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 Desfășurarea lucrării

2.1 Setarea mediului

2.1.1 Configurarea containerului și comenzi.

Vă rugăm să descărcați fișierul `Labsetup.zip` pe VM-ul dvs. de pe Moodle, să-l dezarhivați, să intrați în directorul `Labsetup` și să utilizați fișierul `docker-compose.yml` pentru a configura mediul de laborator. Explicații detaliate ale conținutul acestui fișier și toate fișierele `Dockerfile` implicate pot fi găsite din legătura la **manualul de utilizare - SEED Manual for Containers**. Dacă este prima dată când configurați un mediu de laborator SEED folosind containere, este foarte important să citiți manualul de utilizare.

În cele ce urmează, enumerăm câteva dintre comenzile utilizate frecvent legate de `Docker` și `Compose`. Cum vom folosi aceste comenzi foarte frecvent, am creat aliasuri pentru ele în fișierul `.bashrc` (în SEEDUBuntu 20.04 VM furnizat de noi).

```
$ docker-compose build # Build the container image
$ docker-compose up # Start the container
$ docker-compose down # Shut down the container
// Aliases for the Compose commands above
```

```
$ dcbuild # Alias for: docker-compose build
$ dcup # Alias for: docker-compose up
$ dcdown # Alias for: docker-compose down
```

Toate containerele vor rula în fundal. Pentru a rula comenzi pe un container, avem adesea nevoie să obținem un shell pe respectivul container. Mai întâi trebuie să folosim comanda "`docker ps`" pentru a afla ID-ul containerului și apoi să folosim "`docker exec`" pentru a lansa un shell pe acel container. Am creat aliasuri pentru ele în fișierul `.bashrc`.

```
$ dockps // Alias for: docker ps --format "{{.ID}} {{.Names}}"
$ docksh <id>// Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275 hostA-10.9.0.5
0af4ea7a3e2e hostB-10.9.0.6
9652715c8e0a hostC-10.9.0.7
$ docksh 96
root@9652715c8e0a:/#
// Note: If a docker command requires a container ID, you do not need to
// type the entire ID string. Typing the first few characters will
// be sufficient, as long as they are unique among all the containers.
```

Dacă întâmpinați probleme la configurarea mediului de laborator, vă rugăm să citiți secțiunea "Common Problems" din manual pentru a afla posibile soluții.

2.1.2 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 imaginile de container furnizate. Vom utiliza două containere, unul care rulează serverul de web (`10.9.0.5`) și un altul care rulează baza de date MySQL (`10.0.9.6`). Adresele IP ale acestor două containere sunt scrise în cod în diferite părți ale configurației, așa că *nu e voie să le schimbați*.

Containerul Elgg. Găzduim aplicația web Elgg folosind serverul web Apache. Configurarea site-ului web este inclusă în `apache_elgg.conf` în directorul `image_www` Elgg. Configurația specifică adresa URL pentru site-ul web și directorul în care este stocat codul aplicației web.

```
<VirtualHost *:80>
    DocumentRoot /var/www/elgg
    ServerName  www.seed-server.com
    <Directory /var/www/elgg>
        Options FollowSymlinks
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Containerul Attacker. Folosim un alt container (`10.9.0.105`) pentru mașina atacatorului, care găzduiește un site web rău intenționat. Configurația Apache pentru acest site web este listată în următoarele:

```
<VirtualHost *:80>
    DocumentRoot /var/www/attacker
    ServerName  www.attacker32.com
</VirtualHost>
```

Deoarece trebuie să creăm pagini web în interiorul acestui container, pentru comoditate, precum și pentru păstrarea paginile pe care le-am creat, am montat un director (`Labsetup/atacker` pe VM-ul de găzduire) în directorul `/var/www/attacker` din container, care este directorul `DocumentRoot` din configurația noastră Apache. Prin urmare, paginile web pe care le punem în directorul `attacker` de pe VM vor fi găzduite de atacator site-ul web. Am plasat deja câteva schelete de cod în interiorul acestui director.

Configurare DNS. Accesăm site-ul web Elgg, site-ul atacatorului și site-ul de apărare folosind adresele URL respective. Trebuie să adăugăm următoarele intrări în fișierul `/etc/hosts`, astfel încât aceste nume de gazdă să fie mapate la adresele lor IP corespunzătoare. Trebuie să utilizați privilegii de supervizor pentru a modifica acest fișier (folosind `sudo`). Trebuie remarcat faptul că este posibil ca aceste nume să fi fost deja adăugate în fișier datorită altor laboratoare. Dacă sunt mapate la adrese IP diferite, vechile intrări trebuie eliminate.

