Laborator de atac: Falsificarea cererilor intre situri (Cross-Site Request Forgery - CSRF)

(Aplicația Web: `Elgg`)

1 Scopul lucrării

Obiectivul acestei lucrări de laborator este să ajute studenții să înțeleagă atacul prin falsificarea cererii inter-sit (cross-site-request forgery - CSRF or XSRF). Un atac CSRF implică un utilizator victimă, un sit de încredere și un sit rău intenționat (malicious). Utilizatorul victimă are o sesiune activă la un sit de încredere și, simultan, vizitează un sit rău intenționat. Situl rău intenționat injectează o cerere HTTP pentru situl de încredere în sesiunea utilizatorului victimă cauzându-i astfel daune.

În această lucrare veți ataca o aplicație web de rețele sociale folosind atacul CSRF. Aplicația cu sursă deschisă, pentru rețele sociale, se numește `Elgg` și a fost instalată deja în VM. `Elgg` are contramăsuri împotriva CSRF, dar le-am dezactivat pentru acest laborator.

Acest laborator acoperă următoarele subiecte:

- Atacul CSRF
- Contramăsuri pentru CSRF: jeton secret și cookie pentru același sit (Same-site cookie)
- Cererile HTTP GET și POST
- JavaScript și Ajax

2 Mediul folosit

Configurațiile pentru acest laborator sunt specifice pentru VM `Ubuntu` 16.04 din cauza configurărilor necesare, care sunt gata efectuate pe VM `Ubuntu` 16.04. rezumăm în această secțiune aceste configurații.

Aplicația web `Elgg` Folosim aplicația cu sursă deschisă numită `Elgg` în această lucrare. `Elgg` este o aplicație bazată pe web pentru rețele sociale. Este deja setată în imaginea `Ubuntu` preconstruită. Am creat și câteva conturi de utilizator pe serverul `Elgg`. Datele referitoare la conturi sunt în tabela următoare.

Utilizatorul	Numele de utilizator	Parola
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

Configurația DNS. Acest laborator folosește două situri de web, situl victimei și cel al atacatorului. Amândouă sunt setate în VM. URL-urile și directoarele lor sunt date mai jos:

```
Situl de web al atacatorului
    URL: http://www.csrflabattacker.com
    Director: /var/www/CSRF/Attacker/

Situl de web al victimei (Elgg)
    URL: http://www.csrflabelgg.com
    Director: /var/www/CSRF/Elgg/
```

Cele două URL-uri sunt accesibile doar din interiorul mașinii virtuale, deoarece am modificat fișierul `/etc/hosts` ca să mapeze numele de domeniu al fiecărui URL la adresa IP locală a mașinii (`127.0.0.1`). Puteți mapa orice nume de domeniu la o anume adresă folosind `/etc/hosts`. Spre exemplu, puteți mapa `www.example.com` la adresa IP locală adăugând următoarea intrare în `/etc/hosts`:

```
127.0.0.1 www.example.com
```

Dacă serverul de web și browserul rulează pe mașini diferite, atunci trebuie să modificați `/etc/hosts` pe mașina browser-ului în mod corespunzător pentru a mapa aceste nume de domeniu la adresa IP a serverului de web nu la `127.0.0.1`.

Configurația lui Apache. În imaginea VM preconstruită, am folosit serverul `Apache` pentru a găzdui toate siturile de web folosite în laborator. Se poate folosi caracteristica de găzduire virtuală din `Apache` pentru a găzdui câteva situri de web (sau URL-uri) pe aceeași mașină. Fișierul de configurare numit `000-default.conf` din directorul "`/etc/apache2/sites-available`" conține directivele necesare pentru configurare. În fișierul de configurare, fiecare sit de web are un bloc `VirtualHost` care specifică URL pentru situl de web și directorul din sistemul de fișiere care conține sursele sitului respectiv. Exemplele următoare arată cum se configurează un sit de web cu URL `http://www.example1.com` și un altul cu URL `http://www.example2.com`:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>
<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

Puteți modifica aplicația web accesând sursele din directoarele precizate. Spre exemplu, având configurația de mai sus, aplicația web `http://www.example1.com` se poate modifica prin modificarea surselor din directorul `/var/www/Example_1/`. După efectuarea unei schimbări în configurație, serverul `Apache` trebuie restartat. Folosiți comanda:

```
$ sudo service apache2 restart
```

3 Desfășurarea lucrării

Pentru sarcinile din laborator vom folosi două situri de web setate local în VM. Primul este situl vulnerabil `Elgg` accesibil la `www.csrflabelgg.com` în mașina virtuală. Cel de al doilea sit de web este cel rău intenționat al atacatorului folosit la atacarea `Elgg`. Acest sit d web este accesibil la `www.csrflabattacker.com` în mașina virtuală.

