



## Security Test Report

SUBJECT OF WORK:

Penetration tests of the Technician Blog's official website.

WORK EXECUTION DATES

12.02.2024 – 23.02.2024

WORK LOCATION

Poland

AUDITOR

TrickySnowStorm

## Spis treści

Introduction.....	3
Summary and Recommendations .....	3
Key Findings .....	6
Detailed Findings.....	14
Conclusion and Security Improvement Suggestion.....	29

## Introduction

This report is a summary of the security pentest of Technician Blog's official website conducted by Monika Hamala, HackerU student. The subject of the examination was security test of the website <http://10.0.2.19:80> (blackbox). The report lists vulnerabilities, documents the process of their discovery, provides descriptions, and indicates the method of remediation for each.

During the audit few critical or high vulnerabilities were found. Particular emphasis was placed on vulnerabilities that could or may have a negative impact on the confidentiality, integrity, and availability of processed data. The aim of the tests was not the analysis of privacy aspects, but rather the detection of potential security vulnerabilities. Furthermore, in this tests, the analysis of the source code of the application and external resources were not included.

Security tests were conducted in accordance with commonly accepted methodologies for testing web applications. As part of the audit, a combination of manual testing based on the methodologies mentioned above and support from a variety of automated tools was utilized. These tools include Burp Suite Professional and nmap.

Vulnerabilities have been detailed described in the subsequent sections of the report.

## Summary and Recommendations

**The assessment** of vulnerability classification usually relies on several factors such as:

Potential consequences: Can the exploitation of the vulnerability lead to serious consequences, such as loss of confidential data, compromise of user accounts, or the execution of malicious software?

Ease of exploitation: How easily can attackers exploit the vulnerability? Does it require effort and specialized knowledge, or can it be exploited using simple tools and techniques?

Scope of potential impact: Does the vulnerability affect a wide range of functionalities or data on the website, or is it more limited in scope?

Availability of exploits: Are there publicly available exploits for the vulnerability, suggesting that it is well-known in hacker and attacker communities?

In the report below, vulnerabilities of the following severities have appeared:

High Severity: This means that the vulnerability has the potential for a significant impact on the

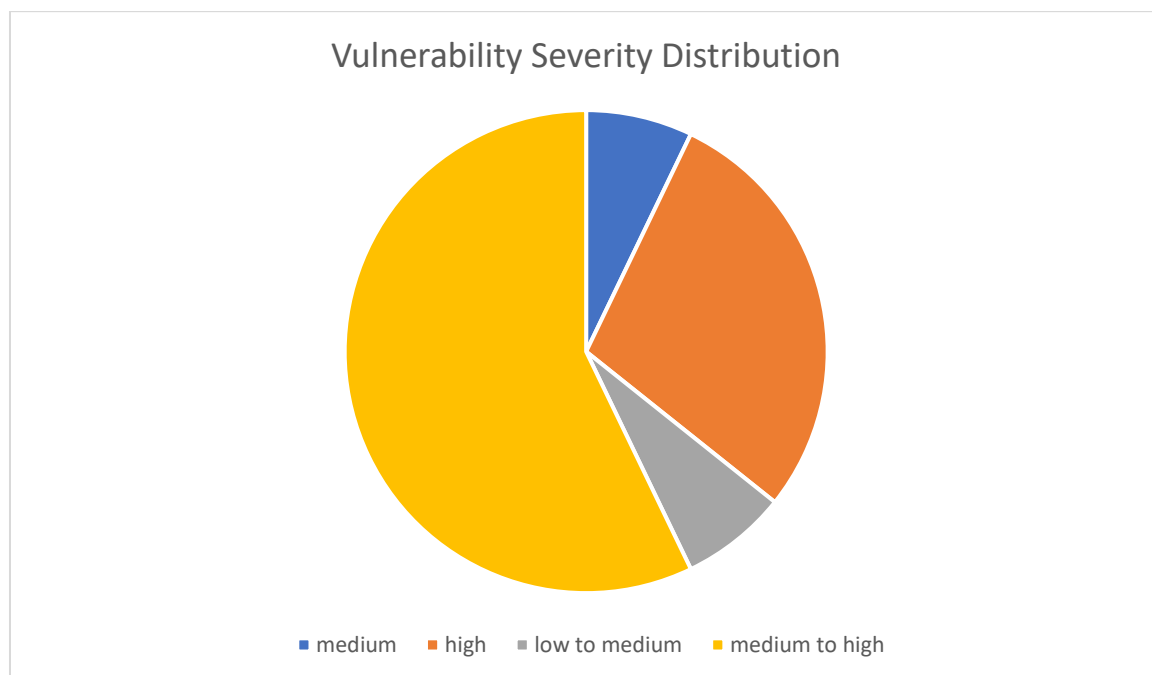
security of the website or application. Attackers can easily exploit this vulnerability, leading to the loss of sensitive data, compromise of user accounts, execution of malicious software, or other serious consequences. It requires immediate action to fix and secure the website.

Medium to High Severity: Indicates that the vulnerability is more complex than those classified as medium but still poses a serious risk to security. It may be more difficult for attackers to exploit or have a more limited scope, but it still requires attention and immediate action.

Medium Severity: A vulnerability that has a moderate impact on security. It may lead to undesirable consequences but is less likely to cause serious harm. It still requires attention and should be fixed, but it's not as urgent.

Low to Medium Severity: This means that the vulnerability has a lower impact on security but still requires attention and fixing. It may be less complex or have a more limited scope than vulnerabilities classified as medium, but it should not be disregarded.

Vulnerability classification depends on assessing various factors such as potential consequences, ease of exploitation, scope of impact, and availability of exploits. It's important to evaluate each vulnerability individually and take appropriate actions to fix and secure the system.



In my professional assessment, **the level of security of examined website is low**. The site is vulnerable to XSS, Brute Force in Admin Panel, CSFR, SQLi, Path Traversal, LFI, Information Disclosure, Parameter Tampering, Unrestricted File Upload, IDOR and some more. Exploiting most of those would require medium level of technical cybersecurity knowledge.