```
10.9.0.5 www.seed-server.com
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com
```

Baza de date MySQL. Containerele sunt de obicei de unică folosință, așa că odată ce sunt distruse, toate datele din interiorul lor se pierd. Pentru acest laborator, dorim să păstrăm datele în baza de date MySQL, astfel încât să nu pierdem munca noastră când închidem containerul. Pentru a realiza acest lucru, am montat directorul `mysql_data` pe mașina gazdă (în interiorul `Labsetup`, acesta va fi creat după ce containerul MySQL rulează o dată) la directorul `/var/lib/mysql` din interiorul containerului MySQL. Acest director este locul în care MySQL își stochează baza de date. Prin urmare, chiar dacă containerul este distrus, datele din baza de date sunt încă păstrate. Dacă doriți să porniți cu o bază de date curată, puteți elimina acest director:

```
$ sudo rm -rf mysql_data
```

Conturi de utilizator. Am creat mai multe conturi de utilizator pe serverul Elgg; numele de utilizator și parolele sunt date în tabelul 1

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

Tabela 1: Numele de utilizatori și aprolele lor în Elgg

2.2 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 HTTP POST în Elgg. În raport, identificați parametrii folosiți în aceste cereri, dacă există.

2.3 Sarcina 2: Atac CSRF folosind cererea GET

În cadrul acestei sarcini, avem nevoie de două persoane în rețeaua socială Elgg: Alice și Samy. Samy dorește să se împrietenească cu Alice, dar Alice refuză să îl adauge pe Samy la lista sa de prieteni Elgg. Samy 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 Samy: www.attacker32.com. Pretindeți că sunteți Samy și descrieți cum puteți construi conținutul paginii de web astfel încât, de îndată ce Alice vizitează pagina, Samy 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 succes 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.

2.4 Sarcina 3: Atac CSRF folosind cererea POST

După ce s-a adăugat la lista de prieteni a lui Alice, Samy vrea mai mult. El vrea ca Alice să declare în profilul ei "Samy is my Hero", astfel ca toți să afle despre asta. Lui Alice nu-i place de Samy, nici vorbă să pună o asemenea declarație în profilul ei. Samy planuieste să folosească un atac CSRF pentru a-si realiza scopul.

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 malefic al dvs. (adică al lui Samy) www.attacker32.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 un 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 **??** 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 (cf. conținutul dintre cele două simboluri ❌):

```
http://www.seed-server.com/action/profile/edit

POST /action/profile/edit HTTP/1.1
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv: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.seed-server.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.

