

A la découverte de Chickenfoot

Atlab - <http://www.atlab.fr> - contact@atlab.fr

Table des matières

1	Introduction	2
1.1	Chickenfoot ?	2
1.2	Comparaison	2
1.3	Notes	3
2	Se lancer en 15 minutes	4
2.1	Installation	4
2.2	Premier test	4
2.3	Record actions	5
2.4	Mise en forme	6
2.5	Conclusion	8
3	Approfondir sa découverte	9
3.1	Exemples de base	9
3.2	Match dynamique	9
3.3	Gestion de tabs	11
3.4	Parsing JavaScript	12
3.5	Fichiers et classes	13
3.6	GetElementsByClassName	14
3.7	ClickAllOnClick	15
4	Chrome ou l'internal Firefox	16
4.1	Lancement de processus	16
4.2	Changement de préférences	17
4.3	Screenshot	17
4.4	LoginManager	18
5	Remarques et autres idées diverses	19

1 Introduction

1.1 Chickenfoot ?

Chickenfoot[1] (développé par le User Interface Design Group[2] du MIT) est une extension Firefox[3]. Il fournit une interface qui permet de contrôler son navigateur au moyen de scripts.

Ses principaux atouts sont :

- Les scripts créés tournent dans l'espace chrome (au delà du DOM de la page, on est au niveau des extensions, on peut donc interagir avec le code de Firefox).
- De nombreuses fonctions et bibliothèques extrêmement utiles sont fournies.

C'est le juste milieu entre :

- La macro de souris/clavier extrêmement imprécise (en particulier lorsqu'il s'agit de gérer les erreurs).
- Le script curl ou équivalent, détectable et limité (comportement non classique, non exécution du javascript, non téléchargement des images, manque de cookie flash, etc.).

Chickenfoot permet donc d'automatiser un comportement utilisateur sans avoir à émuler un navigateur.

Nous nous sommes uniquement intéressés à l'aspect automatisation d'actions, mais il est à noter que ses fonctionnalités sont aussi orientées sur la présentation de contenu (la plupart des tutoriaux le présente d'ailleurs sous cet aspect).

1.2 Comparaison

Dans ses rivaux, on notera Greasemonkey[6], qui est plus pensé pour faciliter la navigation, son principal avantage étant qu'il est bien plus utilisé et beaucoup plus stable.

Par contre, Chickenfoot a de son côté :

- 99% des scripts Greasemonkey fonctionnent dans Chickenfoot tandis que seul 1% des scripts Chickenfoot fonctionnent dans Greasemonkey (le Javascript de Chickenfoot a beaucoup plus de droits, il tourne au niveau du moteur, dans le contexte des extensions). Le Javascript de Greasemonkey a les mêmes droits que celui qu'on peut voir sur les sites web, limité au DOM de la page.
- Les bibliothèques et fonctions de base, sans égales (go, find, enter, click, opentab, etc.)

Pour résumer, on automatise des actions humaines avec Chickenfoot, pas la page.

Dans ses concurrents proches qui agissent sur la même couche, on notera aussi SeleniumHQ[7], qui depuis quelques temps est très actif. N'ayant pas poussé l'étude de ce dernier (en particulier les récentes fonctionnalités, l'IDE), nous laisserons au lecteur le soin de se faire sa propre idée.

(Si une personne souhaite échanger sur le sujet, qu'elle n'hésite pas à nous contacter par mail).

1.3 Notes

La majorité des exemples de cet article ont été codés sur un Chickenfoot 1.0.5 (released July 1, 2009), Firefox 3.0.12 sous Ubuntu classique.

