Vulnérabilités et solutions de sécurisation des applications Web

Patrick CHAMBET
EdelWeb – ON-X Consulting
patrick.chambet@edelweb.fr
http://www.edelweb.fr
http://www.chambet.com

Eric Larcher
RSSI Accor Services
info@internet-securise.com
http://www.internet-securise.com

Aujourd'hui, rares sont les DSI qui accepteraient de mettre en ligne une application Web sans s'être assuré qu'un minimum de sécurité a été mis en place.

En particulier, l'installation d'un firewall, afin de protéger des attaques venant d'Internet une architecture Web est entrée dans les mœurs, en tout cas dans les entreprises disposant d'un budget informatique significatif. De même, la mise en place d'un certificat SSL, de façon à protéger la transmission d'informations sensibles (logins, mots de passe, etc.), entre un navigateur web et un serveur, est de plus en plus fréquente.

Malheureusement, ce type de protection n'est plus suffisant. Comme nous le montrerons tout au long de cet article, de nombreuses attaques, très préjudiciables à une application Web, peuvent être réalisées avec succès, même à travers un firewall correctement configuré. Et l'installation d'un certificat SSL n'empêche pas non plus ce type d'attaques.

Nous allons, dans cet article et après de brefs rappels sur le fonctionnement d'une application Web et d'un firewall, détailler, exemples à l'appui, les principales attaques pouvant affecter une application Web. Nous traiterons également des différentes manières de contrer ces attaques avant de conclure sur les systèmes de filtrage applicatif de type « reverse-proxy ».

Anatomie d'une application Web

Une application Web est une application qui ne s'appuie que sur le protocole HTTP ou HTTPS afin d'être pilotée par un utilisateur distant. Ce dernier n'a besoin que d'un simple navigateur Web ou d'une application propriétaire utilisant le protocole HTTP/HTTPS.

L'avantage des applications Web est que l'utilisateur peut se situer très loin de l'applicatif et travailler à travers Internet, au besoin via un VPN chiffré (type IPsec).

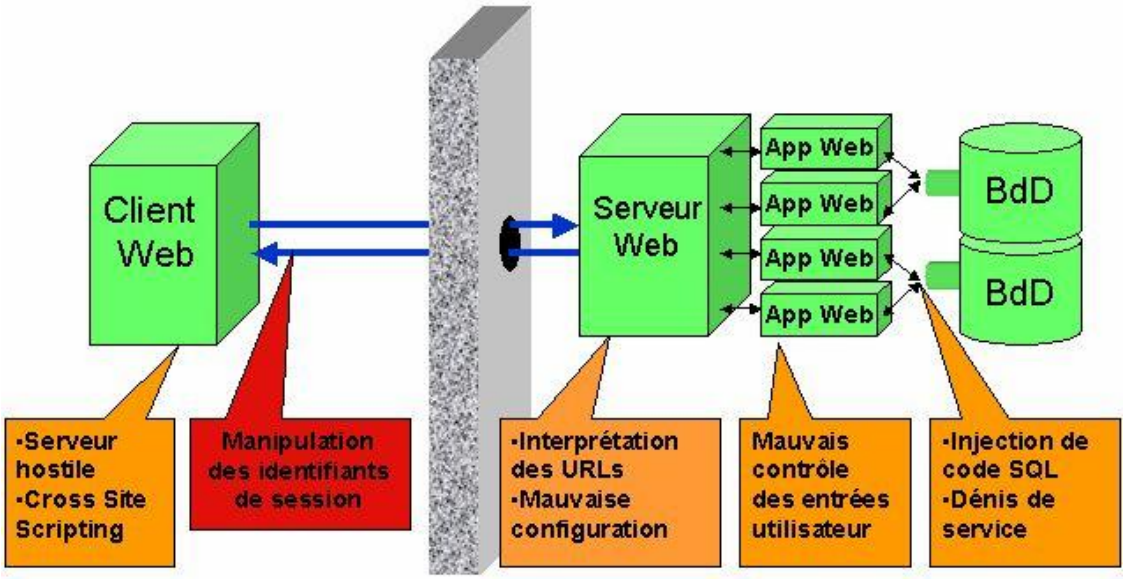
Remarque : contrairement à une idée reçue très répandue, l'utilisation de SSL ne suffit pas à protéger une application Web. Le chiffrement SSL (sans utilisation de certificats clients X.509) ne concerne que la confidentialité, et ne protège pas des **intrusions**.