Listing 1: Exemplu de cod Javascript

```
<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'>";                                          ❶
    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 ❶ se setează nivelul de acces al unui 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 Samy o ține anume pe Alice, înainte de atac, el poate găsi modalități ca să obțină id de utilizator al lui Alice's. Samy nu știe parola `Elgg` a lui Alice, așa că nu se poate loga în contul lui Alice pentru a obține informația. Descrieți cum poate Samy să rezolve această problemă.
- Întrebarea 2: Samy 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 Apărarea

Inițial, pentru a se apăra împotriva atacurilor CSRF, majoritatea aplicațiilor web pot încorpora un *jeton*[1] *secret* în paginile lor. Toate cererile care provin din aceste pagini trebuie să aibă acest simbol, sau vor fi considerate ca o solicitare între site-uri și vor fi considerate ca cerere între situri, care nu au același privilegiu ca și cererile de pe același site. Mai recent majoritatea browserelor au implementat un mecanism numit *SameSite cookie*, menit să simplifice implementarea contramăsurilor CSRF. Vom experimenta cu ambele metode.

3.1 Sarcina 4: Activarea contramăsurii pentru `Elgg`

Pentru a se apăra împotriva atacurilor CSRF, aplicațiile web pot încorpora un jeton[2] secret în paginile lor. Toate cererile care provin din aceste pagini trebuie să aibă acest simbol, sau vor fi

[1] (engl. token)
[2] (engl. token)

considerate ca o solicitare între site-uri și vor fi considerate ca cerere între situri, care nu au același privilegiu ca și cererile de pe același site. Atacatorii nu va putea obține acest simbol secret, așa că cererile lor vor fi ușor identificate ca solicitări între site-uri.

Elgg folosește această abordare cu jeton secret ca contramăsură preconstruită pentru a se apăra împotriva atacurilor CSRF. Noi au dezactivat contramăsurile pentru ca atacul să funcționeze. Elgg încorporează doi parametri `__elgg_ts` şi `__elgg_token` în cerere. Cei doi parametri sunt adăugați la corpul mesajului HTTP pentru cereri POST și la șirul URL pentru cererile HTTP GET. Serverul le va valida înainte de procesarea cererii.

Încorporarea jetonului secret și a mărcii de timp în paginile web. Elgg adaugă jetonul de securitate și marca de timp la toate cererile HTTP. Codul HTML următor este prezent în toate formularele în care este nevoie de acțiunea utilizatorului. Acestea sunt două câmpuri ascunse; la trimiterea formularului, cei doi parametri ascunși sunt adăugați la cerere:

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

Elgg atribuie valoarea jetonului de securitate și a mărcii de timp unor variabile JavaScript astfel ca acestea să poată fi accesate ușor din codul JavaScript de pe aceeasi pagină.

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

Jetonul secret și marca de timp sunt adăugate paginilor web ale Elgg de către modulul `vendor/elgg/elgg/v`
Fragmentul de cod de mai jos arată cum sunt adăugate în mod dinamic la paginile web.

```
$ts = time();
$token = elgg()->csrf->generateActionToken($ts);

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

Generarea jetonului secret. Jetonul de securitate Elgg este o valoare de dispersie (rezumat de mesaj md5) a valorii secretului sitului (din baza de date), marca de timp, ID-ul sesiunii utilizator și șirul de sesiune generat aleatoriu. Codul mai jos arată generarea jetonului secret în Elgg (în `vendor/elgg/elgg/engine/classes/Elgg/Security/Csrf.php`):

```
/**
* Generate a token from a session token (specifying the user),
* the timestamp, and the site key.
*/
public function generateActionToken($timestamp, $session_token = '') {
    if (!$session_token) {
        $session_token = $this->session->get('__elgg_session');
        if (!$session_token) {
            return false;
        }
    }

    return $this->hmac
    ->getHmac([(int) $timestamp, $session_token], 'md5')
    ->getToken();
}
```

Validarea jetonului secret. Aplicația web Elgg validează simbolul generat și marca temporală pentru apărare împotriva atacului CSRF. Fiecare acțiune a utilizatorului apelează funcția `validate` din `Csrf.php` care validează jetoanele. Dacă jetoanele nu sunt prezente sau invalide, acțiunea va fi refuzată, iar utilizatorul o va fi redirecționat. În configurarea noastră, am adăugat un `return` la începutul acestei funcții, dezactivând validarea.

```
public function validate(Request $request) {
        return; // Added for SEED Labs (disabling the CSRF countermeasure)
        $token = $request->getParam('__elgg_token');
        $ts = $request->getParam('__elgg_ts');
        ... (code omitted) ...
}
```

Sarcină: Activați contramăsura. Pentru a activa contramăsura, intrați în containerul Elgg și, în directorul `/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security`, eliminați instrucțiunea `return` din `Csrf.php`. Un editor simplu numit `nano` este disponibil din interiorul containerului.