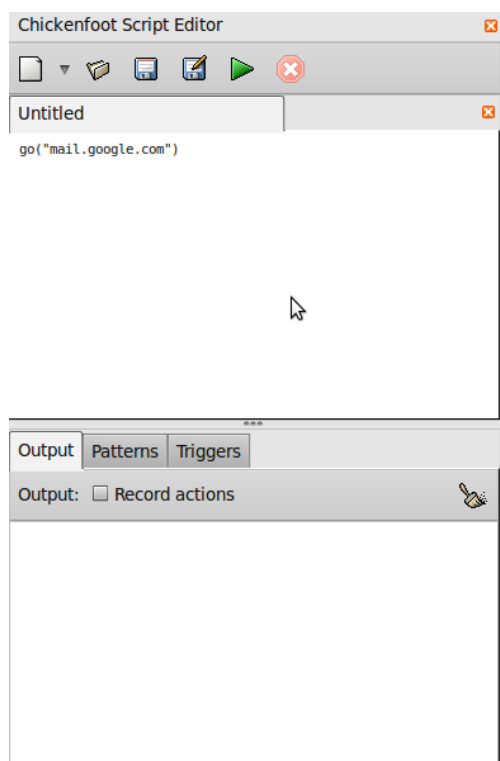
Ils ont été testés pour la plupart sous Windows et Linux, et sur différentes versions de Chickenfoot/Firefox. Normalement aucun problème de compatibilité majeur n'est à noter pour ces scripts. L'adaptation des codes pour Windows implique tout de même une modification des chemins de fichiers (et vice-versa).

2 Se lancer en 15 minutes

2.1 Installation

Vous commencerez par vous munir d'un firefox récent[3], vous irez installer l'extension Chickenfoot[5], puis après avoir redémarré, il sera nécessaire d'utiliser le raccourci "F8"¹

2.2 Premier test



On remarque donc deux zones, celle du haut, l'éditeur de texte, celle du bas, qui fait un peu de tout (debug, log, tools). On va placer notre première ligne de Javascript dans l'éditeur pour vérifier que tout fonctionne correctement :

```
go("mail.google.com");
```

On lance le script avec Alt+R² :

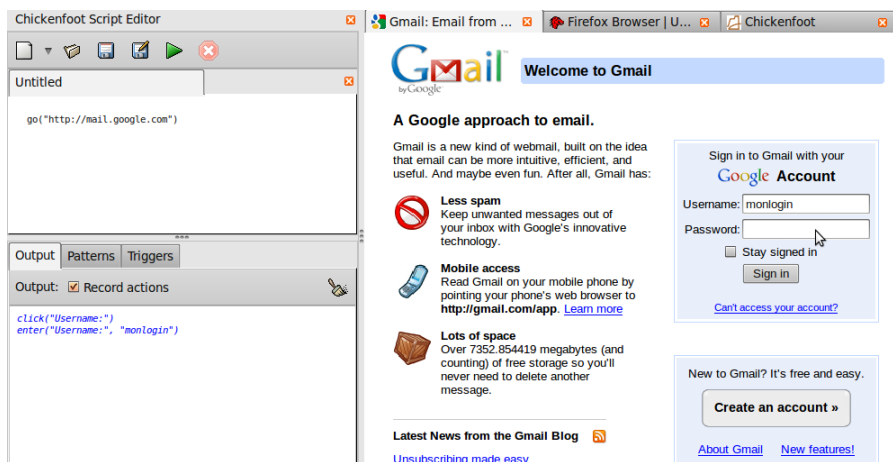
⇒ La tab courante charge <http://mail.google.com>

1. ou View⇒Sidebar⇒Chickenfoot.

2. ou le bouton "play"

2.3 Record actions

Activons maintenant la case “Record actions”, et utilisons un couple d’identifiants valides sur l’écran de login qui vient de se charger.



La fenêtre de log devrait se remplir des éléments suivants :

```
click("Username:")
enter("Username:", "monlogin")
enter("Password:", "monpassword")
click("Sign in button")
go("https://www.google.com/accounts/ServiceLogin?service")
go("https://www.google.com/accounts/ServiceLogin?service...")
go("https://www.google.com/accounts/ServiceLoginAuth?service=mail")
go("https://www.google.com/accounts/ServiceLoginAuth?service=mail")
go("https://mail.google.com/mail/?zx=xxxxxxxxxxxx&shva=1")
go("https://mail.google.com/mail/?zx=xxxxxxxxxxxx&shva=1")
go("https://mail.google.com/mail/?zx=xxxxxxxxxxxx&shva=1")
go("https://mail.google.com/mail/?zx=xxxxxxxxxxxx&shva=1")
go("https://mail.google.com/mail/?zx=xxxxxxxxxxxx&shva=1#inbox")
```

Copiez/collez ce qui est pertinent en dessous du go(), à savoir :

```
enter("Username:", "monlogin")
enter("Password:", "monpassword")
click("Sign_in_button")
```

(Le reste étant lié aux redirections HTTP effectuées par Google pour Gaia).

