Webh4ck.

Ce document aborde un sujet sensible. Il explique certains mécanismes des sites web et fournit des outils pour pouvoir passer outre certains "verrous", ou simplement mieux les comprendre. Le procédé n'exclut pas les utilisations abusives, bien que respectant scrupuleusement les règles du jeu sur le web. Chacun doit balayer devant sa porte. Ce document est public. Je décline toute responsabilité sur les effets que pourraient avoir l'application du procédé, la mise en oeuvre de tout code source tiré de ce document ou l'utilisation de tout logiciel mentionné.

La technique décrite démystifie les mécanismes de fonctionnement des sites web, et en particulier de certains mécanismes récurrents que le visiteur régulier aimerait pouvoir éviter. Car ils sont synonymes de perte de temps. Par exemple, les formulaires de login permettent de fournir un certain niveau de sécurité dans l'accès à une partie de site qui vous est réservée, mais si vous devez vous logger 50 fois par jour, en tant qu'utilisateur vous avez le droit de considérer que la qualité de service n'est pas au rendez-vous. Et ce à partir du moment où il n'y a pas moyen de court-circuiter cette étape. Un navigateur web évolué comme Internet Explorer 5+ permet de sauvegarder localement des données entrées dans n'importe quel formulaire. C'est déjà un très bel effort, mais ce n'est pas tout. Alors que de nombreuses applications sont désormais développées dans la logique du 100% web, nous devons faire de plus en plus face à des formulaires en plusieurs étapes, et à d'autres mécanismes laborieux. Et il est donc souhaitable de les comprendre et de voir quel degré d'automatisation pourrait être intégré.

Il ne s'agit pas que de formulaires. Il s'agit aussi de pages à l'intérieur d'un site, pages auxquelles vous ne pouvez pas accéder directement parce que le concepteur du site a décidé qu'il en serait ainsi. Le cas échéant, vous recevrez un message d'erreur suffisamment technique pour vous passer l'envie d'essayer de court-circuiter la traversée du site vers le point cherché. Décrire les mécanismes internes liant les différentes pages d'un site permet du coup de les rendre moins opaque et d'apporter pourquoi pas un degré de confiance.

Naturellement, un des points motivants de cet exposé est qu'il repose sur le protocole HTTP et le langage HTML, bref les standards du web. C'est-à-dire que le serveur soit IIS, Apache, Zope, qu'il utilise ou non du script serveur, qu'il soit ou non architecturé autour d'un serveur d'application, la logique fondamentale est toujours la même. Cela permet d'espérer pour de nombreuses applications de beaux jours devant elles. Par ailleurs, avec le web, nous avons un vrai standard à l'échelle de la planète, ce qui court-circuite les classiques problématiques de compatibilité.

Les utilisations de ce procédé sont nombreuses. Automatiser le remplissage d'un formulaire complexe permet par exemple de déposer un CV en ligne, d'un clic ! En comparaison, déposer un CV manuellement sur un site comme Monster prend 30 minutes. Par extension, le CV pourra être déposé automatiquement non plus sur un site, mais sur autant de sites d'emploi que l'on souhaite. Ce service existe déjà, notamment par le site Robopost, mais celui-là ne garantie pas la fiabilité de ce que l'on fait - en fait il cache tout - et est par ailleurs englué dans le marketing one-to-one et dans les bannières de publicité. Quitte à ce qu'un service "innovant" existe, il doit être fiable et adapté à l'attente des visiteurs. Robopost ne l'est pas.

Le procédé est complémentaire de procédés de reconnaissance automatique de formulaire sur un site. A ce titre, on peut citer FormsSpy (cartographie), FormsAnalyzer (détecte automatiquement le type de formulaire (login, newsletter, recherche, shop, ...), FormsPresenter (un outil de (re)validation de formulaire), et FormsServer (un catalogue de formulaires préremplis et prêts à l'emploi).

Chacun pourra utiliser le procédé comme il l'entend. Il est relativement clair que le procédé n'exclut pas les utilisations abusives, bien que respectant scrupuleusement les règles du jeu sur le web.

1. La sacro-sainte URL

L'accès à une page web quelconque se fait par la connaissance d'une combinaison d'URL, éventuellement complexe, d'un ensemble d'entêtes, de l'existence éventuelle d'un cookie et de l'existence éventuelle d'une variable de session serveur.

Cette URL est la composante principale de la page. Elle l'identifie de manière univoque.