După ce ați făcut schimbarea, repetați atacul din nou și vedeți dacă atacul are succes sau nu.

Indicați simbolurile secrete din cererile HTTP capturate.

Explicați de ce atacatorul nu poate trimite aceste jetoane secrete în atacul CSRF; ce îl împiedică să afle jetoanele secrete din pagina web?

Trebuie remarcat (important) că atunci când lansăm atacul de `edit-profile` în timp ce contramăsura este activată, încercarea eșuată va face ca pagina atacatorului să fie reîncărcată, ceea ce va declanșa cererea POST falsificată din nou. Acest lucru va duce la o altă încercare eșuată, astfel încât pagina va fi reîncărcată din nou și o altă cerere POST falsificată va fi trimisă. Această buclă infinită va încetini computerul. Prin urmare, după ce ați verificat că atacul a eșuat, închideți tab-ul pentru a opri bucla infinită.

3.2 Sarcina 5: Experimentarea cu metoda `SameSite` cookie

Majoritatea browserelor au implementat acum un mecanism numit cookie `SameSite`, care este o proprietate asociată cu cookie. Când trimit cereri, browserele vor verifica această proprietate și vor decide dacă se atașează cookie-ul într-o cerere între site-uri. O aplicație web poate seta un cookie ca `SameSite` dacă nu dorește cookie-ul care urmează să fie atașat la solicitările între site-uri. De exemplu, pot marca cookie-ul ID de sesiune ca `SameSite`, astfel încât nici o solicitare între site-uri nu poate folosi ID-ul sesiunii și, prin urmare, nu poate lansa atacuri CSRF.

Pentru a ajuta studenții să-și facă o idee despre modul în care cookie-urile `SameSite` pot ajuta la apărarea împotriva atacurilor CSRF, am creat un site web numit `www.example32.com` în unul dintre containere. Vă rugăm să vizitați următoarele URL-uri (`hostname` este deja mapat la `10.9.0.5` în fișierul `/etc/hosts`; dacă nu utilizați SEED VM, ar trebui să adăugați această mapare în mașina dvs.):

```
URL: http://www.example32.com/
```

Odată ce ați vizitat acest site o dată, trei cookie-uri vor fi setate în browserul dvs., `cookie-normal`, `cookie-lax` și `cookie-strict`. După cum indică numele, primul cookie este doar unul normal, cel de al doilea și al treilea cookie sunt cookie-uri `SameSite` de două tipuri diferite (tipurile `Lax` și `Strict`). Am conceput două seturi de experimente pentru a vedea ce cookie-uri vor fi atașate atunci când trimiteți o solicitare HTTP înapoi la server. De obicei, toate cookie-urile aparținând serverului vor fi atașate, dar nu este cazul dacă un cookie este de tip `SameSite`.

Vă rugăm să urmați legăturile pentru cele două experimente. Legătura A indică o pagină de pe `example32.com`, în timp ce legătura B indică o pagină de pe `attacker32.com`. Ambele pagini sunt identice (cu excepția culorii de fundal), și ambele trimit trei tipuri diferite de solicitări la