Pour permettre l'utilisation d'une application Web, il suffit que le firewall protégeant celle-ci ne laisse entrer que le protocole HTTP, en général sur le port TCP 80 (et éventuellement HTTPS sur le port TCP 443). Dans ces conditions, il semble difficile d'attaquer une application Web. Pourtant, nous allons voir qu'à travers ce seul port 80, considéré comme « amical » mais devenu un port « fourre-tout » par lequel passent de plus en plus de flux et de protocoles (DCOM, RPC, SOAP,

XML, streaming sur HTTP, …), il est possible de lancer des attaques extrêmement dangereuses.

Les différentes sortes d'attaques sur les applications Web sont les suivantes :

- Interprétation des URLs
- Mauvais contrôle des données entrées par l'utilisateur
- Injection de code SQL
- Attaques sur les identifiants de session
- Cross Site Scripting (XSS)
- Autres attaques



Les composants d'une application Web et leurs vulnérabilités

1. INTERPRETATION DES URLS

Les URLs utilisées dans le cadre d'une application Web sont du type :

`http://www.domaine.fr/chemin/fichier.ext?param1=x&param2=y`

Une telle URL est composée de plusieurs parties :
- **http://** : protocole utilisé
- **www.domaine.fr** : adresse du serveur (FQDN)
- **chemin** : arborescence de répertoires sur le serveur Web
- **fichier.ext** : fichier lu ou exécuté (son extension `.ext` est très importante)
- **param1** et **param2** : paramètres d'exécution, interprétés soit au niveau des composants métiers, soit directement au niveau de la base de données.

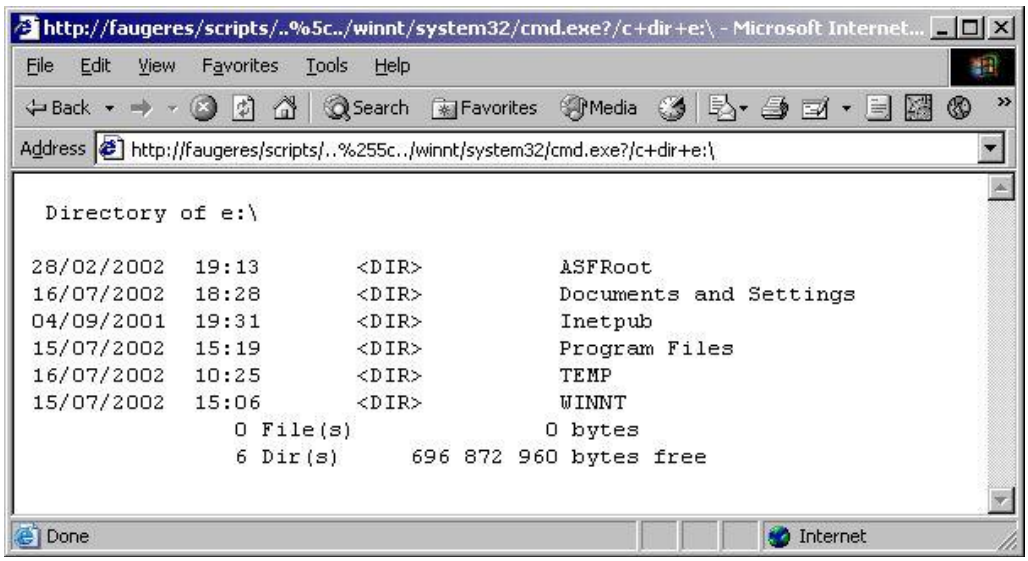
Chacune de ces parties est susceptible d'être attaquée :

- **Protocole** : on peut par exemple essayer de remplacer le protocole https:// par http:// afin de désactiver une authentification par certificat client.

- **Serveur** : on peut le remplacer par son adresse IP ou par les noms de domaines d'autres sites hébergés sur le même serveur, afin d'avoir accès à d'autres parties du site.

- **Chemin** : on peut tenter de naviguer dans l'arborescence pour accéder à des parties du site non autorisées ou pour remonter dans l'arborescence par l'utilisation de « `/../../` », ou en utilisant des vulnérabilités particulières (le bug Unicode d'IIS, par exemple).

