Web Application Penetration Testing - Final Project

Paweł Kamiński

Global Red Extended Course

RM270822UW1

BLACK-BOX PENETRATION TEST

Scope:
The client, "Technician Blog", specified that the testing will occur only on a beta version of the website located in a container made by “Docker.” The client did not allow testing on the live site.

Out of scope:
The client specified that no external resource testing will be permitted, including JS codes that hold URLs that are not part of the domain.
Social Engineering was not included in the test scope.

Auditor:
Paweł Kamiński

Date of testing:
10.05 - 10.06.2023

Version:
1.0

Tools used:
Kali linux with tools, host-station

Introduction

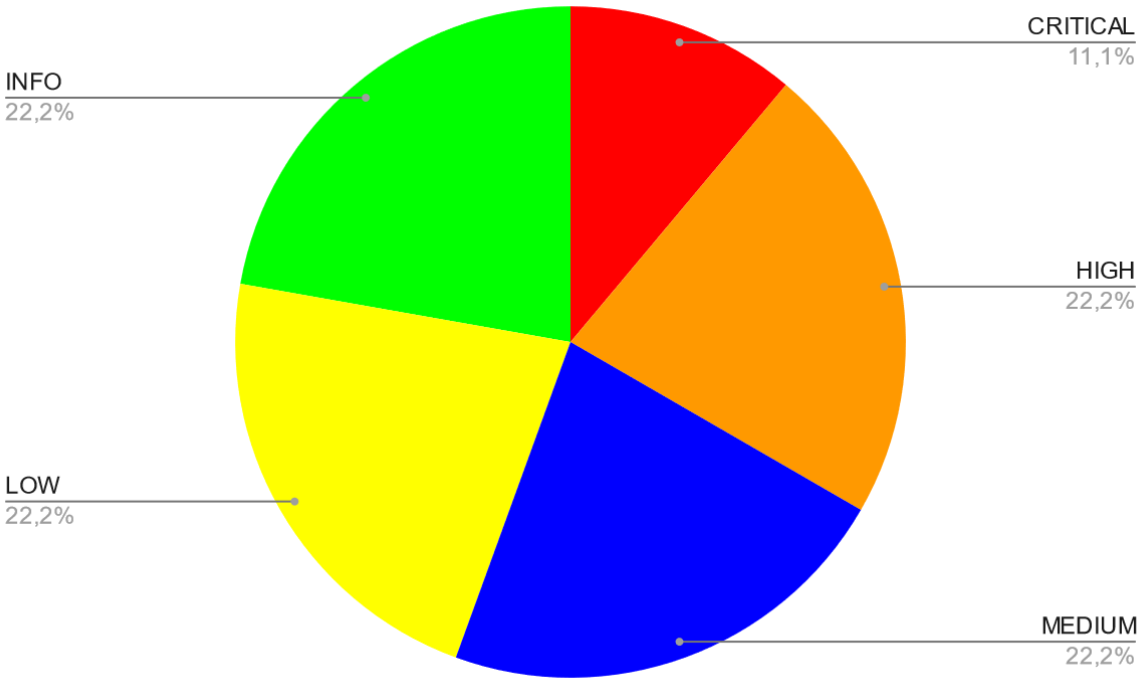
Black-box penetration testing is a comprehensive security assessment technique that aims to evaluate the vulnerabilities and weaknesses of a target. In this context, we will conduct a black-box penetration test on the TechNation website.
TechNation is a technology blog that provides up-to-date information on various technological advancements, news, and reviews. The website features a user-friendly interface with different sections, including news, reviews, deals, and an admin panel. It covers a wide range of topics, including the latest gadgets, inventions, and breakthroughs in the world of technology.