Elle est complexe lorsqu'à la fois elle identifie la page et passe une ligne de commande (CGI). Lorsqu'une URL est complexe, l'URI (partie principale) est suivie d'un point d'interrogation et d'un certain nombre de paires *variable* = *valeur*, séparées par le signe &. Exemple : http://www.microsoft.com?id=53&subscriber=no

Ces paires sont appelées paramètres de type GET (par référence au protocole HTTP inventé par le consortium W3).

L'ensemble d'entêtes est un ensemble éventuellement vide de paires *variable* = *valeur*, séparées par un retour à la ligne (0D0A). Ces entêtes sont invisibles à travers du navigateur. Elles transitent sur la ligne entre la machine du visiteur et la machine hébergeant le serveur web, grâce au protocole HTTP. Le navigateur les reçoit et les traite, mais ne les affiche jamais à l'écran.

Ces paires sont appelées paramètres de type POST.

Le cookie est un fichier installé sur le disque dur de l'utilisateur. Lorsqu'on accède à une page Web, ce cookie est par défaut renvoyé par le navigateur au serveur web, toujours de manière cachée. Ce cookie permet de construire l'historique de navigation d'un visiteur sur le site, y compris lorsque l'ordinateur a été éteint entre temps.

Les cookies sont des cas particuliers de paramètres de type POST.

Les variables de session sont des objets créés par un serveur Web lorsqu'un visiteur arrive sur un site. Une variable de session permet de suivre l'utilisateur le long de sa visite, au même titre qu'un cookie. Mais à la différence du cookie, rien n'est stocké sur le disque dur de l'utilisateur. Parce que rien n'est stocké, il suffit que l'utilisateur quitte son navigateur (en fait il suffit qu'il aille sur un autre site web) pour que le serveur considère que la session est expirée, et que les variables de session associées soient balayées de la mémoire.

Le visiteur n'a aucun accès sur ces variables.

Pour des raisons d'unicité, pour chacun de ces types de paramètre, il est souvent nécessaire de recourir à un générateur de clef unique. Le serveur web génère une clef et l'attribue au visiteur courant. Lorsqu'une de ces clefs passe dans un paramètre de type GET, on peut la voir, et la noter.

Accéder à une page web quelconque autre que la homepage à partir de rien revient donc à reconstituer l'ensemble des paramètres dont la page web a besoin (GET et POST), le(s) cookie(s) ainsi que la(les) variable(s) de session.

La plupart du temps, le principe même des cookies et des variables de session permet de s'affranchir d'avoir à les simuler. En effet, un cookie et une variable de session sont d'une part deux facettes d'une même

technique de suivi de la visite. Or arriver sur une page web quelconque d'un serveur web ne porte pas préjudice à la notion de "première page visitée sur le site", "page courante", "dernière page visitée sur le site".

En d'autres termes, un cookie sera créé si on arrive sur une page autre que la homepage à chaque fois que le serveur considère l'utilisation des cookies comme centrale.

Un raisonnement similaire peut être appliqué à une variable de session.

Sauf cas spécial, conséquence d'un code serveur web dédié, il n'est pas utile de "construire" les cookies et les variables de session. L'accès à une page web quelconque d'un site se limite alors à être capable de simuler les paramètres de type GET et POST.

Si malgré tout il fallait effectivement construire des cookies, il faut avoir en tête que ce ne sont que des cas particuliers de paramètre POST. Pour ce qui est des variables de session, une astuce consiste à accéder à la hompege du site, forçant ainsi le serveur web à créer une variable de session, puis ensuite seulement accéder à la page recherchée. Il faut avoir compris que le fait d'accéder à la homepage pour constituer cette variable de session, sur laquelle nous n'avons pas accès, se fait par une requête web non visuelle. C'est en pratique totalement transparent.

Par conséquent, pour faire simple on peut constituer une base de données dans laquelle on associe à chaque page web l'ensemble des paramètres de type GET et POST associés.

On peut imaginer une application permettant de fournir un accès à une page quelconque d'un serveur web, en consultant cette base de données et en construisant un lien URL complexe reprenant l'ensemble de ces paramètres.

Les questions à traiter sont :

- je connais l'URL et les paramètres type GET et POST. Comment fais-je pour accéder à la page web en question, sachant que si je tape l'URL dans un navigateur, le serveur web m'indique qu'il y a une erreur ?
- peut-on imaginer une application simple à utiliser ?
- je ne connais pas les paramètres type GET et POST. Comment puis-je les obtenir ?
- peut-on envisager une automatisation de l'extraction des paramètres type GET et POST ?