3.1 Sarcina 1: Observarea cererii HTTP

.

În atacurile de tipul Cross-Site Request Forgery, e nevoie să fie falsificate cererile HTTP. De aceea, trebuie să se știe cum arată o cerere HTTP legitimă și ce parametri folosește etc. Putem utiliza în acest scop un add-on al Firefox numit "`HTTP Header Live`". Scopul acestei sarcini este să vă obișnuiți cu această unealtă. Instrucțiuni detaliate asupra folosirii acestei unelte sunt date în secțiunea Ghid 4. Folosiți această unealtă pentru a captura o cerere HTTP GET și una POST în `Elgg`. În raport, identificați parametrii folosiți în aceste cereri, dacă există.

3.2 Sarcina 2: Atac CSRF folosind cererea GET

În cadrul acestei sarcini, avem nevoie de două persoane în rețeaua socială `Elgg`: Alice și Boby. Boby dorește să se împrietenească cu Alice, dar Alice refuză să îl adauge pe Boby la lista sa de prieteni `Elgg`. Boby decide să folosească atacul CSRF pentru a-și îndeplini scopul. El îi trimite lui Alice un URL (printr-un mesaj de email sau o postare în `Elgg`); Alice, curioasă, dă clic pe URL, fapt care o duce la situl de web al lui Boby: `www.csrflabattacker.com`. Pretindeți că sunteți Boby și descrieți cum puteți construi conținutul paginii de web astfel încât, de îndată ce Alice vizitează pagina, Boby este adăugat la lista de prieteni a lui Alice (presupunând că Alice are o sesiune activă cu `Elgg`).

Pentru a adăuga un prieten la lista victimei, avem nevoie să identificăm cum arată o cerere legitimă HTTP **Add Friend**, care este o cerere GET. Putem folosi unealta "`HTTP Header Live`" pentru a investiga. În cadrul acestei sarcini nu aveți voie să scrieți cod JavaScript pentru a lansa atacul CSRF. Treaba dvs. e să faceți ca atacul să aibă succed de îndată ce Alice vizitează pagina de web, fără a da vreun clic (sugestie: puteți folosi eticheta (engl. tag) `img` tag, care declanșează automat o cerere HTTP GET). `Elgg` a implementat o contramăsură pentru apărarea împotriva atacurilor CSRF. În cererile HTTP `Add-Friend`, puteți vedea că fiecare cerere include doi parametri cu aspect ciudat, `elgg ts` și `elgg token`. Acești parametri sunt folosiți de contramăsură, astfel încât, dacă ei nu conțin valori corecte, cererea nu va fi acceptată de `Elgg`. Am dezactivat contramăsura pentru acest laborator, așa că nu e nevoie să-i includeți în cererile falsificate.

3.3 Sarcina 3: Atac CSRF folosind cererea POST

După ce s-a adăugat la lista de prieteni a lui Alice, Bob vrea mai mult. El vrea ca Alice să declare în profilul ei "Boby is my Hero", astfel ca toți să afle despre asta. Lui Alice nu-i place de Boby, nici vorbă să pună o asemenea declarație în profilul ei.

O modalitate de a realiza atacul este să pună un mesaj în contul `Elgg` al lui Alice, în speranța că Alice va da clic pe URL-ul din mesaj. Acest URL o va duce pe Alice la situl rău intenționat al dvs. (adică al lui Boby) `www.csrflabattacker.com`, unde puteți lansa atacul CSRF.

Obiectivul atacului este modificarea profilului victimei. În particular, atacatorul are nevoie să falsifice o cerere pentru a modifica informația din profilul utilizatorului victimă al `Elgg`. Permiterea modificării profilului unui utilizator de către acesta este o caracteristică a `Elgg`. Dacă utilizatorii doresc să-și modifice profilele, ei navighează la pagina de profil a `Elgg`, completează u formular și apoi trimit formularul — prin trimiterea unei cereri POST — la scriptul de pe partea de server `/profile/edit.php`, care procesează cererea și realizează modificarea profilului.