Vulnerability classification

Vulnerabilities have been classified on a five-point scale reflecting both the probability of finding a vulnerability, as well as the significance of the effects of its exploitation. Below is included a brief description of each materiality level

1. **Critical**: Critical vulnerabilities are the most severe and pose a significant threat to the security of the application. Exploiting these vulnerabilities can result in complete compromise of the system, unauthorized access, or significant data breaches.
2. **High**: High-severity vulnerabilities indicate serious security issues that can lead to unauthorized access, data leakage, or significant disruption of the application's functionality. While not as severe as critical vulnerabilities, they still require immediate attention.
3. **Medium**: Medium-severity vulnerabilities represent potential security weaknesses that could be exploited to gain unauthorized access or compromise sensitive data. Although not as critical, they still require prompt remediation to mitigate risks.
4. **Low**: Low-severity vulnerabilities are less critical but should not be ignored. While the impact of exploiting these vulnerabilities may be limited, they can still contribute to the overall security posture of the application and should be addressed in a timely manner.
5. **Info**: Informational vulnerabilities are not security risks in themselves but provide valuable information regarding the system's configuration, version information, or potential areas of weakness. They serve as recommendations or insights for improving security but may not require immediate action.

GENERAL HIGHLIGHTS



Critical Vulnerability:

1. Unrestricted File Upload: The application allows users to upload files without proper validation

High Vulnerability:

1. Outdated applications
2. Password Field Submitted Using GET Method

Medium Vulnerabilities:

1. Brute Force
2. Admin/user panel

Low Vulnerabilities:

1. Admin.php
2. No Certificates

Info Vulnerability:

1. ROOT folder
2. The absence of an .htaccess

[CRITICAL] **Unrestricted File Upload**

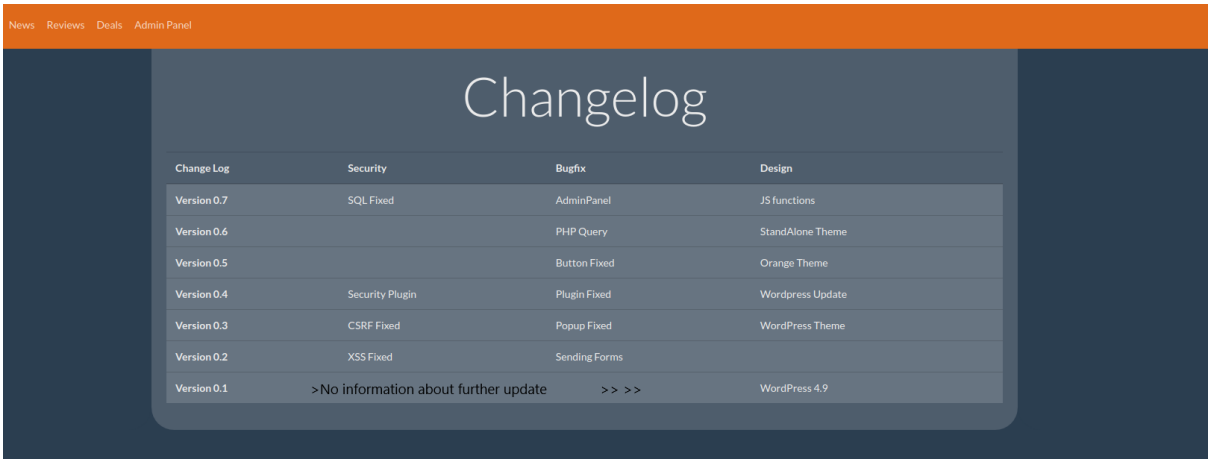
Unrestricted file upload can lead to several potential damages and security risks, including:

1. Malware or virus distribution: Attackers can upload malicious files, such as executable files (.exe) or script files (.js), which can contain malware or viruses. When these files are executed or accessed by other users, it can lead to the infection of systems or compromise the integrity of the server.
2. Remote code execution: If an attacker can upload executable files and gain the ability to execute them on the server, they may be able to run arbitrary commands or scripts, effectively taking control of the server. This can lead to unauthorized access, data theft, or even complete system compromise.
3. Defacement and data manipulation: Unrestricted file upload can allow attackers to overwrite or manipulate existing files on the server. This can result in website defacement, where the attacker replaces the legitimate content with their own malicious or inappropriate content.
4. Denial of Service (DoS): Attackers may upload large files or a large number of files to consume server resources, causing a denial of service condition. This can result in the server becoming unresponsive or inaccessible to legitimate users.
5. Privilege escalation: If an attacker can upload and execute files on the server, they may be able to exploit vulnerabilities in the server software or misconfigured permissions to escalate their privileges. This can enable them to gain administrative access to the server or other sensitive resources.
6. Data leakage or exposure: Unrestricted file upload can allow attackers to upload sensitive files or access restricted areas of the server. This can lead to the exposure or theft of confidential data, including personally identifiable information (PII), financial records, or intellectual property.