Vulnerability	Status	Severity
Cross Site Scripting	active	high
SQL Injection	active	high
Unrestricted File Upload	active	high
Credential Exposure via URL	active	high
Brute Force Vulnerability in Admin Panel	active	medium to high
Cross-site request forgery	active	medium to high
Path Traversal	active	medium to high
Local File Inclusion	active	medium to high
Insecure Direct Object References	active	medium to high
Directory Listing	active	medium to high
HTTP Verb Tampering	active	medium to high
Server Version Disclosure	active	medium to high
Parameter Tampering	active	medium
Information Disclosure	active	low to medium

Based on the results of the conducted test, I **recommend implementing** several key steps to enhance the security of the website:

1. **Regular Updates and Patching:** Ensure that all software, frameworks, and scripts used on the website are regularly updated to remove known vulnerabilities and protect against new threats.
2. **Data Filtering and Validation:** Implement mechanisms for filtering and validating input data to prevent XSS (Cross-Site Scripting), SQLi (SQL Injection), and other vulnerabilities that can exploit improper input data.
3. **Strong Passwords and Authentication:** Deploy a policy of strong passwords and secure authentication to limit the risk of Brute Force attacks on the admin panel and IDOR (Insecure Direct Object References).
4. **Access Control:** Ensure that users have access only to necessary functions and resources to limit the risk of CSFR (Cross-Site Request Forgery), Path Traversal, and LFI (Local File Inclusion) attacks.
5. **Monitoring and Logging:** Implement monitoring and logging mechanisms for user actions and system events to quickly detect and respond to attacks and incidents related to information disclosure.
6. **External Audits and Penetration Testing:** Regularly conduct independent security audits and penetration tests to identify and address any vulnerabilities before they are exploited by potential attackers.

Please remember, website security is an ongoing process that requires attention and commitment at all stages of its development and operation.

## Key Findings

Describe vulnerability, explain it in general terms, provide a risk score, severity, probability and fix effort level. Can use OWASP top 10.

**Cross Site Scripting** also called XSS is a vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. These scripts can execute in the context of a victim's browser, potentially hijacking user sessions, defacing websites, or stealing sensitive data. Imagine a website that allows users to input comments. An attacker can input malicious code (such as JavaScript) into the comment field. When other users view the comment section, this code executes in their browsers, allowing the attacker to perform actions on behalf of the user, such as stealing cookies or redirecting to a malicious website.

**Risk Score** of this vulnerability is high. XSS can lead to significant data breaches, unauthorized access, and other security compromises.

**Severity:** High. XSS vulnerabilities can have severe consequences for both users and the organization hosting the vulnerable website.

**Probability:** Common. XSS vulnerabilities are prevalent in web applications due to insufficient input validation and output encoding.

**Fix Effort Level:** Medium to High. Fixing XSS vulnerabilities typically involves implementing proper input validation and output encoding techniques throughout the web application, which can require significant development effort and testing.

**SQL Injection** (SQLi) is a type of attack where an attacker injects malicious SQL code into input fields or parameters of a web application, exploiting vulnerabilities in the application's code that fail to properly sanitize or validate user inputs. Imagine a web application that accepts user input to perform database queries. An attacker can input specially crafted SQL commands into these input fields to manipulate the database or execute unauthorized operations. For example, by inputting "1' OR '1'='1" into a login form, an attacker can bypass authentication checks and gain unauthorized access to the system.

**Risk Score:** High. SQL Injection attacks can lead to unauthorized access to sensitive data, data manipulation, or even full database compromise, posing significant risks to the security and integrity of the application and its data.

**Severity:** High. The severity of SQL Injection vulnerabilities depends on the sensitivity of the data accessible via the vulnerable application and the potential impact of data breaches or manipulation.

**Probability:** Common. SQL Injection vulnerabilities are widespread and can be found in many web applications, especially those with improper input validation or insufficient security measures.

**Fix Effort Level:** High. Fixing SQL Injection vulnerabilities requires implementing robust input validation, parameterized queries, and other security measures to prevent malicious SQL injection attacks. It may involve significant code changes and thorough testing to ensure the vulnerability is properly mitigated.

Addressing SQL Injection vulnerabilities is critical to protecting the confidentiality, integrity, and availability of data stored and processed by web applications. Failure to mitigate SQL Injection vulnerabilities can expose sensitive information and lead to severe security breaches.

**Unrestricted File Upload** is a type of security vulnerability that occurs when a web application allows users to upload files without proper validation and controls, enabling attackers to upload malicious files to the server. Unrestricted File Upload vulnerabilities arise when a web application accepts file uploads from users without verifying the file type, content, or size, and without enforcing proper access controls on the uploaded files. In a web application vulnerable to Unrestricted File Upload, attackers can upload malicious files, such as scripts or malware, to the server. These files can then be executed or accessed by other users, leading to various security risks, including remote code execution, data breaches, or server compromise.

**Risk Score:** High. Unrestricted File Upload vulnerabilities can lead to severe security breaches, allowing attackers to execute arbitrary code on the server, steal sensitive data, or compromise the integrity and availability of the application and its resources.

**Severity:** High. The severity of Unrestricted File Upload vulnerabilities depends on the capabilities of the uploaded files and the potential impact of their execution or access on the security and functionality of the application.

**Probability:** Common. Unrestricted File Upload vulnerabilities are relatively common, especially in web applications that allow user-generated content or file uploads without proper validation and access controls.

**Fix Effort Level:** High. Fixing Unrestricted File Upload vulnerabilities typically involves implementing strict file upload validation, content type verification, file size restrictions, and proper access controls to prevent unauthorized file uploads and mitigate the risk of malicious file execution.

**Credential Exposure via URL** this vulnerability involves sending sensitive data, such as usernames and passwords, via the GET method in the URL. The login credentials are visible in

the browser's address bar, leading to potential exposure of this information. In this vulnerability, authentication data, such as usernames and passwords, are transmitted in the URL, making them visible to anyone who has access to the browser or network request logs. This poses a significant risk as login credentials can be intercepted by third parties and used for unauthorized access to user accounts.

