Scrapping et sessions 1

[**Scrapping**](#_j5wl9gegwb1o) **3**

[**Exemple**](#_y07ao2hydxyf) **4**

[Accès protégé](#_wxim91n0ftoa) 4

[Investigation chrome](#_dyky9kah23yo) 4

[Investigation curl](#_hg4zopehg6i4) 6

[Pseudo code node](#_x8jvlax7cuic) 8

[Code node documenté](#_sw6fqk3ecisb) 9

[Code node compact](#_7g6aedowwyjg) 12

# Scrapping

En général le module request est principalement utilisé pour récupérer / pusher du contenu via API cependant il est également très utiliser pour effectuer du scrapping (robots/programme se faisant passer pour un utilisateur pour récupérer du contenu sur un autre site).

Comme on a pu le voir, c’est le client qui décide ce qu’il envoi (heaader, données, etc…), ainsi il est très facile pour un programme informatique, un robots, etc… d’imiter le comportement d’un navigateur et ainsi se faire passer pour un utilisateur légitime.

Il serait par exemple très facile de réaliser un site qui vend des places de cinéma (sans en avoir réellement), simplement en allant récupérer le contenu des autres sites por la recherche, puis en imitant le comportement d’un utilisateur lors de l’achat d’une place.

C’est pourquoi, la plupart des sites web on des systèmes de protections (contrôle des headers, contrôles des sessions, robots blockers, etc... ) permettant d’éviter que des robots leurs envoi des requêtes. Cela empêche les programmes “ennemis” d’utiliser les services du sites web hors du contexte légitime (navigateur client).

Le moyen le plus simple pour tromper ces systèmes de protections est en général de passer les mêmes informations que le navigateur :

* Header Cookie : session client et information client
* Header User-Agent : plateforme de l’utilisateur (chrome, firefox, IOS, etc…)
* Header Accept : Encodage accepté,
* Détection de javascript
* Etc…

Pour qu’une opération de scrapping soit donc fonctionnelle et indétectable elle doit donc ressembler le plus possible à celle d’un navigateur.

En principe on effectue les actions suivantes :

* On effectue la requête avec le navigateur en mode incognito et on inspecte ce qui transite sur le réseau via la console network (F12 sous chrome) : On recherche les headers non standard, les identifiant de session dans les données GET/POST les cookies, etc...
* On recherche la présence de cookie de session obligatoire : on supprime les cookies un par un puis on continue dans le tunnel de vente ou on actualise pour vérifier si le cookie est requis ou non (si il y a une erreur c’est qu’il est requis).
* On rejoue ensuite la ou les requêtes sous curl, en supprimant les headers / données / cookies non obligatoires
* On code le module sous node

# Exemple

## Accès protégé

Dans l’exemple ci dessous, nous souhaitons accéder à la page <https://stage.optionizr.com/protected> avec notre programme. Cette page est protégée par une session (vous pouvez essayer avec votre navigateur), elle nécessite d’être passé précédemment par la page <https://stage.optionizr.com/session>. L’accès à cette page est désactivée pour les robots. Le but du programme est de récupérer le secret contenu sur la page <https://stage.optionizr.com/protected>

L’accès direct à la page <http://stage.badasse.com/protected> avec une mauvaise valeur en query ou sans cookies, ou avec un cookie invalide entraîne une erreur.

L’accès à une des 2 pages via un robot / programme type cURL entraîne également une erreur.



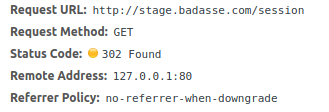
# 

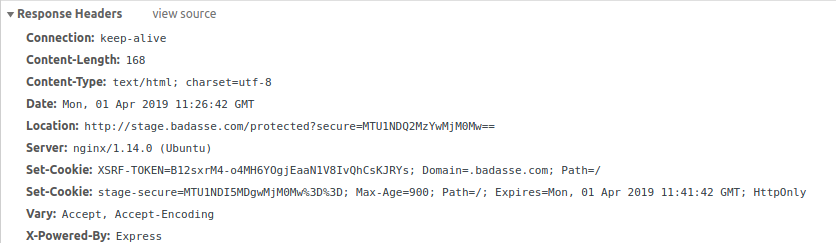
## Investigation chrome

Dans votre navigateur, l’accès à la page <https://stage.optionizr.com/session> entraine les actions suivantes :

* 1/ Le serveur vérifie que vous êtes un navigateur, si c’est le cas il vous renvoi un identifiant dans un cookie sécurisé (header set-cookie).
* Il vous retourne ensuite la page de redirection (<https://stage.optionizr.com/protected>) avec un paramètre get contenant un identifiant de session
* La navigateur vous redirige ensuite vers <https://stage.optionizr.com/protected> avec le paramètre query de session et le cookie sécurisé (header cookie) et le pr