Videz le buffer de log³, et cliquez maintenant sur “Settings” dans l’interface Gmail. Le log devrait ressembler à quelque chose du style :

```
click(new XPath(““/HTML[1]/BODY[1]/DIV[1]/DIV[2]/DIV[1] \
    /DIV[2]/DIV[1]/DIV[2]/DIV[1]/DIV[1]/DIV[1]/
    [...] /SPAN[1]/#text[1]”))
```

Vous l’aurez compris, la fonction “Record actions” a ses limites. Ici, elle a simplement trouvé la balise `span` correspondant à votre clic dans la source HTML. On préférera donc utiliser un `click(“Settings”)`⁴.

Ce qui se passe avec cet appel est simple, Chickenfoot parcourt toute la page à la recherche de la chaîne “Settings”, cherche un lien hypertexte à côté⁵, et émule un clic sur ce dernier.

2.4 Mise en forme

On répète le processus, on teste chaque commande individuellement (en commentant le précédent code avec les commentaires Javascript `/* */`) jusqu’à arriver à quelque chose de ce type :

```
go(“http://mail.google.com”)
enter(“Username:”, “monlogin”)
enter(“Password:”, “monpassword”)
click(“Sign_in_button”)
click(“Don’t_always_use_https”)
click(“Save_Changes”)
```

3. Le bouton juste à côté de “Record actions”

4. Ou l’équivalent français suivant la langue de votre interface

5. Dans le cas où il y aurait plusieurs possibilités, il est possible de lui préciser la combienième l’on souhaite, au moyen de `click(“second Settings”)` ou encore `click(“third Settings”)`

Afin de rendre la chose fonctionnelle, chaque action va être temporisée (la précédente doit être terminée avant d'entamer la nouvelle) :

```
go("http://mail.google.com")
sleep(3)
enter("Username:", "monlogin")
enter("Password:", "monpassword")
click("Sign_in_button")
sleep(10)
click("Settings")
sleep(3)
click("Don't_always_use_https")
sleep(2)
click("Save_Changes")
output("[+]_Done")
```

Et pour la version française :

```
go("http://mail.google.com")
sleep(3)
enter("Nom_d'utilisateur:", "monlogin")
enter("Mot_de_passe:", "monpassword")
click("Connexion_button")
sleep(10)
click("Param\`etres")
sleep(3)
click("syst\`ematiquement")
sleep(2)
click("enregistrer")
output("[+]_Done")
```

Maintenant, vérifiez que votre compte est bien en "Always use https" (ce qui devrait être le cas pour des raisons de sécurité), déconnectez vous et lancez le script.

N'oubliez pas de bien remettre l'option dans le cas où ce ne serait pas un compte de test.

2.5 Conclusion

En quelques lignes, nous sommes capables de nous connecter et d'utiliser un des webmails les plus complexes qu'il soit à scripter.

De plus, google n'y a vu qu'un utilisateur classique, étant donné qu'on scripte sur Firefox. L'exemple présenté ici est extrêmement basic, ce qui laisse la place au chapitre suivant.

3 Approfondir sa découverte

3.1 Exemples de base

L'approche sera à partir de maintenant orientée autour de codes source commenté.

```
go("http://twitterfall.com")
wait(window)
output("[+]_Done.")
```

Le `wait()` permet d'attendre que la page soit chargée. Il prend en paramètre une fenêtre afin de s'affranchir des `sleep()`.

```
go("google.com")
wait(window)

enter("Google_search", "linkedin")
click("I'm_feeling")
results = find("Account_&_Settings")

if (results.count >= 1) {
    output("[+]_Connected_on_linkedin")
} else {
    output("[+]_Not_connected_on_linkedin")
}
output("[+]_Done")
```

`find()` agit comme `wait()`, il attend le chargement complet de la page avant de chercher les occurrences de la chaîne passée en paramètre.