**Risk Score:** High. Transmitting sensitive data in the URL exposes it to unintended exposure, potentially leading to compromise of user accounts and privacy breaches.

**Severity:** High. The visibility of login credentials in the URL poses a serious security threat as it facilitates attackers in intercepting and exploiting this information for unauthorized access.

**Probability:** Medium to High. This vulnerability is commonly found in web applications, especially those that have not implemented proper security practices for transmitting authentication data.

**Fix Effort Level:** Low to Medium. Fixing this vulnerability involves changing the way login credentials are transmitted, such as using the POST method instead of GET, and implementing encryption and security measures to protect login data from unauthorized access.

One of the most effective ways to address this vulnerability is to ensure that login credentials are transmitted using the POST method rather than GET, and to use the HTTPS protocol to encrypt communication to protect data during transmission. Additionally, it's important to implement strong authentication and authorization mechanisms to prevent unauthorized access to user accounts.

**Brute Force in Admin Panel** is an attack trial-and-error method used by attackers to gain unauthorized access to a system, such as an admin panel, by systematically trying various password combinations until the correct one is found. In the context of an Admin Panel, attackers attempt to log in using different username and password combinations repeatedly until they gain access. This method relies on the assumption that passwords are weak or predictable.

**Risk Score:** Medium to High. Brute Force attacks can potentially lead to unauthorized access to sensitive administrative functionalities, allowing attackers to manipulate or extract confidential data.

**Severity:** Medium to High. Depending on the level of access granted to the admin panel, the severity of this vulnerability can vary. If the admin panel controls critical functionalities or data, the severity would be higher.

**Probability:** Common. Brute Force attacks are common and relatively easy to execute, especially if proper security measures such as account lockout policies or strong password requirements are not implemented.



**Fix Effort Level:** Medium. Mitigating this vulnerability typically involves implementing security measures such as account lockout policies, CAPTCHA verification, or multi-factor authentication to prevent or limit the effectiveness of brute force attacks.

**Cross-site request forgery** is a type of attack where a malicious website tricks a user's web browser into executing unwanted actions on another website where the user is authenticated. An attacker crafts a malicious URL or form submission that, when clicked or submitted by a victim who is authenticated on another site, causes unintended actions to occur on the target site. These actions could include changing account settings, making transactions, or performing other sensitive operations without the victim's knowledge.

**Risk Score:** Medium to High. CSRF attacks can result in unauthorized actions being performed on behalf of authenticated users, leading to potential data breaches, financial losses, or other security compromises.

**Severity:** Medium to High. The severity depends on the actions that can be performed via the CSRF attack and the level of access granted to the authenticated user.

**Probability:** Common. CSRF attacks are prevalent, especially on websites that lack proper CSRF protection mechanisms.

**Fix Effort Level:** Medium. Mitigating CSRF vulnerabilities typically involves implementing mechanisms such as anti-CSRF tokens, same-site cookie attributes, or referrer checks to validate the origin of requests and prevent unauthorized actions.

**Path Traversal**, also known as Directory Traversal, is a type of security vulnerability that allows attackers to access files or directories outside of the web application's root directory. Path Traversal occurs when an attacker manipulates input parameters used by a web application to access files or directories that are not intended to be accessible. This can lead to unauthorized access to sensitive files, including configuration files, database files, or other critical system files. In a web application vulnerable to Path Traversal, attackers can manipulate file paths or directory names in URLs or input fields to navigate to directories or access files outside of the web application's intended scope. For example, by inputting "../" or "../../../" sequences into a file path parameter, an attacker can traverse up the directory tree and access files or directories located in higher-level directories.

**Risk Score:** Medium to High. Path Traversal vulnerabilities can result in unauthorized access to sensitive data or system files, potentially leading to data breaches, system compromise, or other security incidents.

**Severity:** Medium to High. The severity of Path Traversal vulnerabilities depends on the sensitivity of the files or directories accessible via the traversal and the potential impact of unauthorized access to such resources.

**Probability:** Common. Path Traversal vulnerabilities are relatively common, especially in web applications that handle file uploads, file downloads, or file system operations without proper input validation or access controls.

**Fix Effort Level:** Medium. Fixing Path Traversal vulnerabilities typically involves implementing proper input validation and access controls to restrict access to files and directories based on the application's intended functionality. This may require modifying file path handling logic and implementing security measures to prevent unauthorized directory traversal.

Addressing Path Traversal vulnerabilities is essential to prevent unauthorized access to sensitive data and protect the confidentiality and integrity of web applications and their resources. Failure to mitigate Path Traversal vulnerabilities can lead to severe security breaches and compromise the security of the entire system.

**Local File Inclusion (LFI)** is a type of security vulnerability that allows attackers to include files on a server through the web browser. LFI occurs when a web application allows an attacker to include files residing on the server's file system. This vulnerability can be exploited to disclose sensitive information, execute arbitrary code, or gain unauthorized access to system files. In a web application vulnerable to LFI, attackers can manipulate input parameters or URLs to include files stored on the server. This can allow them to read sensitive files, such as configuration files or user data, execute malicious scripts, or gain unauthorized access to system resources.

**Risk Score:** Medium to High. LFI vulnerabilities can lead to unauthorized access to sensitive data, remote code execution, or server compromise, depending on the files accessible and the capabilities of the attacker.

**Severity:** Medium to High. The severity of LFI vulnerabilities depends on the sensitivity of the files accessible via the inclusion and the potential impact of unauthorized access to such resources.

**Probability:** Common. LFI vulnerabilities are relatively common, especially in web applications that handle file inclusion without proper input validation or access controls.

**Fix Effort Level:** Medium. Fixing LFI vulnerabilities typically involves implementing proper input validation, access controls, and file system protections to prevent unauthorized file inclusion and restrict access to sensitive files and directories.

Addressing LFI vulnerabilities is crucial to prevent unauthorized access to sensitive data and protect the confidentiality and integrity of web applications and their resources. Failure to mitigate LFI vulnerabilities can lead to severe security breaches and compromise the security of the entire system.