Exemple 1 :



Une illustration du bug Unicode d'IIS

Exemple 2 :

L'URL suivante : http://www.vulnerable.com/////////////////…////////////////////// permettait, sur un serveur Web Apache, de récupérer la liste des fichiers du répertoire racine, même s'il existait un fichier par défaut (`index.html`).

- **Fichier** : son extension va déterminer de quel type d'exécutable il s'agit : CGI, scripts ASP, HTR ou autre code exécutable, etc… Plusieurs types de fichiers ont connu des vulnérabilités attachées à leur mode d'exécution, et en particulier à leur interpréteur. Ainsi, c'est une vulnérabilité dans le filtre ISAPI d'IIS, qui interprète les fichiers .IDA/.IDQ, qui a permis la propagation du ver CodeRed.

  Paramètres : la manipulation des noms de paramètres et de leur contenu peut conduire à des effets dangereux. Cela sera abordé plus en détail au paragraphe 2.

Les conséquences de ces attaques peuvent être la divulgation d'informations importantes, l'accès au code source des scripts ou au contenu des fichiers, l'exécution de commandes sur le serveur, ou même l'obtention d'un shell sur le serveur, éventuellement sous un compte privilégié (SYSTEM sur Windows NT/2000/XP/2003, par exemple).

Pour contrer les attaques ci-dessus, il convient de prendre en compte les recommandations suivantes :

- Sécuriser le système d'exploitation et le serveur Web (appliquer en particulier les derniers patches de sécurité, chrooter le service, …)

- Installer l'arborescence Web sur une partition dédiée
- Contrôler strictement l'arborescence Web et supprimer les répertoires inutiles
- Désactiver le « directory browsing » sur l'ensemble du site Web
- Supprimer tous les filtres, interpréteurs de scripts, CGI et autres exécutables inutiles
- Supprimer tous les fichiers inutiles sur un serveur de production, notamment les pages d'exemples
- Appliquer des permissions d'accès strictes sur les fichiers au niveau du serveur Web mais aussi du système de fichiers. En particulier, l'utilisateur anonyme ne doit avoir que des permissions en lecture sur les pages statiques
- Utiliser un filtre d'URLs (par exemple le module mod_rewrite pour Apache ou le filtre ISAPI URLScan pour IIS) et appliquer des règles strictes afin de contrôler, de réécrire ou d'interdire toutes les URLs contenant des caractères dangereux comme les caractères Unicode
- Installer un reverse proxy (module mod_proxy d'Apache, par exemple) : voir le paragraphe 6 sur le filtrage applicatif
- Enfin, envisager l'installation d'un IDS sur la DMZ hébergeant le serveur Web et/ou le reverse proxy, afin de détecter les attaques classiques comme la remontée dans l'arborescence Web.

2. MAUVAIS CONTROLE DES DONNEES ENTREES PAR L'UTILISATEUR

Au cours de nos audits et tests d'intrusion, nous rencontrons encore trop souvent des applications Web où le contrôle des entrées par l'utilisateur se fait uniquement côté client. Dans ce cas, il faut toujours considérer qu'un attaquant pourra outrepasser le contrôle effectué de son côté et donc envoyer les données qu'il désire au serveur, à l'aide d'un proxy intrusif par exemple. La règle à adopter est de **ne jamais faire confiance à du code tournant côté client**, comme des applets Java par exemple. En effet, ce code pourra toujours être décompilé, analysé et modifié pour effectuer les tâches voulues par un attaquant.

Considérons le cas d'une page Web permettant d'effectuer un virement bancaire. Par précaution, la banque limite les virements par Internet à 10000 Euros. Un script Javascript contenu dans la page HTML contrôle en temps réel le montant saisi et empêche toute validation avant que le montant soit correct.

Dans ces conditions, il suffit de modifier le script, ou même simplement de désactiver Javascript, pour pouvoir saisir le montant voulu et valider le formulaire.

La recommandation qui s'impose est donc de toujours contrôler la validité des données fournies par l'utilisateur *au moins* côté serveur. Ce contrôle peut bien sûr être doublé côté client pour des raisons d'ergonomie, mais le contrôle final devra toujours avoir lieu côté serveur.