3.2 Match dynamique

```
var some_trend = "Jackson";

go("twitterfall.com");
wait(window)
var last = 0;
for(var i = 0; i < 30; i++) {
    results = find(some_trend);
    if (results.count != last) {
        last = results.count;
        output("[+]_" + last + "_matches");
    }
}
```

```
    }  
    sleep(1);  
  }  
  output(" [+] Done. ");
```

Cet exemple permet de jouer avec une application en ajax, twitterfall. Le concept est de parcourir la page afin de détecter de nouveaux éléments.

The screenshot displays a web browser window with a Twitter page and a script editor. The script editor, titled "Chickenfoot Script Editor", shows the following code:

```
1 var some_trend = "Nexus";  
2  
3 go("twitterfall.com");  
4 wait(window)  
5 var last = 0;  
6 for(var i = 0; i < 30; i++) {  
7   results = find(some_trend);  
8   if (results.count != last) {  
9     last = results.count;  
10    output(" [+] " + last + " matches");  
11  }  
12  sleep(1);  
13 }  
14 output(" [+] Done. ");  
15
```

The browser window shows the Twitter interface with various tweets and navigation elements. The output of the script is visible in the bottom left corner of the browser window:

```
[+] 1 matches  
[+] Done.  
  
[+] 12 matches  
[+] 13 matches  
[+] 14 matches  
[+] 15 matches  
[+] Done.
```

3.3 Gestion de tabs

```

tab1 = openTab("about:blank");

tab1.go("http://pastebin.ca/");
wait(tab1.window);
tab1.pick("Content_Type:_listbox", "javascript");
tab1.enter("fourth_textbox", "http://
    lasecuriteoffensive.fr\nBlog_Atlab_-_documentation
    _Chickenfoot");
tab1.enter("Name_/_Title", "Anonyme");
tab1.click("first_Submit_Post");
wait(tab1.window);
while (tab1.document.location == "http://pastebin.ca/
    index.php") { /* meta equiv refresh */
    sleep(2);
}
loc = "" + tab1.document.location;
tab1.close();
output("[+]_Posted_on_pastebin,_" + loc);
output("[+]_Done.");

```

Cette fois-ci on testera les tabs. Pensez à utiliser with() de cette façon :

```

tab1 = openTab("about:blank");
with(tab1) {
    go("http://pastebin.ca/");
    wait(window);
    pick("Content_Type:_listbox", "javascript");
    enter("fourth_textbox", "http://lasecuriteoffensive.
        fr\nBlog_Atlab_-_documentation_Chickenfoot");
    enter("Name_/_Title", "Anonyme");
    click("first_Submit_Post");
    wait(window);
    while (document.location == "http://pastebin.ca/
        index.php") {
        sleep(2);
    }
    loc = "" + document.location;
    close();
}

```

3.4 Parsing JavaScript

```
go("isc.sans.org");
wait(window);
var threatlevel = "";
threatlevel = document.getElementById("threatlevel").
    innerHTML;
threatlevel = threatlevel.split("\\")[9];
threatlevel = threatlevel.split("/")[2];
threatlevel = threatlevel.split("_")[1];
threatlevel = threatlevel.split(".")[0];
output("[+]_Today_is_" + threatlevel + "_ISC_says");

reg = /threatlevel_([a-z]+)\.jpg/;
threatlevel = reg.exec(document.getElementById("
    threatlevel").innerHTML)[1];
output("[+]_same_with_regexp_:_" + threatlevel);
```

L'exemple ci-dessus n'a pas grand intérêt, mais il est là pour vous rappeler que nous avons accès au DOM de la page, et que nous pouvons accéder aux objets window, document... tout comme dans du Javascript classique. Pour plus d'infos sur les expressions régulières en Javascript, suivez le lien[10].

