
Web Sistemaren Pentest

Informazio Sistemen Segurtasuna Kudeatzeko Sistemak

Egileak: *Julen Fuentes eta Peio Llano*

Titulazioa: *Kudeaketaren eta Informazio Sistemen Informatikaren
Ingeniaritzako Gradua*

Kurtsoa: *3. (1. Lauhilekoa)*

2021ko abenduaren 19a

Gaien Aurkibidea

Portada	2
1 Sarrera	2
2 Auditoriaren jarraibideak	3
2.1 Beharrezko aplikazioaren instalazioa	3
2.2 Auditoria burutzea	4
3 Hasierako egoera	6
4 Konpondutako ahuleziak	7
4.1 SQL Injection	7
4.2 Komandoen sekuentzia leku gurutzatuetan	7
4.3 Pasahitzen hash eta salt (Datuen azaltzea)	8
4.4 Kontu korronteen zifraketa (Datuen azaltzea)	9
4.5 Segurtasun-konfigurazio okerra (Pasahitza ahulak ekidin)	10
4.6 Autentikazioa galtzea	10
4.7 Logging eta monitorizazio	11
5 Amaierako egoera	12
5.1 Azken auditoriaren kalteberatasunak	12
5.1.1 X-Frame-Options Header Not Set	12
6 Ondorioak (Hasiera eta amaiera)	13
7 Bibliografia	13

1 Sarrera

Geneukan web sistema funtzionala zen arren, segurtasunari dagokionez ahulezia asko zituen. Horregatik, lan honen helburua gure 1. lanari web-auditoria bat egitea da, pentesting-tresna ezagunenetako bat erabiliz (ZAP1) eta OWASP txostenaren ahultasun batzuk konpontzea.

Horrez gain, adierazitako ahultasunak ulertu, aztertu eta konpondu behar izango ditugu, hauek konpontzeko kodean egindako aldaketak azalduz. Ikerketaren osoaren ondorioz kalte berriak ezagutu ditugu eta honi esker etorkizunean hauek zuzenean saihestu ahal izango ditugu kodea inplementatu eta ondoren aldatzen ibili beharrean.

Oharrak:

- Erabili daitezken erabiltzaileak
 - Email:ramon@gmail.com .Pasahitza: RamonSo12*
 - Email:zorroz@gmail.com .Pasahitza: JokinZo33@
 - Email: jonbas@gmail.com .Pasahitza: JonBas88&
- Hasi baino lehen komando lerroan hurrengo komandoa exekutatu (php direktorioan): «sudo chmod 777 login.txt»

2 Auditoriaren jarraibideak

2.1 Beharrezko aplikazioaren instalazioa

Gure web-orriaren auditoria burutu ahal izateko, lehenik eta behin **orrialde honetara** jo behar izan dugu beharrezko aplikazioa, hots, OWASP ZAP deskargatzeko.

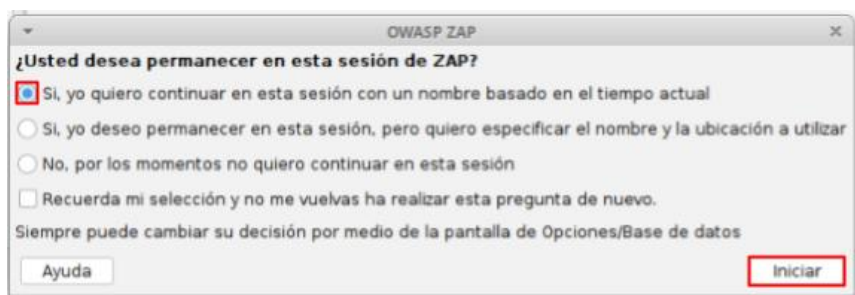
ZAP 2.11.1		
Windows (64) Installer	183 MB	Download
Windows (32) Installer	183 MB	Download
Linux Installer	188 MB	Download
Linux Package	186 MB	Download
MacOS Installer	213 MB	Download
Cross Platform Package	284 MB	Download
Core Cross Platform Package	55 MB	Download

Irudia 1: OWASP ZAP deskargatzeko web-orrialdea

Ondoren, proiektua garatzeko linux sistema eragilea erabili dugunez, «Linux Installer» aukerako «Download» botoia aukeratuko dugu eta honela .sh erako fitxategi bat deskargatzen hasiko da. Behin deskarga amaituta, aplikazioa instalatu egingo dugu. Horretarako, linux-en agindu-kontsola irekiko dugu eta deskargen karpetara mugituko gara, .sh fitxategia exekutatzeke baimenak emango dizkiogu geure buruari (chmod u+x deskargatutakoFitxategiarenIzena.sh) eta azkenik, instalatzailea egikarituko dugu (sudo ./deskargatutakoFitxategiarenIzena.sh). Azken agindu honen ostean, instalatzaile bat irekiko zaigu eta edozein aplikazio instalatzerakoan jarraitu beharreko urrats arruntak jarraituko dira.