D'autre part, il existe des caractères spéciaux dont la signification, au sein d'une chaîne alphanumérique, peut respecter une syntaxe particulière au niveau d'un langage de programmation ou d'un système d'exploitation, ce qui peut conduire à l'exécution de commandes hostiles. Ces caractères figurent parmi les suivants :

! @ $ % ^ & * ( ) - _ + ` ~ \ | [ ] { } ; : ' " ? / , . > <

Ainsi, le caractère « * » est interprété sur UNIX comme l'ensemble des fichiers contenus dans le répertoire courant. De même, le caractère « ; » est un séparateur, les caractères « < » et « > » effectuent des redirections, « % » permet d'entrer des caractères par leur code hexadécimal, etc…

Il est donc indispensable de filtrer ces caractères à l'intérieur des données fournies par l'utilisateur, en les codant en leurs équivalents en HTML (« > » devient &gt ; , « < » devient &lt ;, etc…), ou en leur ajoutant un caractère d'échappement (« \ » par exemple), ou encore en les supprimant, ou enfin en refusant purement et simplement la transaction en demandant à l'utilisateur de modifier sa saisie.

Pour résumer, les recommandations nécessaires pour contrer la saisie de données hostiles par un utilisateur sont les suivantes :

- Nécessité d'un double contrôle côté client **plus côté serveur**
- Comptage du nombre de paramètres et de leur nom
- Neutralisation des caractères spéciaux
- Contrôle de la longueur des données
- Validation du type des données (date, chaîne, nombre)
- Contrôle de l'intervalle de validité des données (dans l'absolu)
- Vérification de la validité réelle des données (en relatif, dans une base de données)
- Limitation du nombre de saisies de données par unité de temps.

  Ceci permet d'éviter les attaques de type force brute et peut s'implémenter par exemple en multipliant à chaque fois le temps d'attente avant la fin de la transaction par deux entre deux transactions identiques.

3. INJECTION SQL

L'injection SQL peut être une conséquence directe d'un mauvais contrôle des données entrées par l'utilisateur. En effet, les caractères « ' » et « ; » peuvent être utilisés pour enchaîner plusieurs requêtes SQL à la suite l'une de l'autre. Considérons par exemple une page HTML comprenant un formulaire d'authentification avec un champ Login et un champ Password. La requête SQL tournant sur le serveur est la suivante :

```
SELECT user FROM table_users WHERE champ_login='login' AND
champ_password='password'
```

Si maintenant un attaquant saisit la chaîne suivante dans le champ Login :

```
Administrateur'; --
```

La requête exécutée finalement sera la suivante:

```
SELECT user FROM table_users WHERE
champ_login='Administrateur'; --' AND
champ_password='password'
```

Le résultat est un contournement de l'authentification : on se retrouve logué en tant qu'Administrateur.

Le cas le plus simple d'injection SQL consiste à s'authentifier dans une application Web en saisissant un login existant et n'importe quel mot de passe suivi de « `OR 1=1` ». L'authentification étant vérifiée par comparaison, le résultat booléen de la vérification du mot de passe est toujours vrai, ce qui permet d'obtenir l'accès à l'application.

Il faut noter que le filtrage des caractères spéciaux ne suffit pas à se protéger contre l'injection de code SQL. En effet, considérons la saisie suivante :

```
toto UNION ALL SELECT champ_Password FROM table_Users WHERE champ_Login LIKE admin
```

Aucun caractère spécial (autre que le "_" utilisé pour des raisons de lisibilité) n'est utilisé, et pourtant la requête obtenue récupère la liste des mots de passe des administrateurs.

De plus, certains gestionnaires de base de données offrent des fonctions supplémentaires potentiellement dangereuses. Par exemple, Microsoft SQL Server possède par défaut un certain nombre de procédures stockées d'administration pouvant conduire à des fuites d'information et à des intrusions.

Il existe donc beaucoup de techniques d'injection de code SQL. Pour contrer ce type d'attaques, il est nécessaire d'effectuer un filtrage beaucoup plus précis du contenu des données saisies par les utilisateurs. Il faudra en particulier interdire ou « échapper » les mots clés comme SELECT, INSERT, UNION, LIKE, etc… L'utilisation de fonctions de substitution et d'expressions régulières est ici très utile. Il est préférable d'utiliser des procédures stockées, moins sujettes à l'injection, et ne pas laisser de requêtes SQL dans les pages de script.

Il est nécessaire ensuite de sécuriser la configuration du service de base de données :

- Suppression des comptes inutiles créés par défaut et création de comptes avec des privilèges réduits (tous les utilisateurs authentifiés ne doivent pas utiliser le même compte pour effectuer toutes les transactions dans la base de données)
- Suppression des procédures stockées présentes par défaut
- Application de permissions d'accès en lecture, suppression, exécution sur les tables, les procédures stockées et les autres objets de la base de données.