3.5 Fichiers et classes

```

include("fileio.js");

function Logger(logfile) {

    this.logfile = logfile;
    this.IsViewer = 0;
    if (!exists(this.logfile))
        append(this.logfile, "["+new Date()+"] "+this.
            logfile+"_created\n");

    this.Log = function (msg) {
        thetime = new Date();
        append(this.logfile, "["+thetime+"] "+msg+"\n");
        if (this.IsViewer)
            this.tempTab.reload();
    }; /* !Log */

    this.LaunchViewer = function () {
        this.tempTab = openTab("about:blank");
        this.tempTab.go("file:///"+this.logfile);
        wait(this.tempTab.window);
        this.IsViewer = 1;
    }; /* !LaunchViewer */

    this.StopViewer = function () {
        if (!this.IsViewer)
            return ;
        this.IsViewer = 0;
        this.tempTab.close();
    }; /* !StopViewer */
}

log = new Logger("/tmp/chickenfoot.txt");
log.LaunchViewer();
try {
    log.Log("Trying_google.com...");
    go("http://google.com");
    wait(window);
    log.Log("I_reached_google.com!");
} catch (e) { log.Log(e) }
// log.StopViewer();

```

Pour finir, voici un exemple de classe (une classe se déclare comme une fonction), qui permet de surcharger le debug, en le redirigeant dans un fichier (l'include fileio.js permet d'accéder aux méthodes exists(), append(), read(), write()).

Pour plus d'informations sur toutes les fonctions fournies, une section référence plutôt bien fournie sur le wiki Chickenfoot est disponible[11].

3.6 GetElementsByClassName

```
function GetElementsByClassName(handle, tag_, class_)
{
    var i, k;
    var T_Result = new Array();
    var O_Tab = handle.document.getElementsByTagName(
        tag_);

    for(i = 0, k = 0; i < O_Tab.length; i++) {
        if(O_Tab[i].className == class_) {
            T_Result[k++] = O_Tab[i];
        }
    }
    return(T_Result);
} /* GetElementsByClassName */
```

Dans certains cas de parsing à la main, vous tomberez sur un problème que cette fonction résoudra (elle commence aussi à apparaître dans certains moteurs Javascript dits "HTML5", et est bien évidemment plus optimisée).

4 Chrome ou l'internal Firefox

Les exemples qui suivent seront orientés autour du moteur de Firefox. L'idée ici est de fournir des fonctions classiques (exécution d'un binaire, modification de configuration).

C'est au lecteur d'adapter en fonction de ses besoins, en poursuivant dans la documentation officielle Mozilla utilisée par les développeurs d'extensions[12].

4.1 Lancement de processus

```
PyExec = function (pythonfile) {
    var file = Components.classes["@mozilla.org/file/
        local;1"]
                            .createInstance(Components.
                                interfaces.nsILocalFile);
    file.initWithPath("/usr/bin/env");
    var process = Components.classes["@mozilla.org/
        process/util;1"]
                            .createInstance(Components.
                                interfaces.nsIPProcess);

    process.init(file);
    var args = ["python", pythonfile];
    process.run(true, args, args.length);
}

PyExec("/tmp/test.py");
```

Un exemple d'exécution de binaire, avec un script python. Il fonctionne correctement évidemment avec n'importe quel script/binaire. Une des choses intéressantes que cela peut permettre est par exemple de résoudre de manière simple la non possibilité de scripter les contenus de plugins comme flash ou Java (grâce à des logiciels comme xdotools[13])

4.2 Changement de préférences

```
ChgSocks = function (host , port) {
    Components.classes ['@mozilla.org/preferences-service
        ;1'].getService(Components.interfaces.
        nsIPrefBranch);

    a.setIntPref('network.proxy.type', 1);
    a.setCharPref('network.proxy.socks', '127.0.0.1');
    a.setIntPref('network.proxy.socks_port', 6677);
    a.setIntPref('network.proxy.socks_version', 5);
    a.setBoolPref('network.proxy.socks_remote_dns', true
    );
}
```

Ici un exemple repris d'une extension Firefox. Sachez qu'un .xpi n'est autre qu'une archive zip contenant des sources et ressources. Il y a assez d'extensions sur addons.mozilla.com pour fournir les exemples permettant de combler la grande majorité des besoins.

