost:3000/check', url, headers)
      .map((response: Response) => response.json())
      .catch(this._errorHandler);
  }
  getWP(wp_url) {
    return this._http.get(wp_url)
      .map((response: Response) => response.text())
      .catch(this._errorHandler);
  }
  // ================ERROR HANDLER=======================================
  private _errorHandler(error: Response) {
    return Observable.throw(error || 'Server Error');
  }
```

4. Bibliografia

[1] WPScan is a black box WordPress vulnerability scanner. [Regim de access] – https://github.com/wpscanteam/wpscan

[2] SQL инъекции. Проверка, взлом, защита [Regim de access] – https://habrahabr.ru/post/130826/

[3] 10 шагов для защиты вашего WordPress блога [Regim de access] – https://habrahabr.ru/post/62814/

[4] Web Application Exploits [Regim de access] – https://www.exploit-db.com/webapps/

[5] WordPress: небезопасен из коробки — получаем RCE с правами редактора. И еще о Google, стартапе и 1 миллиарде долларов [Regim de access] – https://habrahabr.ru/company/dsec/blog/194282/

[6] Пентест WordPress своими руками [Regim de access] – https://habrahabr.ru/company/dsec/blog/196858/

[7] Пентест WordPress своими руками [Regim de access] – https://habrahabr.ru/post/283210/

[8] Vulners — Гугл для хакера. Как устроен лучший поисковик по уязвимостям и как им пользоваться [Regim de access] – https://habrahabr.ru/company/xakep/blog/305262/

[10] Getting Started With Angular Material 2 [Regim de access] – https://alligator.io/angular/angular-material-2/

[11] The top 10 web vulnerabilities… and what to do about them [Regim de access] – http://www.computerworlduk.com/tutorial/infrastructure/the-top-10-web-vulnerabilities-and-what-to-do-about-them-424/2/

[12] Vulnerabilitati web (Pentru incepatori) [Regim de access] – https://rstforums.com/forum/topic/77289-vulnerabilitati-web-pentru-incepatori/
`

[13] Vulnerabilitati Web si securizarea acestora [Regim de access] – http://dgaspcsm.ro/Vulnerabilitati%20Web%20si%20securizarea.pdf

[14] Securitatea aplicaţiilor web cele mai întâlnite vulnerabilităţi/atacuri şi metode de apărare împotriva lor [Regim de access] – http://www.worldit.info/articole/securitatea-aplicatiilor-web-a-cele-mai-intalnite-vulnerabilitatiatacuri-si-metode-de-aparare-impotriva-lor/

[15] Tipuri de atacuri asupra aplicatiilor web | Securitatea Informatica [Regim de access] – http://www.securitatea-informatica.ro/audit-securitate/vulnerabilitati/tipuri-de-atacuri-asupra-aplicatiilor-web/

[16] Introducere în Angular 2 [Regim de access] – https://www.todaysoftmag.ro/article/2161/introducere-in-angular-2