Recommendation:

1. Implement strict file validation: Validate the file types, file extensions, and content of uploaded files. Only allow specific file types that are necessary for your application, such as images, documents, or specific file formats. Use server-side validation and content inspection to verify that the uploaded file matches the expected file type.
2. Restrict file permissions: Set appropriate permissions on the server to prevent uploaded files from being executed or accessed directly. Ensure that uploaded files are stored outside the web root directory or in a restricted folder with limited permissions. This helps to prevent unauthorized execution of uploaded files.
3. Scan uploaded files for malware
4. Secure file storage: Store uploaded files in a secure manner, preferably outside the web root directory.
5. Regularly update and patch server software

[HIGH] OUTDATED APPLICATIONS CVE-2018-14028



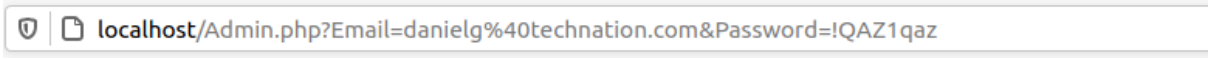
WordPress 4.9, being an older version, may have several known vulnerabilities:

1. Cross-Site Scripting (XSS): XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by users. This can lead to session hijacking, defacement, or the theft of sensitive information.
2. SQL Injection (SQLi): SQLi vulnerabilities enable attackers to manipulate the application's database by injecting malicious SQL code. This can lead to unauthorized access, data leakage, or even complete compromise of the database.
3. Cross-Site Request Forgery (CSRF): CSRF vulnerabilities allow attackers to trick users into performing unintended actions on a website without their knowledge or consent. This can lead to actions like changing passwords, making unauthorized transactions, or modifying settings.
4. Remote Code Execution (RCE): RCE vulnerabilities allow attackers to execute arbitrary code on the server hosting the WordPress application. This can result in complete control over the server, leading to unauthorized access, data breaches, or further compromise of the hosting environment.
5. Privilege Escalation
6. File Inclusion: File inclusion vulnerabilities can enable attackers to include and execute arbitrary files on the server.
7. Denial of Service (DoS)

Recommendation:

1. Backup your website: Before performing any updates, make sure to create a complete backup of your WordPress website.
2. Check for compatibility.
3. Update plugins and themes.
4. Upgrade WordPress: Once you have a backup and have checked for compatibility
5. Test your website.

[HIGH] Password field submitted using GET method



This site is using the GET method to submit the password field, it can introduce several security risks and vulnerabilities. Here are some potential consequences:

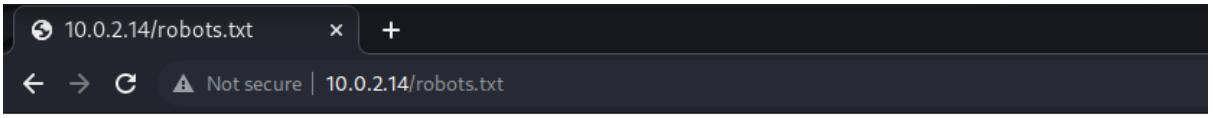
1. Exposure of Passwords: When the password is submitted via the GET method, it is appended to the URL as a query parameter. This means that the password is visible in the URL itself.
2. Caching and Browser History: Browsers may cache URLs that are accessed via the GET method. As a result, the password may be stored in the browser's cache or history, even after the user logs out or closes the browser.
3. Referer Headers: When submitting a form using the GET method, the password can be included in the Referer header. This can potentially expose the password to external websites, especially if the user clicks on links or advertisements on those external sites.
4. Proxy Servers and Network Sniffing: When the password is transmitted via the GET method, it is sent in plaintext as part of the URL. Proxy servers or network sniffing tools can capture and log the URLs, exposing the passwords in transit.

Recommendations:

To address these security risks, it is highly recommended to use the **POST** method instead of the GET method when submitting sensitive information like passwords. The **POST** method sends the data in the request body instead of appending it to the URL, providing better security and confidentiality. Additionally, using secure connections (**HTTPS**) ensures that the data transmitted between the user and the server is encrypted, further protecting sensitive information like passwords.