2. Accès à une page web en connaissant les paramètres GET et POST

2.1 type GET

Comme on l'a expliqué, les paramètres GET font partie d'une URL (Uniform Resource Locator). En fait, la plupart des URLs sont simples, elles identifient une page .html, une image .jpeg, bref une ressource.L'URL la plus générale est composée d'une URI (Uniform Resource Identifier) suivie éventuellement d'un point d'interrogation et d'une suite de paires *variable* = *valeur*.De ce fait, puisque les navigateurs permettent à tout visiteur de taper au clavier une URL la plus générale, les navigateurs peuvent accéder à une page web correspondant à une URL la plus générale, et en l'occurrence formée de la ressource à laquelle sont adjoints des paramètres.Les paramètres type GET se voient dans la ligne URL du navigateur web, et aussi dans le code source HTML final.

2.2 type POST

Les paramètres POST sont à peine plus complexes à mettre en oeuvre. En effet, ils sont générés notamment par la validation de formulaires. Un exemple peut être plus parlant qu'un long discours :

Supposons que nous voulions transmettre à l'URL "http://www.microsoft.com" le paramètre pwd, de telle manière qu'il soit associé à la valeur mypassword. Il suffit de créer le code HTML suivant :

```
<form action="http://www.microsoft.com"
      method="POST" name="myform">
    <input type="text" name="pwd" value="mypassword"></input>
    <input type="submit" value="Submit"></input>
</form>
```

Ce formulaire permet, lors de son envoi (validation), d'appeler l'URL http://www.microsoft.com en passant l'entête invisible **pwd = mypassword**.

2.3 Combinaison GET et POST

Raisonnons de la manière la plus générale. Nous devons transmettre des paramètres type GET et des paramètres de type POST. Rien de plus simple, il suffit d'indiquer dans l'attribut **action** de notre formulaire une URL complexe avec les paramètres souhaités. Exemple :

```
<form action="http://www.microsoft.com?id=53&subscriber=no" method="POST" name="myform">
   <input type="text" name="pwd" value="mypassword"></input>
   <input type="submit" value="Submit"></input>
</form>
```

Ce code permet d'accéder à l'URL http://www.microsoft.com?id=53&subscriber=no en transmettant l'entête invisible pwd = mypassword. Techniquement, cette URL correspond soit à la ressource http://www.microsoft.com (elle-même souvent remappée à l'aide de la page par défaut http://www.microsoft.com/index.html) à laquelle on passe une ligne de commande : id=53, subscriber=no, soit il s'agit d'une page unique sur le site de microsoft. Tout dépend en fait du serveur web, de sa configuration, de son système de script, et de la redirection CGI.

Le code HTML ci-dessus suppose que l'utilisateur valide à la main le formulaire. On peut aisément passer outre cet aspect manuel à l'aide de quelques lignes en javascript :

```
<html>
<body>

<form >
   ...
</form>

<script language="Javascript">
   // automatically executed during HTML parsing (IE + Netscape compatible)
   document.all.forms["myform"].submit;
</script>

</body>
</html>
```

Détails techniques :

- les variables ont un nom et ce nom est composé de lettres et/ou de chiffres. Mais un nom de variable ne doit pas commencer par un chiffre, et ne doit pas comporter d'espace, ni de caractères spéciaux comme +, \, *, …
- par analogie, les valeurs sont des chaînes alphanumériques. Ici, il n'y a pour ainsi dire aucune contrainte sur les espaces et les caractères spéciaux. Mais il faut malgré tout veiller à la bonne transmission des caractères accentués : **é** doit-il être remplacé par **&ecute;** ou même **è** ?
- lorsqu'une valeur est passée dans un paramètre de type GET, les caractères espaces doivent être remplacés par des caractères **+**. mypassword devient alors my+password. Le + est transformé par %2E. Dans les interfaces de programmation, on trouve souvent une méthode appelée URLEncode() affranchissant le développeur de ce détail.

3. Application simple

Si on suppose que l'on dispose d'un outil capable d'extraire automatiquement les paramètres de type GET et POST, il est évident que la tentation est grande de constituer une base de données.

Mais dans quel but ?