Il convient enfin de rédiger toutes les requêtes SQL de son application avec soin, en utilisant une syntaxe la plus stricte possible, avec typage systématique (chaînes entourées de « ' » par exemple) et vérifications de conformité aux différents stades de traitement des requêtes, aussi bien au niveau des composants métiers que des procédures stockées dans la base de données.

4. ATTAQUES SUR LES IDENTIFIANTS DE SESSION

Contrairement au paradigme client/serveur traditionnel, le protocole HTTP est un protocole déconnecté. C'est-à-dire qu'entre deux requêtes, la connexion entre le client et le serveur est coupée. Le serveur ne peut donc pas reconnaître un client qui a déjà commencé une transaction dans l'application Web.

Pour remédier à cela, on utilise un **identifiant de session**, échangé à chaque page entre le client et le serveur, que ce soit au niveau du cookie, de l'URL ou d'un champ caché de formulaire. Le serveur maintient un contexte de transaction pour chaque identifiant de session généré.

Une attaque classique consiste à voler la session d'un utilisateur qui vient de s'authentifier sur le système en essayant de deviner la valeur de son identifiant de session. Si la valeur de celui-ci est découverte, un attaquant peut alors se faire passer pour l'utilisateur légitime en injectant l'identifiant récupéré dans sa propre session, à l'aide d'un proxy intrusif par exemple.

Un identifiant de session se présente par exemple ainsi :

```
HTTP/1.1 200 OK
Date: Mon, 17 Jun 2003 11:43:27 GMT
Server: Apache/1.3.26
Set-Cookie: session=0001WVXSDWAACAB4EMYGBIB0NXI; path=/
Content-Type: text/html
```

Si on essaie de récupérer un grand nombre d'identifiants successifs, on peut ensuite effectuer une étude statistique afin de déterminer des vulnérabilités dans la méthode de génération de ces identifiants.

On peut par exemple remarquer que le jeu de caractères utilisé est faible, ou que l'identifiant de session possède des parties fixes ou variant lentement. Après une analyse plus poussée, on peut identifier la façon dont chaque partie de l'identifiant est construite.

Il est souvent possible de développer un outil qui va permettre de diminuer considérablement l'espace de valeurs de cet identifiant, et qui va automatiser la recherche d'une valeur valide de celui-ci. On peut ainsi augmenter les chances de trouver une bonne valeur à une sur 1000 à 3000 parfois, ce qui est extrêmement peu : quelques minutes seulement suffisent pour voler la session d'un utilisateur authentifié.

Il est donc indispensable de vérifier la qualité du générateur aléatoire et l'étendue de l'espace de valeurs des identifiants de session de son application Web. Cette étendue doit être suffisamment grande pour qu'une attaque en force brute ne puisse pas être menée dans un délai réduit.

Il est déconseillé d'utiliser les fonctions de génération d'identifiants fournies en standard avec certains logiciels ou environnements de développement du marché. Il est parfois préférable d'écrire soi-même une fonction de génération des identifiants de session plus robuste, mais qui devra être vérifiée soigneusement.

Il est toujours préférable de prévoir une durée maximale de validité d'une session.

5. CROSS SITE SCRIPTING

Le principe du Cross Site Scripting (ou XSS) est d'attaquer les utilisateurs de l'application plutôt que l'application elle-même. Pour cela, l'attaquant provoque l'envoi à la victime, par le site Web légitime, d'une page hostile contenant du code malveillant.

Cette page est exécutée sur le poste de la victime, dans le contexte du site Web d'origine (zone Internet, sites de confiance, …), et dans le contexte de sécurité de l'utilisateur courant.