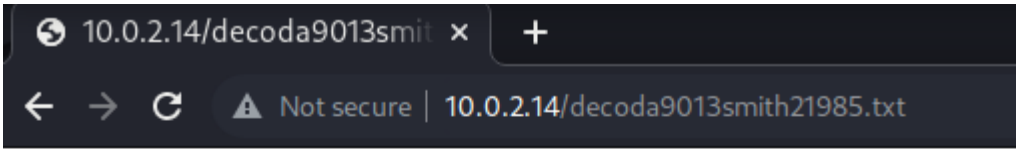
[MEDIUM] THIS SITE IS VULNERABLE TO BRUTE FORCE ATTACK:

If we go to /robots.txt, we see this:




```
User-agent: *
Disallow: /decoda9013smith21985.txt
```

And if we follow, we will get a set of strings:

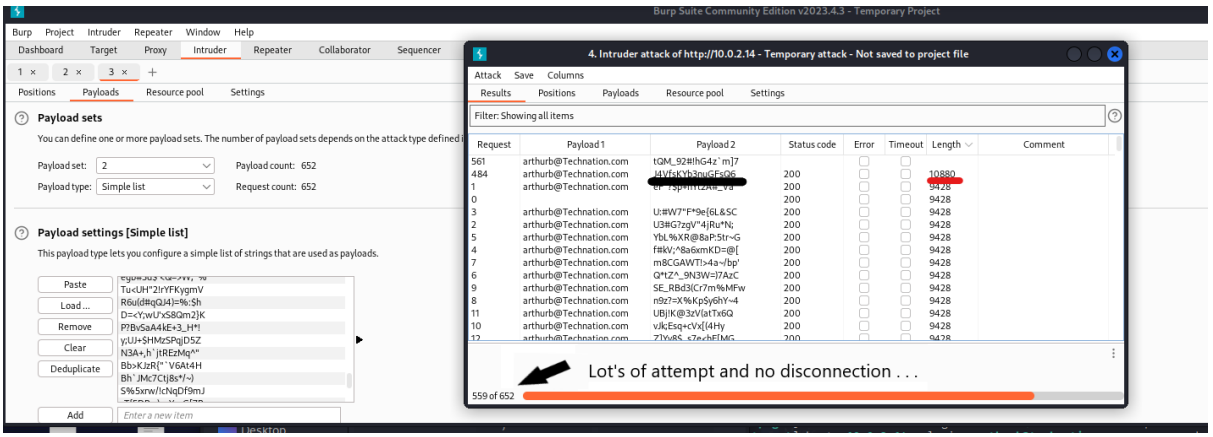



```
Password Generator Prototype DB:

eF"?$p+nYt2A#_Va
U3#G?zgV"4jRu*N;
U:#W7"F*9e{6L&SC
f#kV;^8a6xmKD=@[
YbL%XR@8aP:5tr~G
Q*tZ^_9N3W=)7AzC
m8CGAWT!>4a~/bp'
n9z?=X%Kp$y6hY~4
SE_RBd3(Cr7m%MFw
vJk;Esq+cVx[(4Hy
UBj!K@3zV(atTx6Q
Z]Yy8$_s7e<hE[MG
D9$GsdH[nvB+_p8C
G'd}7BD5Rs(Z-Eh6
u7{x"s*kY&JT`$8y
aeJv,dx4&)PjhX'K
WdSs[6hw9D-kyX/?
yQ5<f$9N}(2RJD]x
f8Z}*;"4@$KzPNxg
N=UKPuFh5@H>L'fy
xMnTa7@pre.%<PB=
JBqa&2xXh;SE.AQ3
p[K8-B]uk{gM=$3j
H}d,;Qn4aL+z]wJv
U_GS{%Wzyx6&w7b]
j/E@!9m=L^d,7b~.
U%)w4ft?,*SLju>8
b-Y}:6M&GWS'JLjd
```

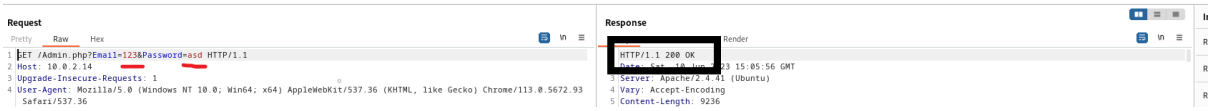
And there is lot more

This set of strings which contains password for one of the users, and because there “max login attempt” on server we can brute force site, but only burp show us the right one by the length of connection:

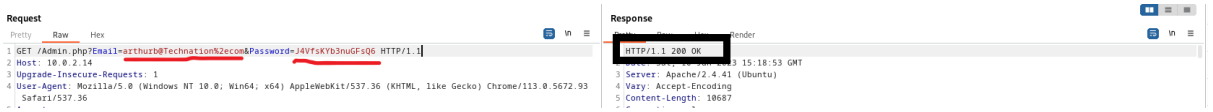


And we have password for user arthurb@technation.com

If we use Burp Suite to check status connection of wrong credentials:



And with the right ones:



We're still getting status: **200 OK**

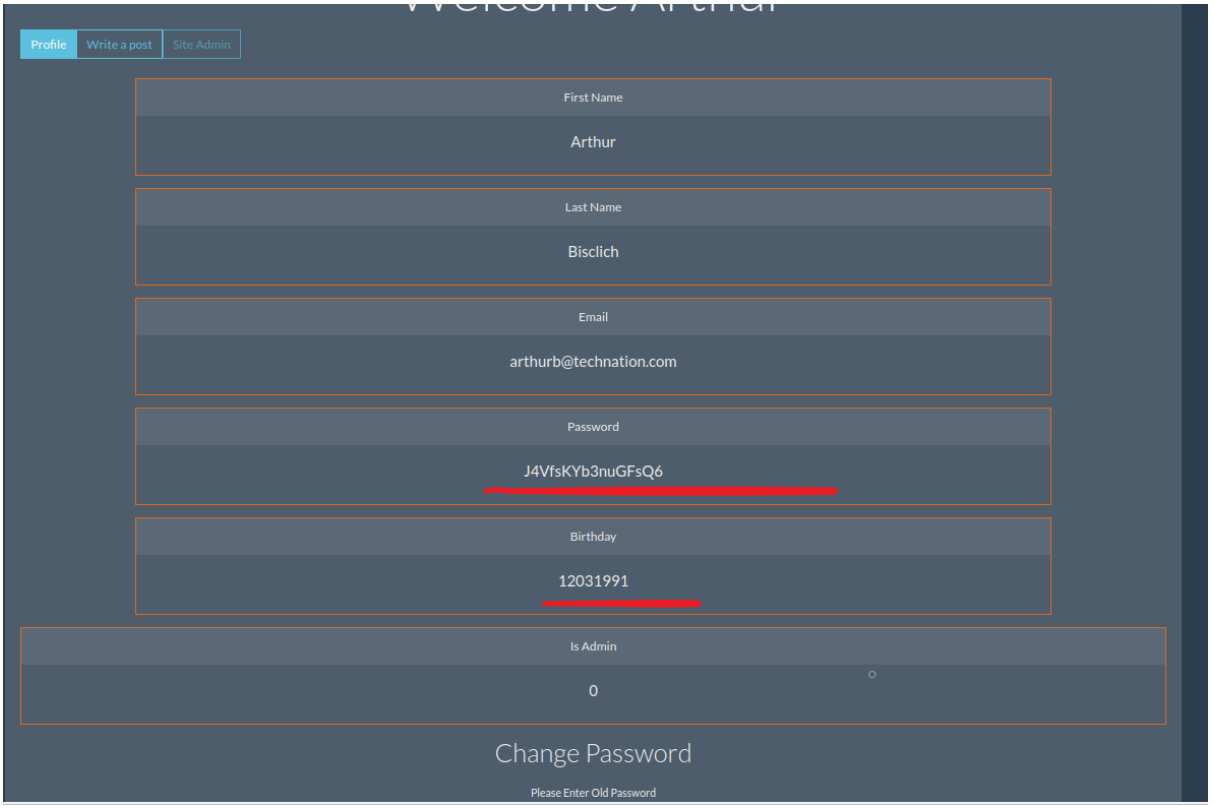
That is why a tool like hydra getting confused:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ hydra -L user -P pass http-get://10.0.2.14/Admin.php
Hydra v9.4 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service orgar
ons, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-06-10 11:04:59
[DATA] max 16 tasks per 1 server, overall 16 tasks, 144 login tries (l:12/p:12), ~9 tries per task
[DATA] attacking http-get://10.0.2.14:80/Admin.php
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: LB&MG$sm2zkLz57k
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: J4VfsKYb3nuGFsQ6
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: VaFG4tbqF9#HB8gD
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: UmZd2JMaCTsr78*8
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: 3!Hmr3FumDtuzFP5
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: 2Lb&Cpu=5$VUppM&
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: U!CPJmWr22yAEqu6
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: 39ZV?bCmk@LN88$d
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: ?aL4KjX%4bEyaRnF
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: xyaEtCqGQN8_D*7s
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: %Z!Z7tJB5Nf+EF?$
[80][http-get] host: 10.0.2.14   login: danielg@technation.com   password: J4VfsKYb3nuGFsQ6
[80][http-get] host: 10.0.2.14   login: arthurb@technation.com   password: 6X?%b*wk6hkbh4Ts
[80][http-get] host: 10.0.2.14   login: danielg@technation.com   password: UmZd2JMaCTsr78*8
[80][http-get] host: 10.0.2.14   login: danielg@technation.com   password: xyaEtCqGQN8_D*7s
[80][http-get] host: 10.0.2.14   login: danielg@technation.com   password: LB&MG$sm2zkLz57k
[80][http-get] host: 10.0.2.14   login: danielg@technation.com   password: VaFG4tbqF9#HB8gD
```

Kind of a bug as a feature :)

[MEDIUM] ADMIN PANEL



1. **There is no policy for new password for user, even 1 sign was acceptable**
2. There is sensitive information showed on user panel like Date of birth or Password