**Insecure Direct Object References (IDOR)** is a type of security vulnerability that occurs when an application exposes internal implementation objects, such as files, directories, or database

records, to users without proper authorization checks. IDOR vulnerabilities arise when an application fails to properly validate user input or enforce access controls, allowing attackers to directly access and manipulate sensitive objects or resources. In a web application vulnerable to IDOR, attackers can manipulate object references, such as URLs or parameters, to access unauthorized resources or perform actions that should be restricted. For example, an attacker may modify a URL parameter to access another user's private data or administrative functionalities.

**Risk Score:** Medium to High. IDOR vulnerabilities can lead to unauthorized access to sensitive data, unauthorized actions, or privilege escalation, depending on the resources accessible and the actions performed by the attacker.

**Severity:** Medium to High. The severity of IDOR vulnerabilities depends on the sensitivity of the accessed resources and the potential impact of their unauthorized access or manipulation on the security and integrity of the application.

**Probability:** Common. IDOR vulnerabilities are relatively common, especially in applications that lack proper authorization checks and access controls on sensitive resources.

**Fix Effort Level:** Medium. Fixing IDOR vulnerabilities typically involves implementing proper access controls, authorization checks, and indirect object references to prevent unauthorized access or manipulation of sensitive resources.

Addressing IDOR vulnerabilities is crucial to protect the confidentiality, integrity, and availability of sensitive data and resources within web applications. Failure to mitigate IDOR vulnerabilities can lead to data breaches, unauthorized actions, or other security incidents with serious consequences.

**Directory Listing** is a vulnerability that occurs when a web server is configured to display the contents of a directory when there is no default index file (e.g., index.html) present in that directory. As a result, when a user accesses the directory via a web browser, the server lists all files and subdirectories within that directory, potentially exposing sensitive information. Imagine a scenario where a web server hosts a directory containing sensitive files, such as configuration files, user data, or application code. If Directory Listing is enabled and there is no index file present in the directory, anyone who accesses the directory via a web browser can view a list of all files and subdirectories within it. This exposes the directory's contents to unauthorized users, potentially leading to data breaches or unauthorized access to sensitive information.

**Risk Score:** Medium to High. Directory Listing can pose a significant risk, depending on the sensitivity of the files and directories exposed. If the directory contains confidential or proprietary information, the risk score would be higher.

**Severity:** Medium to High. The severity of Directory Listing depends on the nature of the exposed files and the potential impact of their disclosure. If the exposed files contain sensitive data or critical system information, the severity would be higher.

**Probability:** Common. Directory Listing vulnerabilities are relatively common, especially in web servers with default configurations or misconfigurations that enable Directory Listing.

**Fix Effort Level:** Low to Medium. Fixing Directory Listing vulnerabilities typically involves configuring the web server to disable Directory Listing and ensuring that default index files (e.g., index.html) are present in all directories served by the web server. This can usually be accomplished through server configuration changes or by creating index files where needed.

Addressing Directory Listing vulnerabilities is essential to protect the confidentiality and integrity of sensitive data hosted on web servers. Failure to mitigate this vulnerability can lead to unauthorized access to sensitive information, data breaches, or other security incidents.

**HTTP Verb Tampering** refers to the manipulation of HTTP methods used in web requests. Attackers exploit this vulnerability by modifying the HTTP methods (such as GET, POST, PUT, DELETE) to induce unexpected behaviours or bypass security controls in web applications. HTTP Verb Tampering allows attackers to modify the HTTP methods of requests sent to a web server. By altering the method used in a request, attackers can attempt to perform unauthorized actions, bypass access controls, or exploit vulnerabilities in the application logic. For example, an attacker may change a POST request to a GET request to retrieve sensitive information that should only be accessible through POST requests.

**Risk Score:** Medium to High. HTTP Verb Tampering can lead to various security risks, including unauthorized access, data manipulation, and bypassing of security controls, depending on the actions performed by attackers and the vulnerabilities present in the web application.

**Severity:** Medium to High. The severity of HTTP Verb Tampering depends on the impact of the manipulated HTTP methods on the security and functionality of the web application. If successful, attackers can potentially compromise the integrity, confidentiality, and availability of data and resources.

**Probability:** Medium to Low. The probability of exploiting HTTP Verb Tampering depends on the effectiveness of existing security controls, the complexity of the web application, and the knowledge and motivations of potential attackers.

**Fix Effort Level:** Medium. Mitigating HTTP Verb Tampering typically involves implementing proper access controls, input validation, and secure coding practices to ensure that the web application properly handles and validates HTTP methods used in requests. Additionally, security testing and code reviews can help identify and remediate vulnerabilities related to HTTP Verb Tampering.

Overall, addressing HTTP Verb Tampering vulnerabilities requires careful consideration of access controls and input validation mechanisms to prevent unauthorized actions and protect the security of web applications.

**Server Version Disclosure** involves displaying the server's version in the response to an incorrect request. Attackers can analyse this server version to find relevant exploits or attacks that can be carried out on that specific server version. When the server displays its version in response to an erroneous request, attackers can leverage this information to find known security vulnerabilities or exploits that can be used to launch attacks on the server.

**Risk Score:** Medium to High. Finding the server version may facilitate attackers in exploiting known security vulnerabilities, potentially leading to server compromise or data privacy breaches.

**Severity:** Medium to High. Server version visibility may make it easier for attackers to launch attacks targeting specific weaknesses associated with that server version.

**Probability:** Medium to Low. This vulnerability occurs when the server reveals its version in response to an erroneous request, which may not always happen with all servers and configurations.

**Fix Effort Level:** Low. Fixing this vulnerability involves configuring the server not to disclose its version in response to an erroneous request. This can be done by adjusting server settings or implementing appropriate security overlays.

The solution to this vulnerability is to configure the server not to disclose its version in response to an erroneous request. This is a relatively simple fix that can significantly reduce the risk of attack.