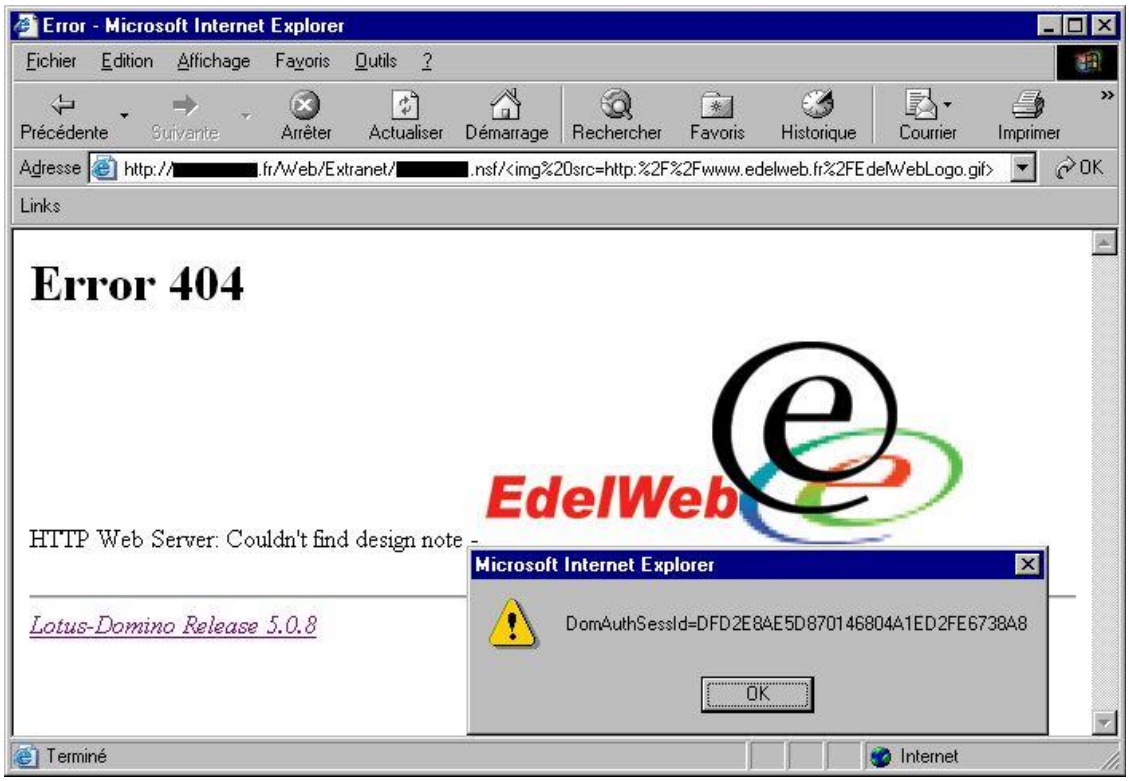
L'exemple le plus simple est celui qui consiste à provoquer une erreur HTTP 404 chez les victimes. Si le site www.banque.com ne filtre pas de façon correcte les données envoyées par les utilisateurs, un attaquant peut saisir le texte suivant sur une page accédée par d'autres utilisateurs :

```
<A HREF=http://www.banque.com/<script>
alert(document.cookie)</script>">Click Here</a>
```

Les internautes recevront la page suivante s'ils ont cliqué sur le lien ci-dessus :

```
<HTML>404 Page Not Found:<script>alert(document.cookie)
</script></HTML>
```

Le script contenu dans cette page d'erreur 404 va provoquer l'affichage des cookies relatifs au site www.banque.com. Il suffit à un attaquant de les récupérer pour se faire passer ensuite pour l'utilisateur légitime auprès du site de la banque.



Exemple de Cross Site Scripting permettant de récupérer le cookie d'un autre utilisateur

Ce type de vulnérabilité est présent sur de nombreux sites, forums et webmails qui interprètent le code javascript présent dans leurs pages. De nombreuses vulnérabilités de ce genre, extrêmement simples à exploiter mais difficiles à filtrer, ont fait les gros titres dans certaines publications. Ces annonces étaient exagérées, mais il est évident que la lutte contre les codes mobiles hostiles est sans fin.

Les précautions à prendre sont les suivantes :

Côté serveur:

- Maintenir le serveur Web à jour (correctifs de sécurité)
- Contrôler la validité des saisies des utilisateurs, notamment les balises `<script>`, les incohérences du type `<img src="javascript:… ">`, etc…

  Ce contrôle peut également être fait au niveau des antivirus côté serveurs effectuant également un contrôle du le flux HTTP.

Côté client:

- Maintenir les navigateurs et clients mail à jour (correctifs de sécurité) ;
- Durcir leur configuration le plus possible.

6. AUTRES ATTAQUES

D'autres attaques traversant les firewalls peuvent être menées contre des applications Web :

Mécanismes d'authentification basés sur Java, JavaScript ou ActiveX :

A éviter : les applications utilisant ces technologies sont sujettes, comme nous l'avons vu, à des manipulations effectuées côté client permettant à un attaquant d'outrepasser le mécanisme d'authentification.