Recommendations:

1. Set Password complexity: Require users to create passwords that meet specific complexity criteria. For example, passwords should be a minimum length, including a combination of uppercase and lowercase letters, numbers, and special characters.
2. Password expiration: Set a policy that requires users to change their passwords periodically.
3. Password history: Prevent users from reusing their previous passwords
4. **Account lockout: Implement an account lockout policy that temporarily locks user accounts after a certain number of failed login attempts. This helps protect against brute-force attacks and unauthorized access attempts.**
5. Two-factor authentication (2FA): Consider implementing two-factor authentication

[LOW] INVESTIGATING Admin.php

There is information that can help potential hackers like:

1. We can see in the code that there is passchangedocument.php and even more, I was able to run it manually, but it redirects me straight to index.php
2. Lack of input validation
3. Missing authentication and authoritarian

Recommendations:

1. Keep sensitive files, including those involved in password-related operations, outside the publicly accessible web root directory
2. Implement proper access controls and authentication mechanisms to restrict access to sensitive files and functionality only to authorized users.
3. Implementing input validation

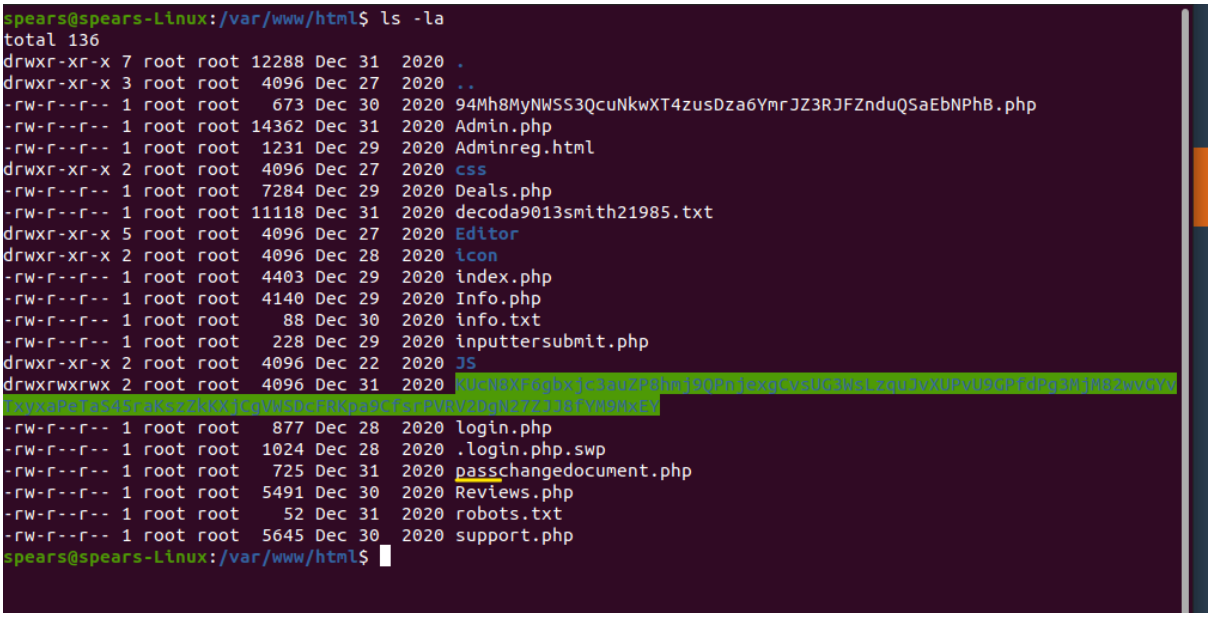
[LOW] No Certificates:

This website does not have a certificate, it typically indicates a vulnerability related to the lack of secure communication. Certificates, specifically SSL/TLS certificates, are used to establish secure encrypted connections between a website and its users. The absence of a certificate means that the website is not utilizing HTTPS (HTTP over SSL/TLS) for secure communication.

Without HTTPS, the data transmitted between the user's browser and the website can be intercepted and potentially manipulated by attackers. This vulnerability exposes users to various risks, such as eavesdropping, data theft, and impersonation attacks.

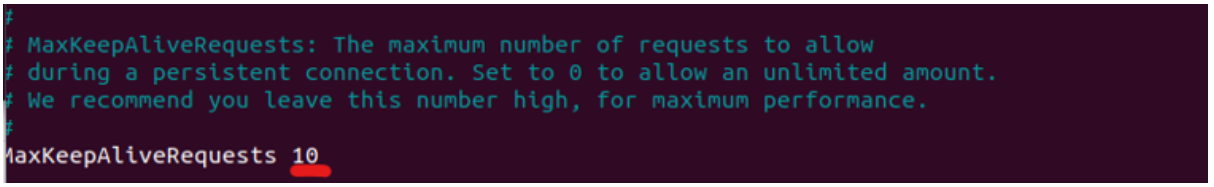
[INFO] ROOT FOLDER