**Parameter Tampering** is a type of security vulnerability that occurs when an attacker modifies parameters used by a web application to manipulate its behaviour or gain unauthorized access to resources. Parameter Tampering vulnerabilities arise when a web application relies solely on client-side input validation and fails to adequately validate or secure parameters sent from the client to the server. In a web application vulnerable to Parameter Tampering, attackers can manipulate parameters in requests sent to the server, such as URL parameters, form fields, or cookies, to modify application logic, bypass security checks, or gain unauthorized access to sensitive resources.

**Risk Score:** Medium. Parameter Tampering vulnerabilities can lead to unauthorized access to sensitive data, manipulation of application logic, or other security breaches, depending on the parameters manipulated and the actions performed by the attacker.

**Severity:** Medium. The severity of Parameter Tampering vulnerabilities depends on the impact of the tampered parameters on the application's functionality and the potential consequences of unauthorized access or manipulation.

**Probability:** Common. Parameter Tampering vulnerabilities are relatively common, especially in web applications that rely solely on client-side input validation and lack proper server-side validation and access controls.

**Fix Effort Level:** Medium. Fixing Parameter Tampering vulnerabilities typically involves implementing proper server-side validation and access controls, securing sensitive resources, and ensuring that client-side input is not trusted blindly by the server.

Addressing Parameter Tampering vulnerabilities is essential to protect the integrity and security of web applications and prevent unauthorized access or manipulation of sensitive data and resources. Failure to mitigate Parameter Tampering vulnerabilities can lead to data breaches, application vulnerabilities, and other security incidents.

**Information Disclosure** vulnerabilities arise when a system provides access to sensitive data, such as user credentials, system configuration details, or other confidential information, to individuals who should not have access to it. In a web application vulnerable to Information Disclosure, sensitive information may be exposed through error messages, debug output, or improperly secured APIs. Attackers can exploit these vulnerabilities to gather intelligence about the system's configuration, user accounts, or other sensitive data.

**Risk Score:** Low to Medium. The risk associated with Information Disclosure vulnerabilities depends on the sensitivity of the data exposed and the potential impact of its disclosure on the organization or its users.

**Severity:** Low to Medium. The severity of Information Disclosure vulnerabilities depends on the nature and volume of sensitive information exposed and the potential consequences of its disclosure.

**Probability:** Common. Information Disclosure vulnerabilities are relatively common, especially in web applications and APIs that fail to properly handle and secure sensitive data.

**Fix Effort Level:** Low to Medium. Fixing Information Disclosure vulnerabilities typically involves identifying and securing sensitive data, implementing proper error handling mechanisms, and restricting access to confidential information based on user permissions.

Addressing Information Disclosure vulnerabilities is essential to protect the confidentiality and integrity of sensitive data and prevent unauthorized access to confidential information. Failure to mitigate Information Disclosure vulnerabilities can lead to data breaches, privacy violations, and reputational damage for organizations.

## **Detailed Findings**

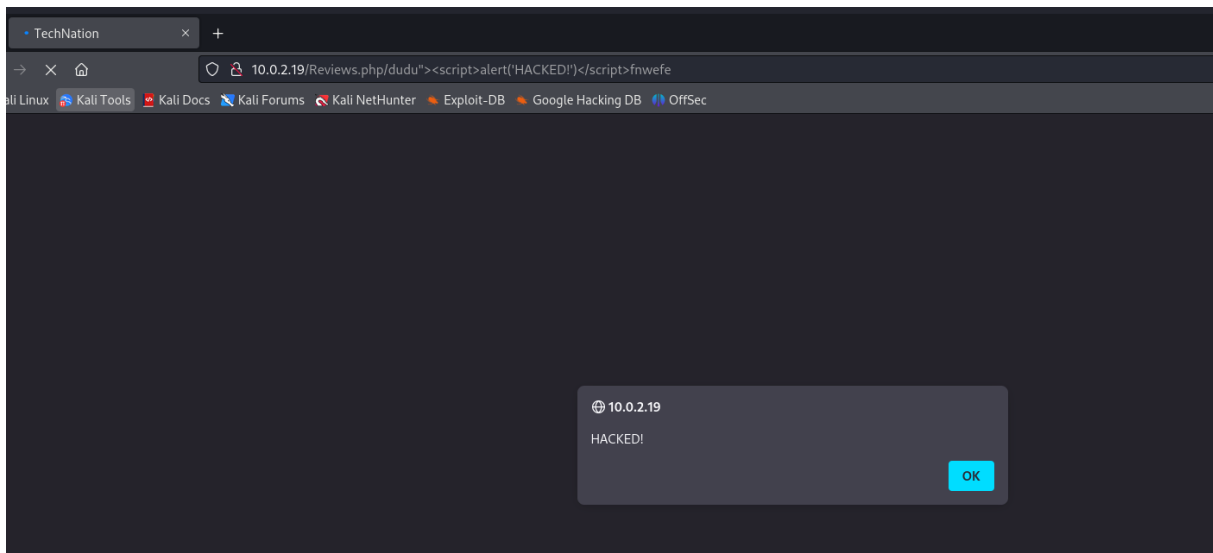
### **Cross Site Scripting**

This input was echoed unmodified in the application's response. This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

Below two examples:

We enter in url bar:

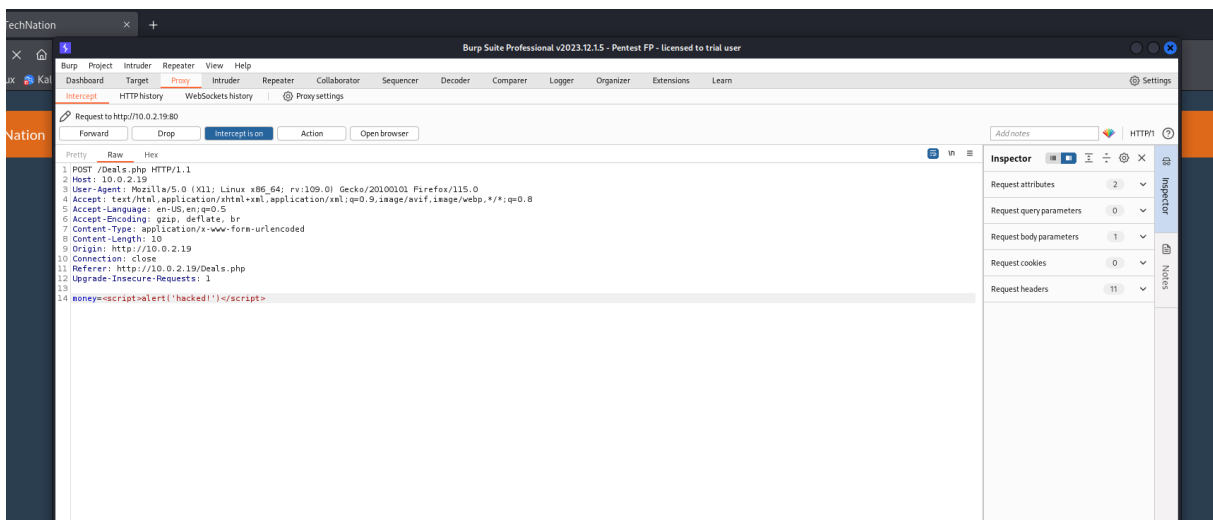
`http://10.0.2.19/Reviews.php/dudu'><script>alert('HACKED!')</script>fnwefe`

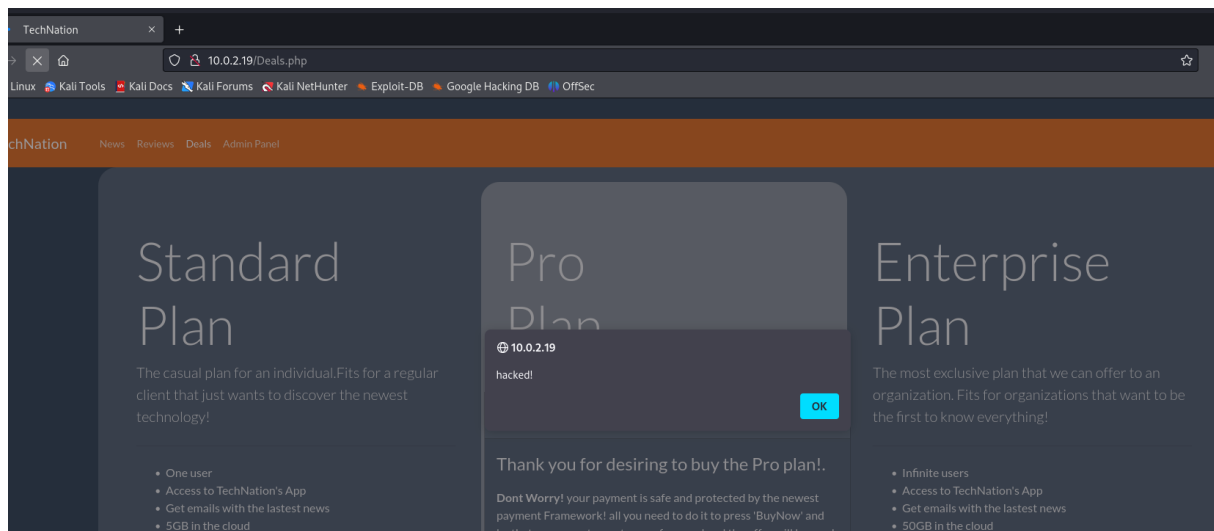


Second example:

Intercept `http://10.0.2.19/Deals.php` and write as value for money parameter:

`<script>alert('hacked!')</script>`





What can be done to avoid cross site scripting attack:

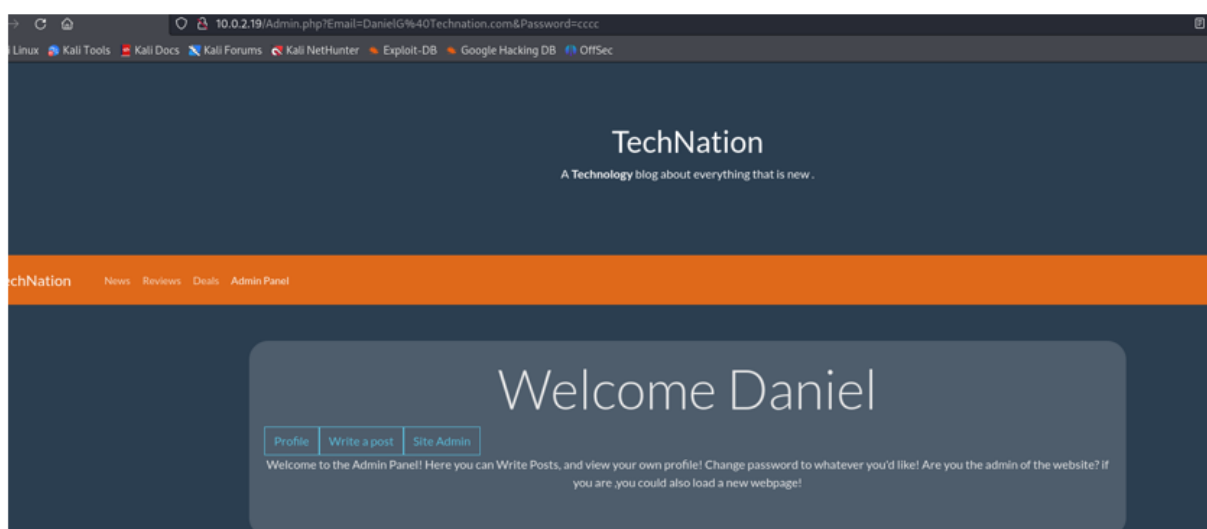
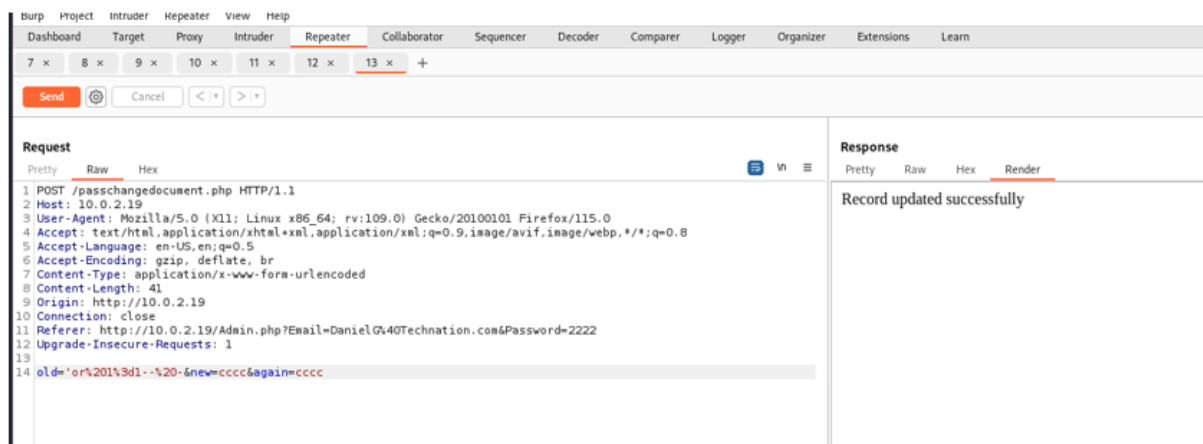
1. Filtering and validating input data: Make sure that any user-input data is filtered and validated to remove or neutralize potentially harmful elements such as JavaScript scripts.
2. Using secure libraries and frameworks: Use secure libraries and frameworks that automatically secure the application against XSS attacks.
3. Implementing Content Security Policy (CSP) mechanisms: CSP allows you to specify which types of resources can be loaded by the browser, thereby limiting the ability to execute scripts.
4. Using HTML Encoding: Before displaying user-entered data on the web page, ensure that it is properly encoded to prevent it from being interpreted as HTML or JavaScript code.
5. `htmlspecialchars` and `htmlspecialchars()` PHP functions should be considered to use.