Un intérêt évident est de bypasser un certain nombre de pages d'un site avant d'arriver à une page qui nous intéresse. Imaginons que vous êtes intéressé par une certaine rubrique d'un portail et que cette rubrique est disponible à partir de la homepage, mais seulement après 3 clicks sur 3 pages successives. La première fois, vous découvrez, donc pas de problème. A partir de la seconde fois et proportionnellement à la fréquence d'utilisation, vous allez vous dire que vous aimeriez bien un moyen d'accès directement à la rubrique. Vous allez essayer, une fois sur la rubrique, de créer un bookmark.

Vous quittez le navigateur, le relancez, puis sélectionnez le bookmark. Mais le serveur vous affiche une sorte de page d'erreur. Pas très compréhensible mais une chose est claire : le bookmark ne fonctionne pas.

La solution ? extraire les paramètres type GET et POST permettant d'accéder manuellement à la rubrique, les stocker dans une base de données.

Chaque nouvelle entrée dans la base de données est en quelque sorte un bookmark enrichi.

Il suffit alors d'un logiciel idoine pour accéder à la rubrique. Ce logiciel est soit un standalone (externe au navigateur), soit c'est un plugin ou une extension de navigateur, soit encore c'est une page d'un service web. Le code HTML donné ci-dessus explicite une solution logicielle basée HTML. Difficile de faire plus simple.

Ce faisant, vous accédez à la rubrique.

Plus intéressant encore, en avez-vous assez de remplir des formulaires de login pour accéder à une partie privée d'un site, par exemple un compte email comme hotmail ? Si oui, lisez ce qui suit.

Le même principe permet d'entrer une fois pour toutes les données du formulaire, de le valider, puis d'ajouter une entrée dans la base de données.

Allons plus loin, imaginons que vous passiez votre temps à remplir des formulaires web pour vous enregistrer : achat, questionnaire, profil, ... Imaginons que vous désiriez automatiser le processus. Est-ce possible à l'aide de notre principe ?

Oui, a priori. En fait, il s'agit avant tout de savoir si le formulaire est en une seule partie, ou en plusieurs. Et s'il est en plusieurs parties, est-ce que seules les données de la partie courante sont enregistrées ou est-ce que toutes les données des parties précédentes sont cumulées et transportées (de façon invisible) ?

Si chaque formulaire intermédiaire enregistre les données de manière indépendante, alors si le but est de traverser le formulaire de façon automatique, il faut créer une sorte de macro mettant en boucle le principe simple vu dans les cas précédents. En principe, cela se traduit par un véritable logiciel.

Un exemple concret. Les sites d'offres d'emploi. Sur un site comme Monster, lorsqu'on est candidat on peut à la fois faire une recherche selon certains critères, ou déposer son CV.

- Faire une recherche selon certains critères met en jeu un formulaire très simple en une seule étape. Le principe de base est valable : on peut créer une entrée dans la base de données sans difficulté.
- Déposer son CV met en jeu un formulaire complexe et en plusieurs étapes. Il faut pouvoir extirper tous les paramètres GET et POST mis en jeu. Il n'y a pas de difficultés, c'est simplement long et laborieux. Une fois toutes les entrées dans la base de données obtenues (il y en a autant que d'étapes dans le formulaire), alors il suffit de les regrouper sous un seul nom : Dépot CV Monster. Le top du top : les entrées dans la base de données sont paramétrables : dans ces conditions, on peut automatiser son dépot de CV à l'aide d'une base de données créée une fois pour toutes, et un ou plusieurs paramètres

externes, fixés à l'aide d'une interface homme machine simple (de type formulaire !!). En fait, à l'aide d'une solution basée HTML, on remplace un formulaire complexe en formulaire très simple : gain de temps, et de tranquilité. Par ailleurs, on n'ose imaginer les effets secondaires de ce système : être capable de déposer un CV de façon automatique peut encourager certaines personnes à s'enregistrer un certain nombre de fois sous des noms différents, et de fait constituer des bases de données polluées d'informations pas ou peu exploitables et peu crédibles. Dans la mesure où les CVs sont à l'origine de flux importants d'emails envoyés aux employeurs (qui payent pour ce service), on imagine assez bien le trouble que cela peut causer. Mais après tout, c'est le web et rien d'autre !!