I put it here because formally we do not have access to this file from the client's side, but from the host's side we do and if we check var/www/html there is few interesting files like i mentioned before “passchangedocument.php”, and also there is no .htaccess file which can cause lot of trouble:



Recommendations:

1. Keep sensitive files outside root directory
2. Set MaxKeepAliveRequest to 10



[INFO]The absence of an .htaccess

The absence of an .htaccess file on a server can have several implications and potential consequences. Here are a few examples:

1. Security vulnerabilities: The **.htaccess** file allows you to define access restrictions and security settings for your website. Without it, your server may be more vulnerable to attacks or unauthorized access, as you won't have fine-grained control over permissions, password protection, or IP blocking.
2. URL manipulation: .htaccess files are often used to configure URL rewriting, which allows you to create user-friendly and search engine-friendly URLs. Without this functionality, your website's URLs may be less readable, impacting user experience and SEO.
3. Error handling: .htaccess files can define custom error pages for different HTTP status codes, such as 404 (Not Found) or 500 (Internal Server Error). Without an .htaccess file, your server will use default error pages, which may not provide helpful information to users.
4. Performance optimization: .htaccess files enable various caching and compression techniques that can improve website performance by reducing load times. Without these optimizations, your website may be slower and consume more server resources.
5. Directory indexing: An **.htaccess** file can prevent directory listing, which means that if it's missing, anyone can access and view the contents of your directories. This can lead to the exposure of sensitive files or directories that should be restricted.

It's important to note that some servers may have default configurations that mitigate some of these issues. However, not having an **.htaccess** file means you lose the ability to customize and enhance your server's behavior in these areas.

SUMMARY:

The application under review has several vulnerabilities but implementing recommended security measures, such as securing file uploads, handling passwords securely, enforcing input validation, updating WordPress, will significantly improve its security and it can be made secure.