2.2 Auditoria burutzea

Behin OWASP ZAP instalatuta hau ireki egingo dugu eta honako leihoa agertuko zaigu:



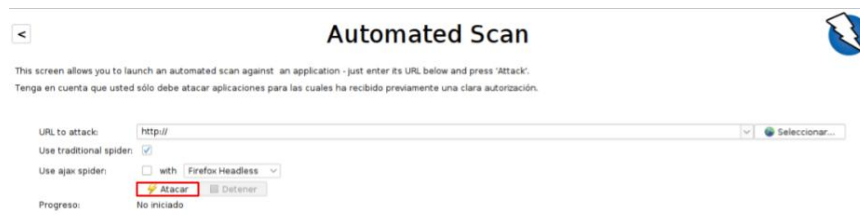
Irudia 2: OWASP ZAP aplikazioko hasierako leihoa

Honetan lehenengo aukera hautatuko dugu eta «Iniciar» botoia sakatuko dugu. Ondoren, honako hau aurkituko dugu:



Irudia 3: OWASP ZAP aplikazioak analisia egiteko eskaintzen dituen aukerak

Honetan aldiz, «Automated Scan» botoia sakatuko dugu eta hurrengo leihohan ageri diren eremua bete beharko dugu: URL to attack (gure kasuan <http://localhost:81>).



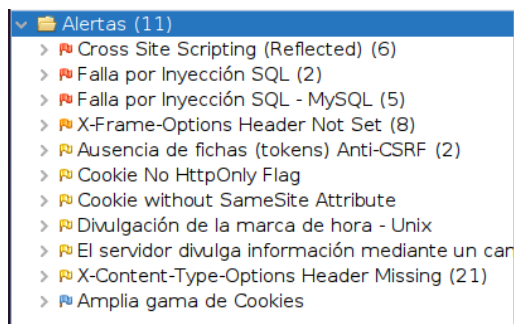
The screenshot shows the 'Automated Scan' interface of OWASP ZAP. At the top, there is a title bar with a back button, the title 'Automated Scan', and a lightning bolt icon. Below the title bar, a message states: 'This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'. Tenga en cuenta que usted sólo debe atacar aplicaciones para las cuales ha recibido previamente una clara autorización.' The main form contains a 'URL to attack:' field with 'http://', a 'Use traditional spider:' checkbox which is checked, and a 'Use ajax spider:' section with a 'with' dropdown set to 'Firefox Headless'. Below this, there are two buttons: 'Atacar' (highlighted with a red box) and 'Detener'. At the bottom, a 'Progreso:' section shows 'No iniciado'.

Irudia 4: OWASP ZAP aplikazioan analisisa egiterakoan bete beharrekoa

Behin eremua beteta, analisi egiteari ekingo diogu «Iniciar» botoia sakatuta eta analisisa amaitu arte itxarotea izango da amaierako urratsa.

3 Hasierako egoera

Hasieran, 3 alerta gorri izan genituen, hala ere horietaz gain beste hainbat ahulezia geneuzkan web sisteman zehar. Gainera, beste arrisku maila batzuetako alertak izan genituen. Alde batetik, SQL injekzioak genituen, egia esan inplementatuta zegoen JavaScript fitxategiarekin, hein handi batean injekzioa ekiditen zen baina ez zen modurik onena hori egiteko. Halaber, XSS motako erasorik onartzen genituela ikus genezakeen. Bestalde, indarrezko erasoak edo pasahitzak zifratu gabe gordetzea bezalako ahulezia geneukan eta ez genuen login-ei buruzko informazioa gordetzen. Hori dela eta, gure web sistema erabilgarria zen arren ahuleziek web sistematik informazio sentikorra edonork ikusteko aukera edo norbaiten izenean zer edo zer egiteko aukera zegoen eta erabiltzaileak aldentzea suposatuko litzake.