4. Extraction des paramètres GET et POST

4.1 Méthodes manuelles

4.1.1 URL et Code source HTML

Les paramètres GET se récupèrent par lecture directe de l'URL courante, affichée dans le navigateur. Ils se récupèrent également en inspectant le code source HTML de la page précédent celle que l'on désire "tagger". En effet, puisque pour arriver sur la page en question, vous faites un click, cela veut dire que la page précédent la page cible contient un tag lien hypertexte qu'il suffit de trouver : l'URL se trouve dans l'attribut **href**. Exemple : Click here.

Cette URL peut être très complexe.

La chose peut se compliquer un petit peu si la page web est composée de plusieurs cadres intermédiaires (un cadre est un objet HTML capable de séparer sur l'écran deux pages web, ou plus, dans le sens horizontal ou vertical). Dans ce cas de figure, l'URL indiquée par le navigateur ne correspond pas nécessairement à l'URL que l'on croit. C'est même souvent le cas. Fort heureusement, de nombreux sites n'utilisent pas les cadres. Dans tous les cas de figure, il est possible d'inspecter le code HTML, et remonter à la partie de code qui définit les cadres.

Pour ce qui concerne les paramètres POST, il faut passer par l'analyse du code source HTML. Les paramètres POST passent essentiellement dans les formulaires. Cf la première partie de ce document. Il faut trouver les tags **form**, et être capable de distinguer dans les tags **input** ce qui correspond aux données transmises. C'est un travail de bidouilleur, aussi préfère-t-on une méthode automatisée, ou semi-automatique !

4.1.2 Analyseur de trame

L'analyseur de trame permet de semi-automatiser le processus d'extraction des paramètres GET et POST.

D'abord, en quoi consiste un analyseur de trame ? Lorsque vous cliquez sur un lien, ou validez un formulaire, ou entrez une nouvelle URL dans la zone de saisie, vous générez une requête. Cette requête est en fait une ligne de commande qui passe sur la ligne réseau sur un certain port. Sur le web, on utilise le port 80 pour véhiculer les requêtes HTTP et les pages web.

Aussi, on peut très bien imaginer un petit outil qui lirait tout ce qui passe sur la ligne sur un certain port. En l'affichant à l'écran ou dans un fichier, nous saurions à la fois tout ce que le navigateur envoie au serveur web (préparez-vous mentalement, car ça fache!), et aussi tout ce que le serveur web renverrait au navigateur. D'ailleurs le serveur web renvoie essentiellement, sur demande, le code HTML d'une page web, précédé d'entêtes. Parmi ces entêtes, on trouve l'identifiant du serveur (Apache, IIS, ...), la taille de la page web, la date de dernière modification de cette page. Exploiter cette date permet de mettre en oeuvre un service qui vous abonne et vous notifie (par email, ...) à chaque fois qu'une page web donnée est mise à jour. Sur le web, ce service est proposé notamment par NetMind. Voici un mécanisme démystifié.

Se "plugger" sur la ligne oblige à développer un logiciel très bas niveau. Il y a des astuces permettant d'éviter cela. En effet, il suffit d'utiliser un rewebber. Un rewebber écoute sur un port les données qui passent, et les

transmet tel quel sur un autre port. Si l'on configure son navigateur web pour envoyer les données sur un port autre que 80, disons 8081 pour fixer les idées, et que l'on configure le rewebber pour écouter sur le port 8081, et transmettre sur le port 80, nous avons de nouveau un navigateur web qui fonctionne.

Pour dire au navigateur d'envoyer les données sur le port 8081, il y a deux possibilités : ponctuellement et systématiquement. Ponctuellement, il suffit d'ajouter un suffixe aux URLs : transformez http://www.microsoft.com par http://www.microsoft.com:8081. Systématiquement, il faut changer les options de connection à Internet : menu Préférences, champ Proxy, indiquez (ou remplacez) le serveur proxy et son port par l'identifiant du rewebber. Exemple : remplacez proxy.clubinternet.fr, port 80, par 127.0.0.1, port 8081.

127.0.0.1 est une adresse spéciale qui indique à la couche réseau de votre machine que le rewebber est disponible localement sur cette machine.

Si ces configurations sont mises en place, mais que le rewebber n'est pas opérationnel, il est évident que votre accès au web ne fonctionnera pas. Le web seulement d'ailleurs, car l'email, les newsgroups, le FTP, l'IRC, ... discutent sur d'autres ports, non impactés.

