Mini Project - Defend

Secure the application

In this first part of the assignment we fixed all the forms to prevent any sort of SQL injection by sanitizing the input. the image below shows how we fixed it in the login form:

```
statement = "SELECT * FROM users WHERE username = ? AND password = ?;"
c.execute(statement, (username,password))
```

instead of the previous sql-injectable:

```
statement = "SELECT * FROM users WHERE username = '%s' AND password = '%s';" %(username, password)
c.execute(statement)
```

we did the same over all the queries (login; register - both for the check and the insertion; notes - for inserting a new note, retrieving the previously written one, and importing a new note) in the app.py.

We modified the file to avoid it from running in debug mode which could show to potential attacker useful error messages:

```
try:
    app.run(host='0.0.0.0', port=runport, debug=False)
```

we eliminated useless check in the register function which could give an attacker information about someone else's password:

```
pass_statement = """SELECT * FROM users WHERE password = ?;"""
c.execute(pass_statement,(password,))
if(len(c.fetchall())>0):
    errored = True
    passworderror = "That password is already in use by someone else!"
```

then we removed the redundant print by commenting them, for example:

```
statement = "SELECT * FROM notes WHERE assocUser = ?;"
c.execute(statement, (session['userid'],))
#print(statement )
notes = c.fetchall()
#print(notes)
```

We thought that notes could be used by attackers to perform Cross-Site Scripting attacks so we tried to use the bleach library, where we retrieve the notes we added the following code

```
for note in notes:
    note['note'] = bleach.clean(note['note'])
```

unfortunately we were not able to install the bleach library correctly on the server so instead we did this:

```
for row in rows:
    note = {
        'id': row[0],
        'assocUser': row[1],
        'dateWritten': row[2],
        'note': row[3],
        'publicID': row[4]
    }
    notes.append(note)
for note in notes:
    note['note'] = clean_html_tags(note['note'])
```

Doing so our notes cannot contain html tags so it is not possible to write html script in the notes.

SETUP

We set up an Apache/2.4.52 (Ubuntu) web server which is hosting the Flask application. Furthermore, we only enabled port 5000 for access to the application. Because we wanted to limit the risks, we also allow ONLY TLS connection on it. However, because our self-signed certificate is not trusted (not signed by a trusted CA) , the user needs to accept the risk in order to access it. We decided to keep the TLS requirement anyway, since the data exchanged between the server and client are still encrypted nonetheless that the browser marks the connection as not secured.

VULNERABILITIES

Our initial goal was to add the Remote Code Execution backdoor vulnerability (RCE-CVE). We found out that the vsftpd 2.3.4 had that vulnerability between 30.06.2011 and 03.07.2011. Namely, the backdoor shell was available on port 6200 after the user used a username that ended with `` `:)` `` to log in the ftp server. We were also able to find the original source code. We successfully compiled the man file which was part of the source folder and generated a `vsftpd` binary file which was used to create an infinitive service running on the host's port 21 (as can be seen on the image below).
However, when trying to exploit it the ftp program freezed when we tried to insert the username that ends with `` `:)` ``. We also confirmed that the vulnerability cannot be exploited used pre-loaded kali LUA script (`ftp-vsftpd-backdoor.nse`) using command: `` `nmap --script ftp-vsftpd-backdoor.nse  192.168.23.103 -p 21` ``, as can be seen on the image below.

```
┌──(kali㉿kali)-[~]
└─$ nmap -sV 192.168.23.103
Starting Nmap 7.94 ( https://nmap.org ) at 2023-11-11 13:14 EST
Nmap scan report for stud103 (192.168.23.103)
Host is up (0.034s latency).
Not shown: 995 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 2.3.4
22/tcp   open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.3 (Ubuntu Linux; protoco
l 2.0)
80/tcp   open  http    Apache httpd 2.4.52
443/tcp  open  http    Apache httpd 2.4.52
5000/tcp open  http    Apache httpd 2.4.52
Service Info: Host: stud103.itu.dk; OSs: Unix, Linux; CPE: cpe:/o:linux:linux
_kernel

Service detection performed. Please report any incorrect results at https://n
map.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.52 seconds
```

```
┌──(kali㉿kali)-[~]
└─$ nmap --script ftp-vsftpd-backdoor.nse 192.168.23.103 -p 21
Starting Nmap 7.94 ( https://nmap.org ) at 2023-11-11 06:14 EST
Nmap scan report for stud103 (192.168.23.103)
Host is up (0.027s latency).

PORT   STATE SERVICE
21/tcp open  ftp

Nmap done: 1 IP address (1 host up) scanned in 1.41 seconds
```

Nonetheless, we still let the `vsftpd` server run since it can be used as a diversion when another group tries to exploit our server.

We decided to hide (leak) the credential for the user (stud) access of our server into the web application. For that purpose we added the link to the new page on the footer of the index page saying "`do you want password?`". On the new page we set up "fake" form which prompts user to try and exploit it using SQL syntax, however the form is not connected to any DB and uses simple JS program to generate random strings as return to the valid SQL inputs (we verify that the inputted query is syntactically correct). This form serves as a diversion for attackers to not be able to find where the password is really hidden. Furthermore, next to the form we added text "No Hint:" but the word "No" cannot be seen because it has the same color as the background. Under the text, we added a picture of vsFTPd server, which could lead the attackers to try and exploit the "vulnerable" 2.3.4 version of vsFTPd server, but we proved that this is not possible - at least using the publicly accessible and known exploits. However, we also hide the password on the webpage, which is saved in the alt of the gif image of Harry Potter at the top of the page. Here the goal is that the users use the web inspector tool or/and page source so that they can see that the form is fake, and eventually notice the password in the alt text.

Second vulnerability (to gain root access) is in the CRON job, which uses the script `job1.sh` located in the `/home/stud/jobs`. The vulnerability here is that the script has permission set to 777 which enables the stud user to execute arbitrary code as root user if the attacker changes the content of script (since script is executed by the root user). The cron job is scheduled to run everyday at 12 (noon) which can be seen using command `` `sudo crontab -l` -> `0 12 * * * /home/stud/jobs/job1.sh` ``.