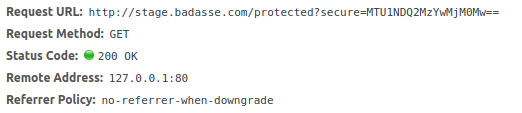
Chrome, requête <https://stage.optionizr.com/session> (informations importantes dans “Response headers”, headers location et Set-Cookie:

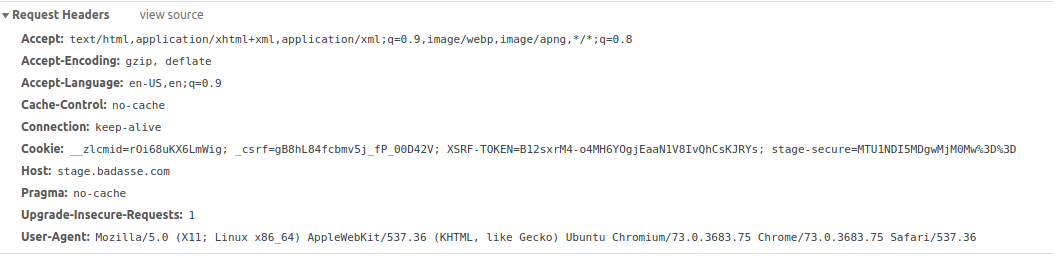






Chrome, requête <http://stage.badasse.com/protected?secure=MTU1NDQ2MzYwMjM0Mw==> (champs important dans “Request Headers”, header cookie)







Nous remarquons donc les éléments suivants :

* Avec ces headers la première requête fonctionne
* La réponse de la première requête retourne un 302 avec l’url contenant la query sécurisé dans le header Location
* La réponse de la première requête retourne le cookie sécurisé dans le header Set-Cookie

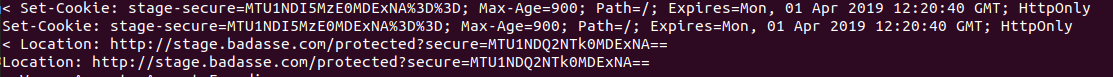
## Investigation curl

La première étape va être d’identifier avec curl les valeurs (header / cookies / donnée obligatoire).

Requête 1 curl avec tous les headers passé par le navigateur (avec option -v et -i pour le debug) :

*curl -X GET '****http://stage.badasse.com/session****' -H 'Connection: keep-alive' -H 'Pragma: no-cache' -H 'Cache-Control: no-cache' -H 'Upgrade-Insecure-Requests: 1' -H 'User-Agent: Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/73.0.3683.75 Chrome/73.0.3683.75 Safari/537.36' -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8' -H 'Accept-Encoding: gzip, deflate' -H 'Accept-Language: en-US,en;q=0.9' -H 'Cookie: \_\_zlcmid=rOi68uKX6LmWig; \_csrf=gB8hL84fcbmv5j\_fP\_00D42V; XSRF-TOKEN=A3NWezL6-MAsIYatpwm34LJg6-kvDfnM-0N0' -vi*

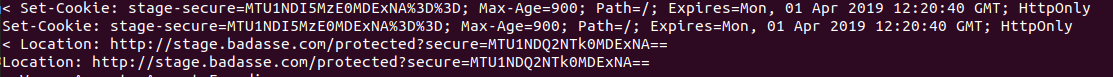




Nous récupérons bien le HTTP 302, l’url avec le paramètre sécurisé dans le header Location et notre cookie dans le header Set-Cookie. Nous allons maintenant supprimer tous les headers non obligatoires pour simplifier la requête (les headers Accept, Content-length, Language, etc… sont en général non obligatoire, on essaye de supprimer également le plus de valeurs possible dans le header cookie):

*curl 'http://stage.badasse.com/session' -H 'User-Agent: Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/73.0.3683.75 Chrome/73.0.3683.75 Safari/537.36' -vi*





Nous récupérons toujours les informations, nous remarquons donc que le seul header obligatoire pour la première requête est le header User-Agent.

Nous allons maintenant pouvoir effectuer la deuxième requête en passant le cookie sécurisé, nous conservons le header de la première (en général les règles ne changent pas d’une page à l’autre) :

*curl -X GET 'http://stage.badasse.com/protected?secure=MTU1NDQ2NTk0MDExNA==' -H 'User-Agent: Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/73.0.3683.75 Chrome/73.0.3683.75 Safari/537.36' -H "Cookie: stage-secure=MTU1NDI5MzE0MDExNA%3D%3D;" -vi*



Nous récupérons bien le secret !