## SQL Injection

After conducting an attack on the login panel and obtaining credentials of one of the users, we intercept a request related to changing the password. In Burp Repeater, we replace the email with the user's email whose credentials we want to acquire, and the parameters are as follows:

- For 'old', we input `"or 1=1 -- ;`
- For 'new', we provide the new password;
- For 'again', we repeat the new password.





The most effective way to prevent SQL injection attacks is to use parameterized queries (also known as prepared statements) for all database access, to double up any single quotation marks appearing within user input before incorporating that input into a SQL query and use stored procedures for database access.

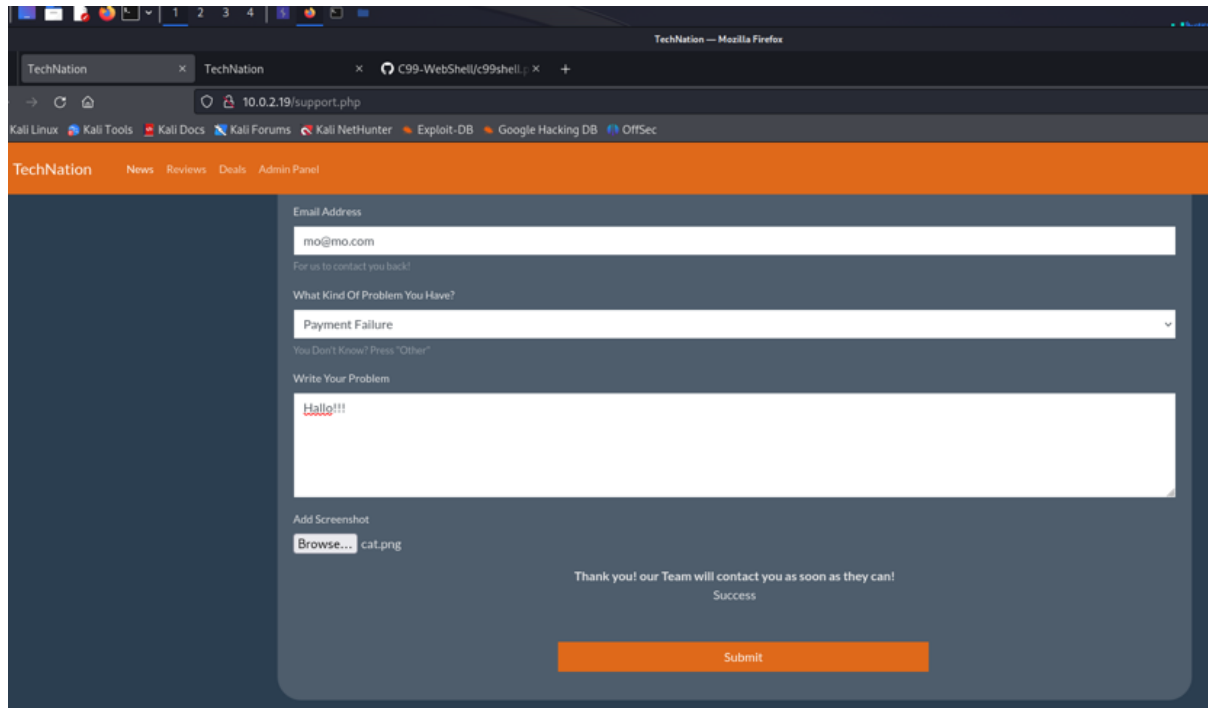
## Unrestricted File Upload

Sending the `c99shell.php` file can serve as an example of exploiting the "Unrestricted file upload" vulnerability. The `c99shell.php` file is typically a malicious PHP file containing a so-called "PHP Shell," which is a script enabling remote execution of commands on the infected server. This PHP Shell file can allow an attacker to remotely take control of the server, execute system commands, browse files, steal data, and perform various other actions.