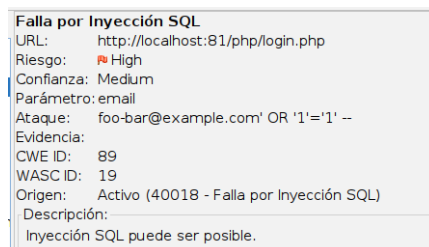


Irudia 5: Hasierako egoera

4 Konpondutako ahuleziak

4.1 SQL Injection

SQL injekzioa intrusio-kodea infiltratzeko metodo bat da, aplikazio batean dagoen ahultasun informatikoaz baliatzen da datu-basean eragiketak egin ahal izateko. Gure kasuan, erabiltzen ditugun SELECT kontsultetan testu zehatz bat sartuz injekzioa egiteko aukera zegoela esaten zuen OWASP ZAP aplikazioak egindako auditoriak. Aipatu bezala, frogak egin ondoren injekzioa ezinezkoa zela konprobatu genuen. Izan ere, gure JavaScript fitxategian zelaia konprobatzen dira eta horrek nahi ez ditugun testu formatuak sartzea ekiditen du.



Irudia 6: SQL Injection-aren auditoria

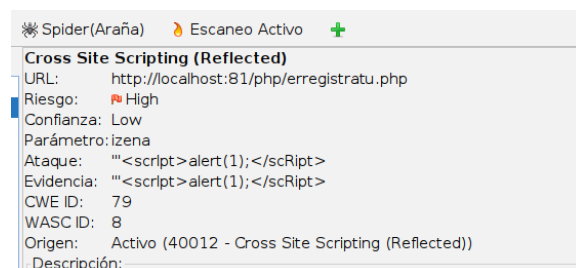
Nahiz eta gehienetan erregistratu.js fitxategiari esker babestuta egon, erabilitako modua ez zen hoberena (eta kasu batzuetan nahikoa) eta kontsultak egiteko modua aldatu dugu, injekzioa saihestuz. Geneukan arazorik handiena, parametroak kontsultan bertan sartzen genituela zen. Alde batetik kontsulta eta bestetik parametroak izatearen arazoa konpondu dugu, kontsultak berak badakielako zer diren parametroak eta zer prestatutako kontsultaren testua.

```
34 $query = $conn->prepare("SELECT * FROM `bezeroa` WHERE `email`  
    ↳ = ?; ");  
35 $query->bind_param('s', $email);  
36 $query->execute();  
37  
38 $result = $query->get_result();
```

4.2 Komandoen sekuentzia leku gurutzatueta

XSS (Cross Site Scripting) zaugarritasun informatiko arruntenetarikoa da, web-sistemak kaltetu egiten dituenak. Honek, aginduak sartzea ahalbidetzen die erasotzaileei datuak eskatzen diren eremuetan, adibidez. Agindu hauen bitartez, erasotzaileek kode-sekuentzia arrikutsuak sar ditzakete gure web-sisteman eta erabiltzaile arrunten web-nabigatzaileetan malware-ak instala ditzakete.

Gure proiektuari dagokionez, XSS zaugarritasuna erregistratu.php fitxategian aurki dezakegu, erregistratu.html fitxategiko formulariotik datuak jasotzen diren zatian, hain zuzen ere. Arazo honen antzematea hurrengo irudian ikus dezakegu:



Irudia 7: XSS erasoaren auditoria

Arazoa konpontzeko egin behar den gauza bakarra, jasotzen diren datuak PHP-ren htmlspecialchars() funtzioarekin tratatzea da. Izan ere, funtzio honek parametro gisa pasatzen diogun string-a hartu eta karaktere guztiak HTML entitate bihurtzen ditu eta hau gure kasuan baliagarria izan da erregistratu.html fitxategiko izen-abizenen eremurako, adibidez. Honetan (htmlspecialchars() funtzioa erabili gabe), «<script>alert(1)</script>» idatzita eta gainontzeko datuak era egokian beteta, erabiltzaile berria sortu eta alert(1) funtzioa egikaritu egiten da, eta are okerrago, kode askoz arriskutsuagoa duen script bat egikaritu liteke. Aipatu berri den funtzioa erabilita (azpian dagoen kodean ikus daitekeen bezala), «<>» sinboloa adibidez, «<» karaktere katean bihurtuko litzateke eta horrela funtzioa ez da egikarituko.

```

23     $izena = htmlspecialchars($_POST['izena']);
24     $nan = htmlspecialchars($_POST['nan']);
25     $tlf = htmlspecialchars($_POST['tlf']);
26     $jaiotze = htmlspecialchars($_POST['jaiotze']);
27     $email = htmlspecialchars($_POST['email']);
28     $pasahitza = htmlspecialchars($_POST['pasahitza']);

```

4.3 Pasahitzen hash eta salt (Datuen azaltzea)

Hash funtzio kriptografikoa algoritmo matematiko bat da, eta datuen edozein bloke arbitrario luzera finkoko karaktere berri batean bihurtzen du. Hash-aren indarra alderantzizko funtziorik ez duela da. Kriptografian, gatza (ingelesezt, salt) ausazko bitak dira, eta horiek gakoaren funtzio deribatzaile bezala erabiltzen dira. Beste sarrera pasahitza izaten da normalean.