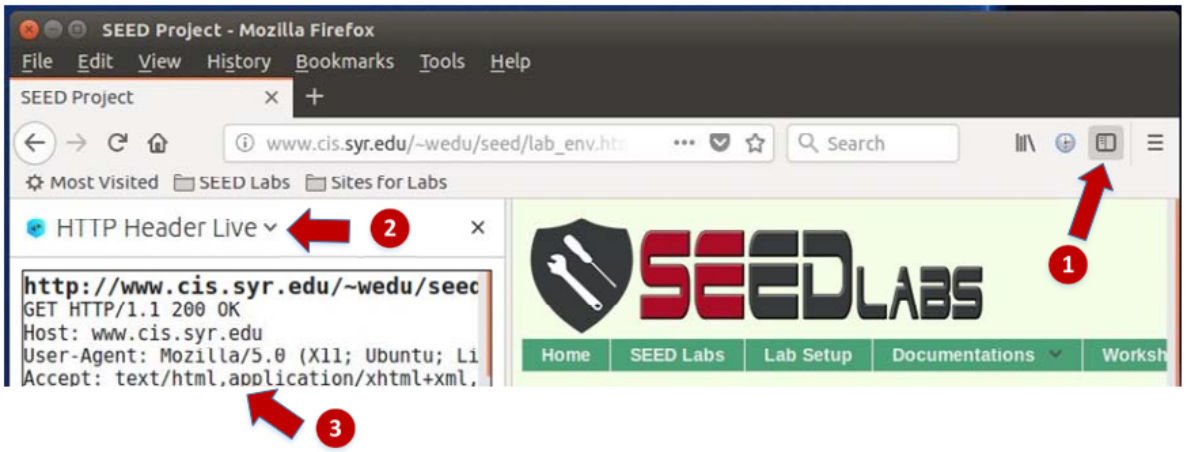


Figura 1: Activarea add-on HTTP Header Live

`www.example32.com/showcookies.php`, care afișează pur și simplu cookie-urile trimise de browser. Privind rezultatele afișajului, puteți spune care cookie-urile au fost trimise de browser. Vă rugăm să faceți următoarele:

- Descrieți ceea ce vedeți și explicați de ce unele cookie-uri nu sunt trimise în anumite scenarii.
- Pe baza înțelegerii dvs., descrieți modul în care cookie-urile `SameSite` pot ajuta un server să detecteze dacă o solicitare este o solicitare între situri sau pe același site.
- Vă rugăm să descrieți cum ați folosi mecanismul cookie `SameSite` pentru a ajuta Elgg să se apere împotriva atacurilor CSRF. Trebuie doar să descrieți idei generale și nu este nevoie să le implementați.

Puncte bonus. Deși nu este obligatoriu, studenții sunt încurajați să modifice aplicația Elgg, așa că ei pot folosi mecanismul cookie `SameSite` pentru a se apăra împotriva atacurilor CSRF. Recomandăm instructorilor să ofere puncte bonus pentru studenții care pot face acest lucru.

4 Ghid

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

Versiunea de Firefox (versiunea 83) din mașina noastră virtuală Ubuntu 20.04 nu suportă add-on `LiveHTTPHeader`. 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

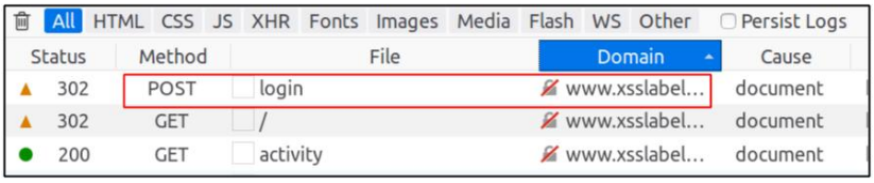


Figura 2: Cererea HTTP așa cum se vede în unealta Web Developer Network

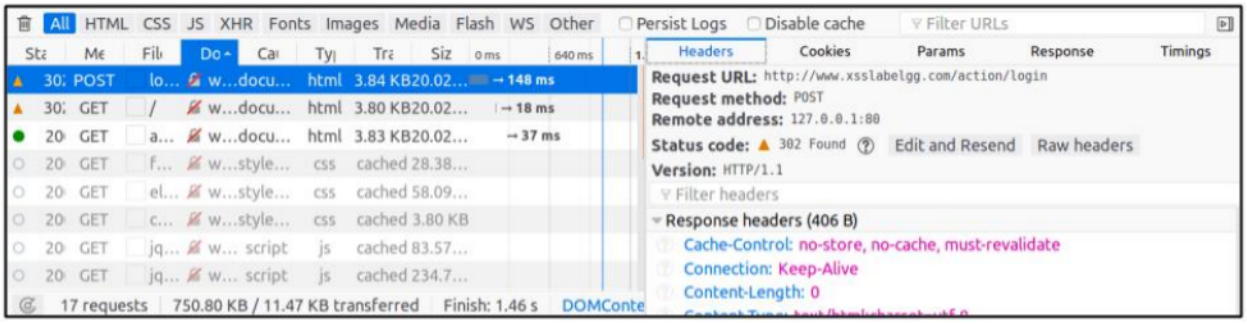


Figura 3: Cererea HTTP și detaliile cererii în două panouri.