Contrôle d'accès basé sur le header HTTP_REFERER :

A éviter : la manipulation des headers à l'aide d'un proxy intrusif est facile.

Manque de ré-authentification :

Le manque de ré-authentification au niveau de certaines fonctionnalités critiques (comme le changement du mot de passe de l'utilisateur, par exemple) permet souvent à un attaquant de prendre le contrôle d'un compte.

Mauvaise gestion du contexte utilisateur :

La mauvaise gestion du contexte utilisateur peut permettre une augmentation progressive de privilèges conduisant à la prise de contrôle total de l'application par un attaquant. Il convient de contrôler strictement et à chaque page le contexte de sécurité (l'utilisateur est-il authentifié ? Quels droits possède-t-il ?).

Attaques côté client:

Il s'agit d'attaquer directement les utilisateurs de l'application plutôt que l'application elle-même. Ce genre d'attaques est rendu possible principalement par les langages exécutés côté client (VBScript, Flash, DHTML, XML, javascript, Applets Java, ActiveX, CSS, …).

Là encore, il est indispensable de maintenir les navigateurs et clients mail à jour et de les sécuriser le plus possible.

Man in the middle:

Une attaque dite "man in the middle" consiste à intercepter les requêtes du client et à les relayer vers le serveur distant légitime et inversement à intercepter les réponses du serveur et à les relayer vers le client. Il est possible au passage, si nécessaire, de modifier à la volée les données fournies par le client et/ou les réponses du serveur.

Cette attaque est possible même si le serveur de destination utilise un chiffrement par SSL : il suffit que le serveur intercepteur possède lui aussi un certificat serveur et que le client clique sur « Accepter » lorsque son navigateur lui propose d'utiliser ce certificat pour dialoguer avec le serveur distant légitime. C'est ce que fera tout utilisateur peu attentif ou qui ne sera pas au fait des problèmes de sécurité. Il ne

reste plus alors au serveur intercepteur qu'à déchiffrer d'un côté et rechiffrer de l'autre, à la volée.

Le seul moyen de se prémunir contre ce type d'attaque est d'imposer une authentification côté client par l'utilisation de certificats clients X.509. Le serveur intercepteur ne pourra alors plus se faire passer pour le client auprès du serveur distant légitime car il ne dispose pas de la clé privée du client.

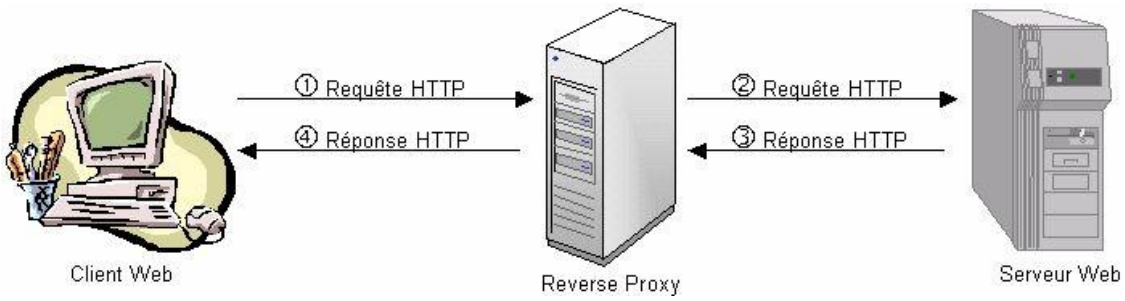
7. FILTRAGE APPLICATIF

Le filtrage applicatif consiste à effectuer un filtrage des requêtes et des réponses non pas au niveau réseau (couche OSI 2 pour Ethernet au niveau d'un switch par exemple, couche 3 pour IP, couche 4 pour TCP et UDP au niveau d'un firewall) mais au niveau application (couche OSI 7). Le filtrage applicatif suppose donc la connaissance de l'application et de ses protocoles afin de connaître la structure des données échangées.

Dans notre cas particulier des applications Web, il s'agit d'interpréter intégralement le protocole HTTP pour en extraire les URLs, les headers, les cookies, les authentifiants, etc…

L'avantage du filtrage applicatif apparaît donc clairement : les règles utilisées à ce niveau peuvent se fonder sur tous ces paramètres de niveau élevé et permettre un filtrage beaucoup plus élaboré qu'un simple filtrage TCP/IP. Il s'agit en fait d'un filtrage de contenu. Celui-ci peut aller jusqu'au filtrage du code mobile hostile véhiculé dans les pages HTML, sous forme de scripts Javascript ou d'ActiveX, comme nous l'avons vu.