Un rewebber est en fait un serveur de relai local. Son intérêt est de pouvoir afficher à l'écran et/ou dans un fichier tout ce qui passe sur la ligne. Un rewebber de qualité permet de facilement comprendre les données, car elles sembleront d'abord strictement incompréhensibles. Le rewebber doit notamment clairement afficher le sens de transport des données : client vers serveur, ou serveur vers client.

Exemples de rewebber disponibles : Tcp viewer, IP monitor.

Revenons au principal, en quoi le rewebber permet de semi-automatiser la collecte des paramètres de type GET et POST ? En fait, tout reste manuel car il ne suffit pas avec rewebber de deux ou trois clics pour que le travail soit terminé !

Mais là où nous devions précédemment collecter les URLs complexes lorsqu'elles passent dans la zone adresse du navigateur, et extirper des URLs du code source HTML, réputé pour être vraiment incompréhensible et vous forçant à faire de nouvelles recherches de paramètres à chaque nouvelle page, sans parler des paramètres POST, le rewebber fournit une approche homogène et standard qui affiche tout : les URLs (les paramètres GET), les entêtes (les paramètres POST) et même les cookies.

4.2 Méthodes automatiques

La méthode automatique consiste à utiliser le rewebber décrit précédemment et l'interfacer avec un outil de "datamining" très simple, capable de filtrer les données importantes dans le flux de données, d'extraire les paramètres GET et POST, et de les stocker dans une base de données.

Typiquement soit le rewebber est "ouvert" et interfaçable avec un autre logiciel, soit il faut développer un rewebber dédié dont une fonctionnalité est le stockage dans une base de données des paramètres pertinents. S'il faut le développer, on ne parle alors plus de rewebber, il s'agit bel et bien d'un "analyseur" de site (webh4ck).

A défaut de développement, une façon de s'interfacer à un rewebber est de lire et analyser en permanence et en temps réel le fichier qu'il génère.

5. Automatisation de l'extraction

Ce qui suit décrit le modèle de données utilisé dans la base de données.

5.1 Une URL complexe

Les paramètres de type GET et POST sont, on l'a vu, des paires *variable* = *valeur*. Dans ces conditions, on

doit les distinguer. Voici une proposition :

- pour un paramètre de type GET, si nous voyons passer par exemple id=53, modélisons la variable en lui ajoutant un préfixe **get**_, de façon à ce que du point de vue de la base de données, nous stockons get_id = 53.

- pour un paramètre de type POST, issu d'un formulaire, ajoutons un préfixe **post**_
- pour un paramètre de type POST, issu d'un cookie, ajoutons un préfixe **cookie**_

De plus, chaque variable est persistante ou non lors de la navigation sur un site. C'est-à-dire qu'elle apparait depuis la homepage et sur toutes les autres au fur et à mesure des clicks.

Nous arrivons au modèle de données suivant :

Champ	Type	Description
Id	INT32	Identifiant unique pour l'URL
Name	VARCHAR(255)	Nom pour l'utilisateur et l'application utilisant la base de données
URL	VARCHAR(1023)	URL : identification de la ressource (page web, image, CGI, ...)
Varname	VARCHAR(255)	Variable
Varvalue	VARBLOB()	Valeur de la variable
Persistant	BOOL	Variable persistante sur le site
KeyGen	Enum	La valeur est issue d'un générateur de clef ? 0 = non, 1 = numérique, 2 = alphanumérique

Pour l'exemple étudié précédemment (repris ci-dessous), les données obtenues sont :

```
<form action="http://www.microsoft.com?id=53&subscriber=no" method="POST" name="myform">
   <input type="text" name="pwd" value="mypassword"></input>
   <input type="submit" value="Submit"></input>
</form>
```

Id	Name	URL	Varname	Varvalue	Persistant	KeyGen
1	Microsoft	http://www.microsoft.com	get_id	53	0	0
1	Microsoft	http://www.microsoft.com	get_subscriber	No	0	0
1	Microsoft	http://www.microsoft.com	post_pwd	Mypassword	0	0

5.2 Une combinaison d'URLs complexes

On a évoqué les formulaires en plusieurs étapes. Automatiser l'envoi de données sur un tel formulaire revient à manipuler une combinaison d'URLs complexes. Un modèle de données d'une autre table de la base de données doit donc associer plusieurs URLs complexes, à l'aide de leurs identifiants Id, un ordre de prise en compte, ainsi que certaines particularités car chaque site est potentiellement un cas particulier.