Scriptul de pe partea de server `edit.php` acceptă atât cereri GET cât și POST, așa că puteți folosi același truc ca în Sarcina 3.2 pentru a realiza atacul.

Totuși, pentru această sarcină, vi se cere să folosiți o cerere POST. Mai precis, atacatorii (dvs.) trebuie să falsifice o cerere HTTP POST de la browser-ul victimei, atunci când victima vizitează situl rău intenționat.

Atacatorii trebuie să cunoască structura unei astfel de cereri. Puteți observa structura cererii , adică parametrii acesteia, făcând câteva modificări în profil și monitorizând cererea folosind unealta "HTTP Header Live". S-ar putea să vedeți ceva asemănător cu ceea ce urmează (spre deosebire de cererile HTTP GET, care adaugă parametri la șirurile URL, parametrii cererilor HTTP POST sunt cuprinse în corpul mesajului HTTP):

```
http://www.csrflabelgg.com/action/profile/edit

POST /action/profile/edit HTTP/1.1
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/elgguser1/edit
Cookie: Elgg=p0dci8baqrl4i2ipv2mio3po05
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 642
__elgg_token=fc98784a9fbd02b68682bbb0e75b428b&__elgg_ts=1403464813
&name=elgguser1&description=%3Cp%3Iamelgguser1%3C%2Fp%3E
&accesslevel%5Bdescription%5D=2&briefdescription= Iamelgguser1
&accesslevel%5Bbriefdescription%5D=2&location=US
```

După ce ați înțeles structura cererii, trebuie să puteți genera cererea din pagina de atac a dvs. folosind cod JavaScript. Pentru a vă ajuta să scrieți un asemenea program JavaScript program, furnizăm cod exemplu mai jos. Puteți folosi acest cod pentru a construi situl de web rău intenționat pentru atacuri CSRF. Este doar un exemplu și trebuie modificat pentru a-l face să funcționeze pentru atac.

```
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

function forge_post()
{
    var fields;
    // The following are form entries need to be filled out by attackers.
    // The entries are made hidden, so the victim won't be able to see them.
    fields += "<input type='hidden' name='name' value=' **** '>";
    fields += "<input type='hidden' name='briefdescription' value=' **** '>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]'
                   value='2'>";
    fields += "<input type='hidden' name='guid' value=' **** '>";

    // Create a <form> element.
    var p = document.createElement("form");

    // Construct the form
    p.action = "http://www.example.com";
```

```
    p.innerHTML = fields;
    p.method = "post";

    // Append the form to the current page.
    document.body.appendChild(p);

    // Submit the form
    p.submit();
}
// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

În linia care conține `accesslevel[briefdescription]` valoarea 2 setează nivelul de acces al uni câmp la "public". Acest lucru este necesar pentru că, în caz contrar, accesul va fi setat implicit la "private", astfel încât alții nu pot vedea acest câmp. Trebuie remarcat că, atunci când copiați-și-lipiți codul de mai sus dintr-un fișier PDF, caracterul de citare simplă poate deveni altceva (deși arată ca un apostrof). Aceasta va duce la erori de sintaxă. Înlocuirea tuturor caracterelor de acest fel cu apostrof tastat va rezolva aceste erori.

Întrebări. Pe lângă descrierea în detaliu a atacului, trebuie să răspundeți la următoarele întrebări:

- Întrebarea 1: Cererea HTTP falsificată are nevoie de id de utilizator al lui Alice (guid) pentru a funcționa corect. Dacă Boby o țintește anume pe Alice, înainte de atac, el poate găsi modalități ca să obțină id de utilizator al lui Alice's. Boby nu știe parola `Elgg` a lui Alice, așa că nu se poate loga în contul lui Alice pentru a obține informația. Descrieți cum poate Boby să rezolve această problemă.
- Întrebarea 2: Boby ar vrea să lanseze un atac asupra oricărei persoane care îi vizitează pagina de web. În acest caz, el nu știe cine îi vizitează pagina, apriori. Poate el lansa un atac CSRF pentru a modifica profilul `Elgg` al victimei? Explicați cum.

3.4 Sarcina 4: Implementarea unei contramăsuri pentru `Elgg`

`Elgg` are contramăsuri preconstruite pentru apărarea împotriva atacurilor CSRF. Le-am comentat pentru a face atacul să funcționeze. Nu e greu de apărat împotriva atacurilor CSRF și există câteva abordări uzuale:

- Metoda cu token secret: Aplicațiile web pot încorpora un jeton secret în paginile lor și toate cererile care vin de la aceste pagini poartă acest jeton. Deoarece cererile între situri nu pot obține acest jeton, aceste cereri false vor fi identificate ușor de server.
- Metoda cu antet de la expeditor (engl. referer header): aplicațiile web pot verifica pagina de origine și utilizând antetul de la expeditor. Totuși, din motive de intimitate (engl. privacy), acest antet ar putea fi deja filtrat pe partea de client.

Aplicația web `Elgg` folosește metoda cu jeton secret. Aceasta încorporează doi parametri, `elgg ts` și `elgg token` în cerere pe post de contramăsură la atacul CSRF. Cei doi parametri sunt adăugați la corpul mesajului HTTP pentru cererile POST și la șirul URL pentru cererile HTTP GET.

Elgg **secret-token și ștampila de timp** (engl. timestamp) **în corpul cererii.** Elgg adaugă jetonul de securitate și ștampila de timp la toate acțiunile utilizatorului care urmează să fie efectuate. Codul HTML care urmează este prezent în toate formularele în care este necesară acțiunea utilizatorului. Acest cod adaugă doi parametri ascunși, elgg ts și elgg token la cererea POST:

```
<input type = "hidden" name = "__elgg_ts" value = "" />
<input type = "hidden" name = "__elgg_token" value = "" />
```

elgg ts și elgg token sunt generate de modulul php views/default/input/securitytoken și adăugate la pagina web. Bucata de cod de mai jos arată cum sunt adăugate s=dinamic la pagina web.

```
$ts = time();
$token = generate_action_token($ts);

echo elgg_view('input/hidden', array('name' => '__elgg_token', 'value' =>
$token));
echo elgg_view('input/hidden', array('name' => '__elgg_ts', 'value' => $ts));
```

Elgg adaugă jetonul de securitate și ștampila de timp și la JavaScript care poate fi accesat de

```
elgg.security.token.__elgg_ts;
elgg.security.token.__elgg_token;
```

Jetonul de securitate Elgg este o valoare de dispersie (un rezumat de mesaj cu MD5) a valorii secrete (obținute din baza de date), a ștampilei de timp, a sessionID a utilizatorului și a unui șir de sesiune generat aleator, pentru apărarea împotriva atacului CSRF. Codul de mai jos arată cum se face generarea jetonului secret în Elgg.

```
function generate_action_token($timestamp)
{
    $site_secret = get_site_secret();
    $session_id = session_id();
    // Session token
    $st = $_SESSION['__elgg_session'];
    if (($site_secret) && ($session_id))
    {
        return md5($site_secret . $timestamp . $session_id . $st);
    }
    return FALSE;
}
```

Funcția PHP session_id() se folosește pentru a obține sau a seta ID de sesiune pentru sesiunea curentă. Bucata de cod de mai jos arată și generarea șirului aleator pentru o sesiune session_elgg separat de Session ID public al utilizatorului.

```
......
// Generate a simple token (private from potentially public session id)
if (!isset($_SESSION['__elgg_session'])) {
$_SESSION['__elgg_session'] =
ElggCrypto::getRandomString(32,ElggCrypto::CHARS_HEX);
........
```

Validarea jetonului secret al `Elgg` . Aplicația web `Elgg` validează jetonul și ștampila de timp generate pentru a apăra împotriva atacului CSRF. Fiecare acțiune a utilizatorului apelează funcția de validare a jetoanelor de acțiune, iar funcția respectivă validează jetoanele. Dacă jetoanele nu sunt prezente sau nu sunt valide, acțiunea va fi refuzată și utilizatorul va fi redirecționat.

Bucata de cod următoare prezintă funcția `validate_action_token`.