4.3 Screenshot

```
include("screenshot.js");

var filename = "C:\\\\Temp\\\\MyChickenfootScreenshot.png"
;
grabScreenshot(filename);
openTab(filename, true);
```

Cet exemple montre comment faire un screenshot en deux lignes, utile à des fins de logs, une image étant parfois bien plus parlante que des codes d'erreurs.

4.4 LoginManager

```
var p = new LoginManager();
p.addEntry('http://www.facebook.com/', 'prannay@mit.edu', 'abc');
p.addEntry('http://www.facebook.com/', 'jaja_binx@mit.edu', 'blurb', null, 'email', 'pass');
output(p.retrieveEntry('www.facebook.com', 'prannay@mit.edu'));
output(p.retrieveEntry('www.facebook.com'));
```

Cet exemple est tiré des sources de Chickenfoot[4], le LoginManager est une surcouche sur le XPCOM “mozilla.org/login-manager” de Firefox, et le rend beaucoup plus simple à utiliser. Cela permet d’éviter le stockage de couples utilisateur / mot de passe en clair dans les fichiers de sources.

5 Remarques et autres idées diverses

Quelques idées et infos en vrac :

Chickenfoot possède son propre système de mise à jour.

greasemonkey.js est une bibliothèque fournie avec Chickenfoot, elle permet de faire tourner la plupart des codes Greasemonkey.

Afin de créer un équivalent de la fonction `wait()` pour les appels XMLHttpRequest (événementiel sur les requêtes ajax), il est possible de remplacer la fonction XMLHttpRequest du DOM par la sienne.

Pyjamas[14], framework web python, compile du code python en Javascript (entre autres joyeusetés), tout comme Google Web Toolkit (GWT) le fait avec du Java.

Les chemins d'inclusion utilisés par la fonction `include()` contiennent le chemin du script exécuté (posez vos includes dans le dossier courant).

Tout objet créé en dessous de `global`[15] est partagé entre tous les scripts Chickenfoot.

Le script de Timo Lindfors, `chickenfoot-launcher`[16], permet de lancer ses scripts depuis le shell (grâce à du Xvnc une instance VNC, cela reste un script PoC).

Références

- [1] Chickenfoot
<http://groups.csail.mit.edu/uid/chickenfoot/>
- [2] User Interface Design Group
<http://groups.csail.mit.edu/uid/>
- [3] Firefox
<http://www.mozilla.com/en-US/firefox/>
- [4] Sources Chickenfoot
<http://groups.csail.mit.edu/uid/chickenfoot/developer-info.html>
- [5] Chickenfoot .xpi
<http://groups.csail.mit.edu/uid/chickenfoot/install.html>
- [6] Greasespot, Greasemonkey's blog
<http://www.greasespot.net/>
- [7] SeleniumHQ -
<http://www.seleniumhq.org>
- [8] PortSwigger - Burp Proxy
<http://portswigger.net/proxy/>
- [9] OWASP - WebScarab
http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- [10] Using Regular Expressions with JavaScript and ActionScript
<http://www.regular-expressions.info/javascript.html>
- [11] Chickenfoot wiki reference
<http://groups.csail.mit.edu/uid/chickenfoot/scripts/index.php?title=Reference>
- [12] Mozilla Developer Center - Code snippets
https://developer.mozilla.org/en/Code_snippets
- [13] xdotool - fake keyboard/mouse input
<http://www.semicomplete.com/projects/xdotool/>
- [14] Pyjamas - Python Javascript Compiler, Desktop Widget Set and RIA Web Framework
<http://pyjs.org/>
- [15] global - Chickenfoot API
<http://groups.csail.mit.edu/uid/chickenfoot/api.html#global>
- [16] chickenfoot-launcher - run chickenfoot scripts from command line - Timo Lindfors
<http://lindi.iki.fi/lindi/darcs/chickenfoot/README>
- [17] Atlab, filiale audit d'Atheos
<http://www.atlab.fr/>
- [18] La sécurité offensive - Blog Atlab
<http://secuoff.fr/>