Le filtre applicatif HTTP couramment utilisé pour la protection d'une application Web est le « reverse proxy », appelé aussi relais inverse. Il s'agit d'un relais applicatif agissant en coupure, qui va s'intercaler entre le serveur Web et les clients extérieurs. C'est le reverse proxy qui répond aux requêtes des clients, et c'est lui qui effectue les requêtes vers le serveur Web final.



Fonctionnement d'un reverse proxy

Remarque : au niveau du firewall, l'accès direct au serveur Web doit bien sûr être interdit depuis l'extérieur. Seule une règle autorisant l'accès au reverse proxy devra être présente.

Un reverse proxy permet en particulier de faire respecter les contraintes suivantes au niveau de l'accès au serveur Web :

- L'utilisation du protocole HTTP uniquement
- L'accès à une liste de répertoires autorisés seulement
- L'accès à une liste de fichiers autorisés dans certains répertoires
- L'exécution des fichiers dont les extensions sont explicitement autorisées
- L'utilisation de paramètres dont le type et le contenu sont autorisés pour chaque objet métier de l'applicatif.

Actuellement, de nouveaux logiciels de protection des applications Web sortent sur le marché.

8. LOGICIELS DE PROTECTION DES APPLICATIONS WEB

Le nombre considérable d'attaques Web traversant les firewalls a conduit plusieurs éditeurs à concevoir des logiciels de protection des applications Web s'intercalant entre Internet et le serveur Web hébergeant l'application. Ils agissent ainsi comme des reverse proxies « intelligents » et permettent de stopper certains types d'attaques.

Il s'agit en particulier des solutions suivantes :

- InterDo de Kavado
- AppShield de Sanctum
- RealSentry d'Axiliance
- RemoteWeb de Deny-All
- DMZ-Shield d'Ubizen
- NetsecureWeb de Netsecure Software
- APS de Terros

Ces produits permettent de protéger les applications Web contre plusieurs des attaques présentées précédemment. De plus, ces produits permettent une pré-configuration automatique des règles de sécurité et un apprentissage, automatique ou non, des applications Web à protéger (chemins de navigations types entre les pages, par exemple).

Ces solutions ne sont pas encore répandues, mais font actuellement l'objet d'un marketing agressif qui devrait augmenter leur pénétration.

Cependant, ces produits sont encore un peu jeunes et parfois compliqués à configurer, d'autant plus qu'à chaque modification, même mineure, de l'application, il est indispensable de modifier également la configuration du produit. Mais on peut imaginer que si leurs modules d'apprentissage se perfectionnent suffisamment, ces produits rencontreront un grand succès dans un proche avenir, au même titre que les firewalls il y a quelques années. L'avenir nous dira s'ils deviendront aussi incontournables.

Pour conclure, nous avons vu dans cet article que même si un firewall restait un outil de sécurisation indispensable, il n'est désormais plus suffisant afin d'assurer un niveau de sécurité acceptable pour une application Web. Il convient donc aujourd'hui de ne pas s'arrêter à la sécurisation des couches réseau (filtrage, segmentation, etc.) mais d'aller au-delà, notamment au niveau système (durcissement des OS, serveurs Web et SGBD) et surtout au niveau applicatif (règles de développement sécurisé, reverse proxies applicatifs, etc.).

Pour réussir dans ce type de démarche, les problématiques sécuritaires doivent être prises en compte le plus en amont possible dans le projet (phase de conception de l'architecture applicative) et être suivie tout au long de la vie de l'application (nouvelles versions, patches de sécurité, etc.).

Enfin, un audit effectué par une société externe est quasiment obligatoire afin de contrôler les mesures mises en place, tout au moins pour les sites les plus sensibles.

Patrick CHAMBET - http://www.chambet.com

Consultant Senior en sécurité Windows NT/2000/XP/2003, audits et tests d'intrusion.

Edelweb - http://www.edelweb.fr

Eric LARCHER

Responsable de la Sécurité des Systèmes d'Information, Accor Services.

Auteur de l'ouvrage « L'Internet Sécurisé » (Eyrolles).

http://www.internet-securise.com