```
function validate_action_token($visibleerrors = TRUE, $token = NULL, $ts =
NULL)
{
    if (!$token) { $token = get_input('__elgg_token'); }
    if (!$ts) {$ts = get_input('__elgg_ts'); }
    $session_id = session_id();
    if (($token) && ($ts) && ($session_id)) {
        // generate token, check with input and forward if invalid
        $required_token = generate_action_token($ts);
        // Validate token
        if ($token == $required_token) {
            if (_elgg_validate_token_timestamp($ts)) {
                // We have already got this far, so unless anything
                // else says something to the contrary we assume we're ok
                $returnval = true;
                ......
            }
            else {
                ......
                register_error(elgg_echo('actiongatekeeper:tokeninvalid'));
                ......
            }
            ......
        }
```

Activați contramăsura. Pentru a activa contramăsura, navigați în directorul `/var/www/CSRF/Elgg/vendor/` și găsiți funcția `gatekeeper()` în fișierul `ActionsService.php`. În funcția `gatekeeper()` comentați instrucțiunea "`return true;`" așa cum se arată în comentariul din cod.

```
public function gatekeeper($action) {
    //SEED:Modified to enable CSRF.
    //Comment the below return true statement to enable countermeasure
    return true;
    ......
}
```

Sarcină: După ce ați activat contramăsura de mai sus, încercați din nou atacul CSRF și descrieți ce observați. Arătați jetoanele secrete din cererea HTTP capturată folosind unealta de inspecție HTTP a Firefox.

Explicați de ce atacatorul nu poate trimite aceste jetoane secrete în atacul CSRF. Ce îi împiedică să găsească jetoanele secrete pentru atacul CSRF din pagina de web?

4 Ghid

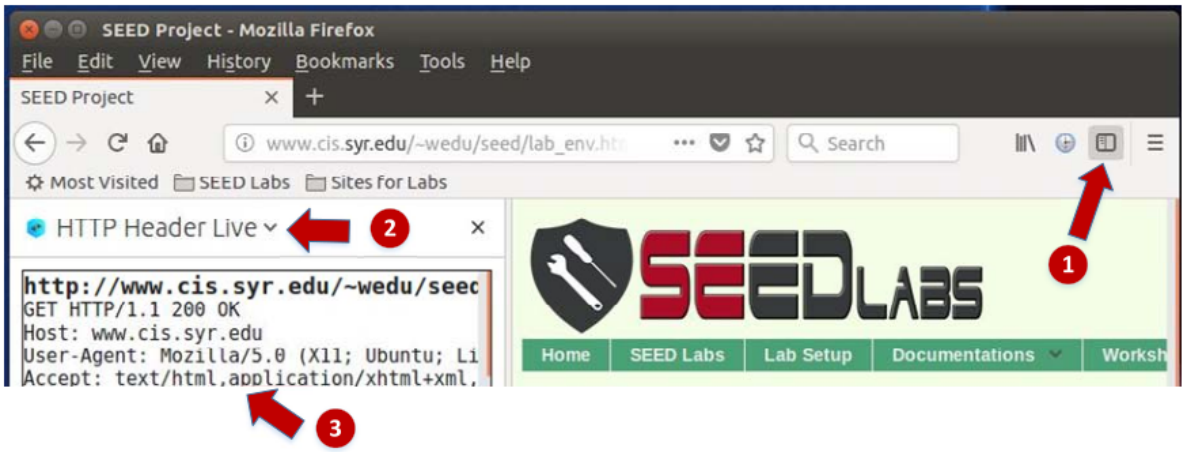


Figura 1: Activarea add-on HTTP Header Live

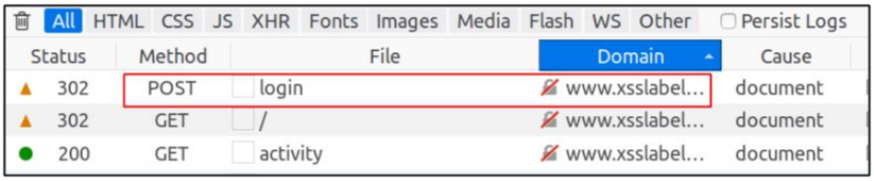


Figura 2: Cererea HTTP așa cum se vede în unealta Web Developer Network

4.1 Folosirea add-on "HTTP Header Live" pentru inspecția antetelor HTTP

Versiunea de Firefox (versiunea 60) din mașina noastră virtuală Ubuntu 16.04 VM nu suportă add-on `LiveHTTPHeader`, folosită în VM Ubuntu 12.04. Se folosește în loc noul add-on numit "`HTTP Header Live`". Instrucțiunile de activare și utilizare a acestei unelte add-on sunt prezentate în figura 1. Trebuie doar să dați click pe icoana marcată cu ① și va apărea o bară laterală pe stânga. Asigurați-vă că este selectat `HTTP Header Live` în poziția marcată cu ② Apoi dați click pe orice legătură dintr-o pagină de web și toate cererile HTTP declanșate vor fi capturate și afișate în zona din bara laterală marcată cu ③. Dacă dați clic pe orice cerere HTTP, va apărea o fereastră pop-up pentru a afișa cererea HTTP aleasă. Din păcate există un bug în această unealtă (este încă în dezvoltare) ; nu se va afișa nimic în fereastra pop-up dacă nu-i schimbați dimensiunea. (Se pare că re-desenarea nu este declanșată automat la afișarea ferestrei, dar re-dimensionarea va declanșa re-desenarea.)

4.2 Folosirea uneltei Web Developer pentru a inspecta antetele HTTP

Există o altă unealtă furnizată de către Firefox care poate fi destul de utilă la inspectarea antetelor HTTP. Unealta este `Web Developer Network Tool`. În această secțiune vom descrie câteva dintre caracteristicile importante ale acestei unelte. Web Developer Network Tool poate fi activată prin următoarea navigare:

```
Clic pe meniul din dreapta-sus al Firefox's top --> Web Developer --> Network
or
Clic pe meniul "Tools" --> Web Developer --> Network
```

Vom folosi ca exemplu pagina de login din `Elgg`. Figura 2 prezintă unealta în momentul în care arată cererea HTTP POST folosită pentru login.

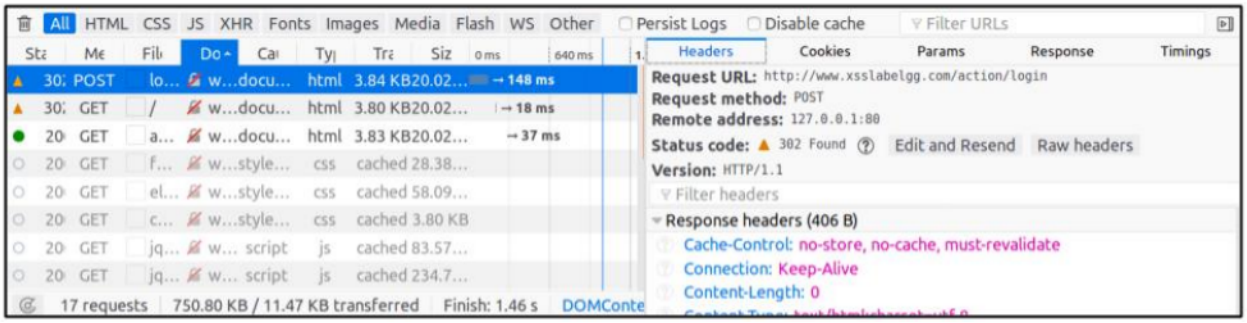
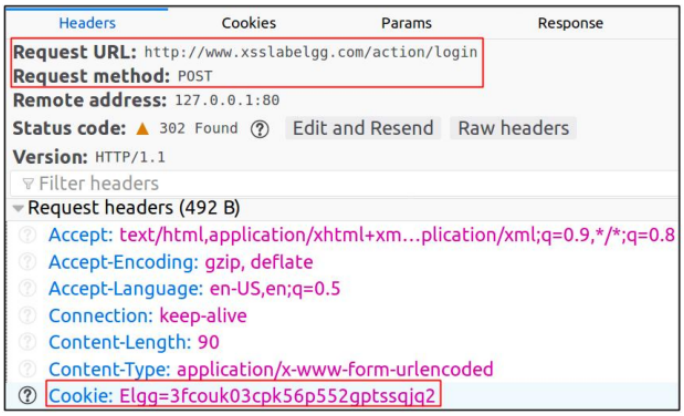
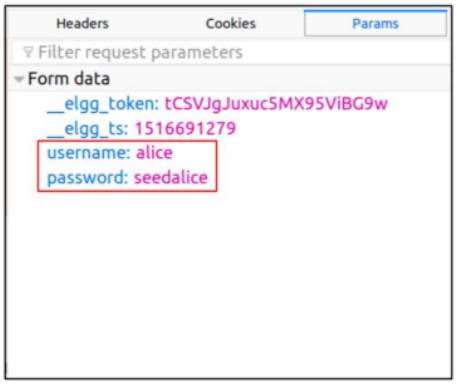


Figura 3: Cererea HTTP și detaliile cererii în două panouri.



(a) Antetele cererii HTTP



(b) Parametrii cererii HTTP

Figura 4: Antetele HTTP și parametrii lor

Pentru a vedea detaliile cererii, putem da click pe o cerere HTTP anume și unealta va arăta informațiile în două panouri (cf. figura 3).

Detaliile cererii vor fi vizibile în panoul din dreapta.

Figura 4(a) prezintă detaliile cererii de login în tab-ul `Headers` (detaliile includ URL, metoda de cerere și cookie). Se pot observa atât antetul de cerere cât și cel de răspuns în panoul din dreapta. Pentru a verifica parametrii implicați într-o cerere HTTP putem folosi tab-ul `Params`. Figura 4(b) prezintă paramerul trimis în cererea de login spre `Elgg`, inclusiv numele de utilizator și parola. Unealta se poate utiliza pentru a inspecta cererile HTTP GET asemănător cu inspecția cererilor HTTP POST.

Font Size. Dimensiunea implicită a fontului din fereastra Web Developer Tools este destul de mică. Se poate crește prin click oriunde în fereastra Network Tool și utilizarea butoanelor `Ctrl` și `+`

4.3 Depanarea JavaScript

Avem nevoie și să depanăm cod JavaScript. Developer Tool din Firefox ne poate ajuta și pentru depanarea codului JavaScript. Ea poate să evidențieze precis locurile în care apar erori. Următoarele instrucțiuni arată cum se activează această unealtă de depanare:

```
Clic pe meniul "Tools"--> Web Developer --> Web Console
or folosiți Shift+Ctrl+K.
```

O dată ce suntem în consola web, clic pe tab-ul `JS`. Clic pe săgeata în jos din dreapta `JS` pentru a vă asigura că există marcată cutia `Error`. Dacă doriți să vedeți și mesajele de avertizare, clic și pe cutia `Warning`. Cf. Figura 5.

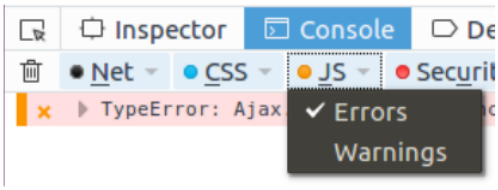


Figura 5: Depanarea codului JavaScript – activarea afișării erorilor

Dacă există erori în cod se va afișa un mesaj în consolă. Linia cu eroare apare în partea dreaptă a mesajului de eroare . Clic pe numărul liniei și veți ajunge în locul care a cauzat eroarea. Cf. figura 6

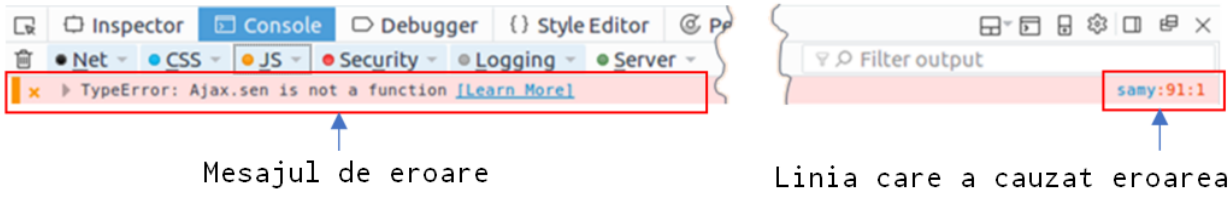


Figura 6: Depanarea codului Javascript – determinarea liniei cu eroarea

5 Trimiterea rezultatelor

Trebuie să trimiteți un raport detaliat în care să descrieți ce ați făcut și ce ați observat; și cum interpretați rezultatele. Rapoartele trebuie să conțină dovezi care să sprijine observațiile. Furnizați detalii privind utilizarea uneltelor add-on ale Firefox, Wireshark și/sau capturi de ecran. De asemenea dați explicații pentru observațiile interesante sau surprinzătoare.