## Pseudo code node

Nous allons donc sous node effectuer les actions suivante :

1. Requête GET sur /session en passant le header User-Agent, on précise également qu’on ne veut pas suivre les redirections (pour pouvoir récupérer le header set-cookie)
2. On vérifie que la requête renvoi bien un 302, on vérifie que le header location contient bien quelque chose
3. On vérifie que le(s) header(s) set-cookie contient bien notre cookie sécurisé
4. On récupère l’url de redirection dans le header Location
5. On récupère la valeur du cookie de session
6. On effectue la requête sur /protected avec le paramètre query secure, en précisant le User-Agent et en passant le cookie.
7. On affiche le résultat

Vous pouvez essayer vous même, sinon la solution est au paragraphe ci-dessous

## Code node documenté

var request = require("request");

request.get({

url: "http://stage.badasse.com/session",

followRedirect: false,

followAllRedirect: false,

headers: {

"User-Agent": "Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/73.0.3683.75 Chrome/73.0.3683.75 Safari/537.36"

}

}, function(err, response, body){

console.log(response.headers);

/\*\*

{ server: 'nginx/1.14.0 (Ubuntu)',

date: 'Mon, 01 Apr 2019 12:27:03 GMT',

'content-type': 'text/plain; charset=utf-8',

'content-length': '84',

connection: 'close',

'x-powered-by': 'Express',

'set-cookie':

[ '\_csrf=CxNxRDSNjzysvJ1buB83OQ4i; Path=/',

'XSRF-TOKEN=mTf5OGzO-s7tfgN5gfnAlaflOL\_m-Fn5T85Y; Domain=.badasse.com; Path=/',

'stage-secure=MTU1NDI5NDQyMzM0Mg%3D%3D; Max-Age=900; Path=/; Expires=Mon, 01 Apr 2019 12:42:03 GMT; HttpOnly' ],

location: 'http://stage.badasse.com/protected?secure=MTU1NDQ2NzIyMzM0Mg==',

vary: 'Accept, Accept-Encoding' }

\*\*/

if(err || !response || response.statusCode != 302)

{

console.log(err);

console.log("Erreur 302");

}

else if(!response.headers["location"])

{

console.log("Erreur location");

}

else if(!response.headers["set-cookie"])

{

console.log("Erreur cookie");

}

else

{

// Récupération valeur cookie - mode complet

// 1/ filter : on récupère l'element du tableau contenant le cookie : ['stage-secure=MTU1NDI5NDQyMzM0Mg%3D%3D; Max-Age=900; Path=/; Expires=Mon, 01 Apr 2019 12:42:03 GMT; HttpOnly']

// 2/ join : on tranforme le tableau en string 'stage-secure=MTU1NDI5NDQyMzM0Mg%3D%3D; Max-Age=900; Path=/; Expires=Mon, 01 Apr 2019 12:42:03 GMT; HttpOnly'

// 3/ split : On split avec les ; : [stage-secure=MTU1NDI5NDQyMzM0Mg%3D%3D , Max-Age=900 , Path=/ , ....]

// 4/ filter: On récupère uniquement la permière partie ["stage-secure=MTU1NDI5NDQyMzM0Mg%3D%3D"]

// 5/ join : On transforme le tableau en string "stage-secure=MTU1NDI5NDQyMzM0Mg%3D%3D"

var securecookie = response.headers["set-cookie"]

.filter(function(e){ // 1

if(e.indexOf("stage-secure") == -1)

{

return false;

}

else

{

return true;

}

})

.join("") // 2

.split(";") // 3

.filter(function(e){ // 4

if(e.indexOf("stage-secure") == -1)

{

return false;

}

else

{

return true;

}

})

.join("");

console.log(securecookie);

// Récupération valeur cookie - mode regex optimisé

var re = new RegExp("(stage-secure=.\*?);", "gmi");

var result = new RegExp("(stage-secure=.\*?);", "gmi").exec(response.headers["set-cookie"].join(""));

securecookie = (result && result.length >= 2) ? result[1] : "";

console.log(securecookie);

request.get({

url: response.headers["location"],

json: true,

headers: {

"Cookie": securecookie,

"User-Agent": "Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/73.0.3683.75 Chrome/73.0.3683.75 Safari/537.36"

}

}, function(err, response, body){

if(err || !response || response.statusCode != 200)

{

console.log(err);

console.log("Erreur requête 2");

}

else

{

console.log("Résultat");

console.log(body)

}

});

}

});

## Code node compact

var request = require("request");

request.get({

url: "http://stage.badasse.com/session",

followRedirect: false,

followAllRedirect: false,

headers: {

"User-Agent": "Mozilla/5.0 (X11; Linux x86\_64))"

}

}, function(err, response, body){

if(err || !response || response.statusCode != 302 || !response.headers["location"] || !response.headers["set-cookie"])

{

console.log(err);

console.log("Erreur");

}

else

{

var re = new RegExp("(stage-secure=.\*?);", "gmi");

var result = new RegExp("(stage-secure=.\*?);", "gmi").exec(response.headers["set-cookie"].join(""));

securecookie = (result && result.length >= 2) ? result[1] : "";

request.get({

url: response.headers["location"],

json: true,

headers: {

"Cookie": securecookie,

"User-Agent": "Mozilla/5.0 (X11; Linux x86\_64)"

}

}, function(err, response, body){

if(err || !response || response.statusCode != 200)

{

console.log(err);

console.log("Erreur requête 2");

}

else

{

console.log("Résultat");

console.log(body)

}

});

}

});