Horiek biak erabiliz, erabiltzaileen pasahitzak datu-basean zifratuta egongo dira eta ezin izango dira deszifratu. Lehenik eta behin, 16 karaktereko

```
$charset = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
↳ XYZ0123456789/\\ [{ } \\'";:?.>,<!@#%~^&*()-_+=|'';
$randStringLength = 16;

$randString = "";
for ($i = 0; $i < $randStringLength; $i++) {
    $randString .= $charset[mt_rand(0, strlen($charset) - 1)];
}

$pasahitzaHash = $randString.$pasahitza.$randString;
$pasahitzaHash = hash("sha512", $pasahitzaHash);
```

Erabiltzailearen datu pertsonalen artean banku-kontuaren zenbaki bat gehituko dugu (2 hizki eta 22 digitu). Banku-kontuaren zenbaki hori enkriptatuta biltegiratu dugu datu-basean, eta behar izan dugunean desencriptatu egin dugu erabiltzaileari erakutsi ahal izateko. Honela, erabiltzailearen datu sentikorrek izan ahalko ditugu hauek era egoki batean gordeta.

9

4.5 Segurtasun-konfigurazio okerra (Pasahitza ahulak ekidin)

Indar gordinaren erasoak izan daitezkeen karaktere-konbinazio guztiak probatzea da, «erantzun zuzena» aurkitu arte. Prozesu horrek denbora asko behar izaten du, eta, beraz, hiztegiak eta pasahitz-zerrenda arruntak erabiltzen dira, hala nola, «qwerty» edo «123456». Hauek ekiditeko asmoz, erabiltzaileak sartzen duen pasahitza segurua izatera behartuko da. Bete beharreko baldintza pasahitzaren luzera 8 karakterekoa izatea, eta letra larriak, xeheak, zenbakiak eta karaktere bereziak izatea izango dira. Honela, gutxienezko segurtasun onargarria izatera behartuz.

```

95  function pasahitzaKonprobatu(pPasahitza) {
96      var minNumberOfChars = 8;
97      var maxNumberOfChars = 20;
98      var regularExpression =
99      ↪ /^(?=.*\d)(?=.*[!@#$%^&*~])(?=.*[a-z])(?=.*[A-Z]).{8,}$/;
100  alert("Hemendik pasa")
101  if(pPasahitza.length < minNumberOfChars || pPasahitza.length
102  ↪ > maxNumberOfChars){
103      alert('Pasahitzek ez dute formatu zuzena betetzen.');
```

```

104      return false;
105  }
106  else if(!regularExpression.test(pPasahitza)) {
107      alert('Pasahitzek ez dute formatu zuzena betetzen,
108      ↪ gutxienez zenbaki bat eta karaktere berezi bat izan
109      ↪ behar du (!@#$%^&*~).');
```

```

110      return false;
111  }
112  else {
113      return true;
114  }
115  }

```

4.6 Autentikazioa galtzea

Baimenik gabeko erabiltzaileak saio abandonatuen bidez sar ez daitezten, baimendutako jarduerarik gabeko gehieneko denbora gainditu ondoren eten egiten da hasitako saioa. Sistemak automatikoki itxiko du saioa, jarduerarik gabeko minutu baten ondoren. Erabiltzaileak berriz ireki behar izango du saioa bere informazioa ikusi nahi badu.

```

8      if(isset($_SESSION['tiempo'])) ) {
9
10         $inactivo = 60; //1min en este caso.
11
12         $vida_session = time() - $_SESSION['tiempo'];

```

```
13
14         if($vida_session > $inactivo)
15         {
16
17             session_unset();
18             session_destroy();
19             //Berbideratu.
20             echo"<script>alert('Jarduera eza dela eta, sesio
                ↳ itxi egin dar. Berriz sesioa
                ↳ hasi.')</script>","<meta http-equiv='refresh'
                ↳ content='0; url=../index.html' />";
21
22             exit();
23         } else {
24             $_SESSION['tiempo'] = time();
25         }
26
27     } else {
28
```