caracteristicile importante ale acestei unelte. Web Developer Network Tool poate fi activată prin următoarea navigare:

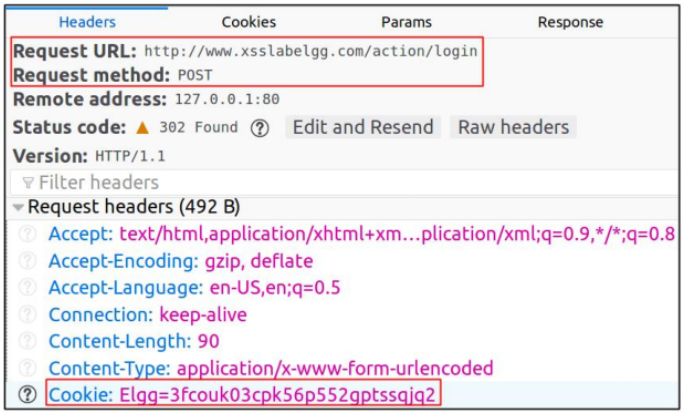
```
Clic pe meniul din dreapta-sus al Firefox's top --> Web Developer --> Network
or
Clic pe meniul "Tools" --> Web Developer --> Network
```

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

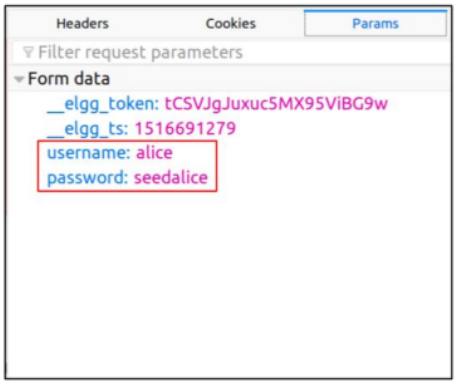
Pentru a vedea detaliile cererii, putem da click pe o cerere HTTP anume și unealta va arăta informațiile în două panouri (cf. figura 3).

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.



(a) Antetele cererii HTTP



(b) Parametrii cererii HTTP

Figura 4: Antetele HTTP și parametrii lor

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.

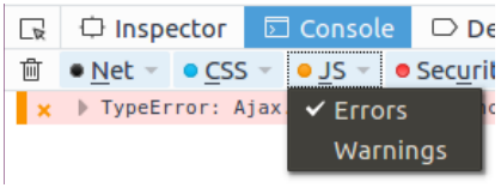


Figura 5: Depanarea codului JavaScript – activarea afișării erorilor

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

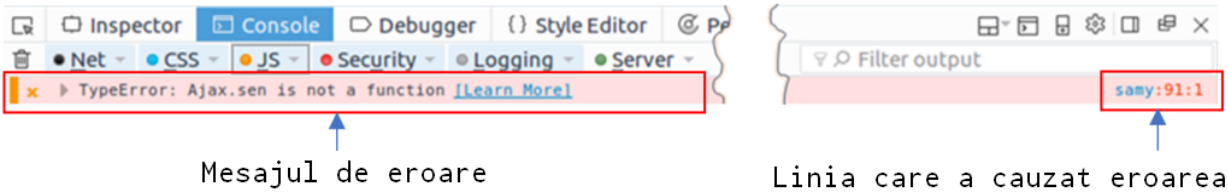


Figura 6: Depanarea codului Javascript – determinarea liniei cu eroarea

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. Rapoartele trebuie sa cuprindă bucățile de cod importante, cu explicații. Includerea codului fără acestea nu contează.