4.7 Logging eta monitorizazio

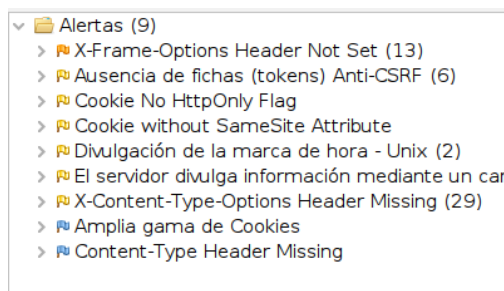
Garrantzi handia dauka erabiltzaile bakoitza noiz saiatu den sesioa hasten eta arrakasta izan duen edo ez. Honela indarrezko erasorik izan dugun jakingo dugu, zein erabiltzailetan eta noiz. Gainera, jokabide susmagarriak detektatzeko (erabiltzaile edo administratzailearen bat sisteman sartzen, behar ez dutenean) baliogarria izango da edo kasu batzuetan, ezinbestekoa da informazio hori gordetzea, DBLO (Datu Pertsonalak Babesteko Lege Organikoa) dela eta.

```
66     $file = fopen("login.txt", "a");
67     fwrite($file, "ZUZENA --> NOR: $email || NOIZ? ".
        ↳ date(DATE_RFC2822) . PHP_EOL);
68     fclose($file);
```

```
74     $file = fopen("login.txt", "a");
75     fwrite($file, "OKERRA --> NOR: $email || NOIZ? ".
        ↳ date(DATE_RFC2822) . PHP_EOL);
76     fclose($file);
```

5 Amaierako egoera

Amaieran, 0 alerta gorri izan genituen, hala ere beste hainbat ahulezia dauzkagu web sisteman zehar. Esan bezala, «Medium» arrisku mailako alerta bat izaten jarraitzen dugu, baita «Low» arrisku mailako hainbat alerta, auditore automatikoak hartzen ez dituen hainbat ahultasunez gain.



Irudia 8: Amaierako egoera

5.1 Azken auditoriaren kalteberatasunak

Aipatutako maila txikiagoko arriskuak tratatuak izan daitezke, hala ere kasu honetan arrisku handiagoei eman diegu garrantzia. Dena den, ulertu eta inplementatu gabeko soluzioa eman diegu horiei.

5.1.1 X-Frame-Options Header Not Set

Clickjacking Interneteko erabiltzaileak engainatzeko teknika maltzurra da, xaloak diren webguneetan informazio konfidentziala lortzeko modua izanik. Modu honetan gure web sistemaz baliatu daitezke gure erabiltzailearen baten informazioa lortzeko. Hori saihesteko, X-Frame Options goiburuak orria frame edo iframe batean ireki ez dadin balio du. Gauzak honela, arazoa konpontzeko irtenbideetako bat goiburu horren bidez *SAMEORIGIN* jartzen badugu, orrialde bat erabil dezakezu marko batean hura sartzen duen tokia zerbitzatzen duen leku bera den bitartean.

6 Ondorioak (Hasiera eta amaiera)

Behin proiektuan eskatutako aldaketak eginda, gure web-sistemaren segurtasunari dagokionez aldea dagoela ikus dezakegu. Hasteko, OWASP aplikazioa erabiliz egindako hasierako eta amaierako analisiak alderatzen baditugu, alerta arriskutsuenak, XSS eta SQL injekzioa, alegia, desagertu egin direla ikus dezakegu. Gainera, egindako analisiak antzeman ez dituen zaurgarritasun batzuk konpondu ditugu sistemaren funtzionamendu hobetzeko, besteak beste, pasahitzen segurtasuna areagotzea (pasahitzak ezaugarri batzuk betetzera derrigortuz eta hauek datu basean gordetzerakoan era seguruago batean eginez, gatza erabilita), jarduera ezaren ondoriozko deskonexioa edota web-sistemara egindako sarbide guztiak gordetzea.

Egindako lan guzti hau baliagarria zaigu web-sistema bat osatzerakoan jazo daitezkeen kalteberatasunak detektatzeko eta horiek kontuan ez hartzeak eragin ditzakeen kalteak ikusteko. Honi esker, etorkizunean web-sistemaren bat osatzekotan, segurtasun arazo horiek ekiditeko eta sor daitezkeen zaurgarritasunak zein diren aztertzeke eta hauek konpontzeko gaitasuna izango dugu.

7 Bibliografia

Autentikazioa galtzea: <https://es.stackoverflow.com/questions/33750/como-cerrar-sesi%C3%B3n-despu%C3%A9s-de-un-tiempo-de-inactividad/33756>

Zifraketa simetrikoa (Kontu korronterako): <https://www.youtube.com/watch?v=D-d10rXX4nE>