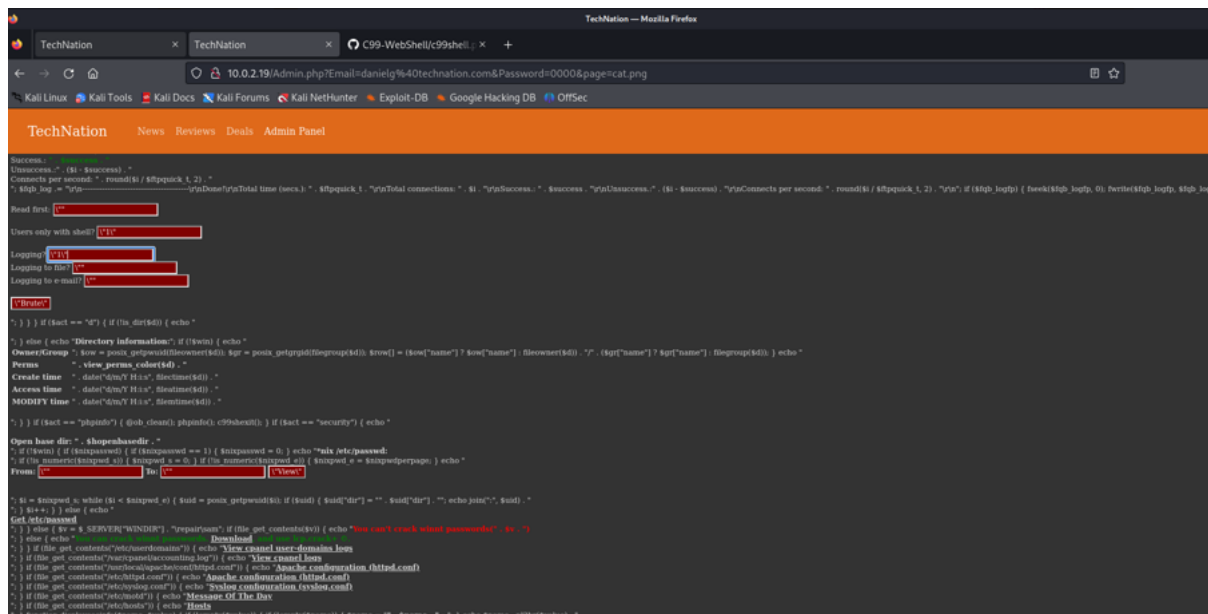
When such a file is uploaded to an application or internet service using the "Unrestricted file upload" vulnerability, the attacker can then gain access to this file by executing it in a browser or using other methods, thereby enabling them to remotely control the infected server.

<https://github.com/cermmik/C99-WebShell/blob/master/c99shell.php>

Change name of the file to hide .php extension



After log in to the panel, change in url bar path http://10.0.2.19/.....&page=cat.png



In general, the most effective way to protect your own websites from these vulnerabilities is to implement all of the following practices:

1. Check the file extension against a whitelist of permitted extensions rather than a blacklist of prohibited ones. It's much easier to guess which extensions you might want to allow than it is to guess which ones an attacker might try to upload.
2. Make sure the filename doesn't contain any substrings that may be interpreted as a directory or a traversal sequence (../).
3. Rename uploaded files to avoid collisions that may cause existing files to be overwritten.
4. Do not upload files to the server's permanent filesystem until they have been fully validated.
5. As much as possible, use an established framework for preprocessing file uploads rather than attempting to write your own validation mechanisms.

## Credential Exposure via URL

Login credentials sent via the GET method they are visible in url bar.

The image shows a web browser window at the top and a Burp Suite interface at the bottom. The browser window displays the TechNation website with the URL `10.0.2.19/Admin.php?Email=monika%40mo.com&Password=hacked` in the address bar. The Burp Suite interface shows the 'Repeater' tab with a request list. The first request is a GET request to `/Admin.php?Email=monika%40mo.com&Password=hacked` with HTTP/1.1. The request details are shown in the 'Raw' tab, displaying the full HTTP request including headers like `Host: 10.0.2.19`, `User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0`, and `Referer: http://10.0.2.19/Admin.php?Email=monika%40mo.com&Password=hacked`.

Use method POST instead of GET. Applications should use transport-level encryption (SSL or TLS) to protect all sensitive communications passing between the client and the server. Communications that should be protected include the login mechanism and related functionality, and any functions where sensitive data can be accessed or privileged actions can be performed. These areas should employ their own session handling mechanism, and the session tokens used should never be transmitted over unencrypted communications. If HTTP cookies are used for transmitting session tokens, then the secure flag should be set to prevent transmission over clear-text HTTP.

## Brute Force in Admin Panel

Based on the information found on the website <http://10.0.2.19/decoda9013smith21985.txt>, a list of potential passwords was created. Based on the gathered names and surnames from the webpage, a list of potential usernames was created. A Burp Intruder attack was conducted using the ClusterBomb method.

The screenshot displays the Burp Suite Intruder interface. The main window shows a table of attack results for the URL `http://10.0.2.19`. The table has columns for Request, Payload1, Payload2, Status code, Error, Timeout, Length, and Comment. The first row (Request 21383) shows a successful login for the user `arthurb@technation.com` with a status code of 200 and a length of 10916. A detailed view of this result is shown in a pop-up window titled "Result 21383 | Intruder attack". This window displays the request and response details. The request is a GET request to `/Admin.php?Email=arthurb@technation.com&Password=J4VfskYb3nuGFsQ6` with a status code of 200. The response is an HTML page with a status code of 200 and a length of 10916. The response body contains the text "Welcome Arthur".

Request	Payload1	Payload2	Status code	Error	Timeout	Length	Comment
21383	arthurb@technation.com	J4VfskYb3nuGFsQ6	200			10916	

Result 21383 | Intruder attack

Payload1: arthurb@technation.com  
Payload2: J4VfskYb3nuGFsQ6  
Status code: 200  
Length: 10916  
Timer: 2010

Request: GET /Admin.php?Email=arthurb@technation.com&Password=J4VfskYb3nuGFsQ6 HTTP/1.1  
Host: 10.0.2.19  
User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate, br  
Connection: keep-alive  
Referer: http://10.0.2.19/Admin.php  
Upgrade-Insecure-Requests: 1

Response: Welcome Arthur

To prevent brute force should be implementing the following countermeasures:

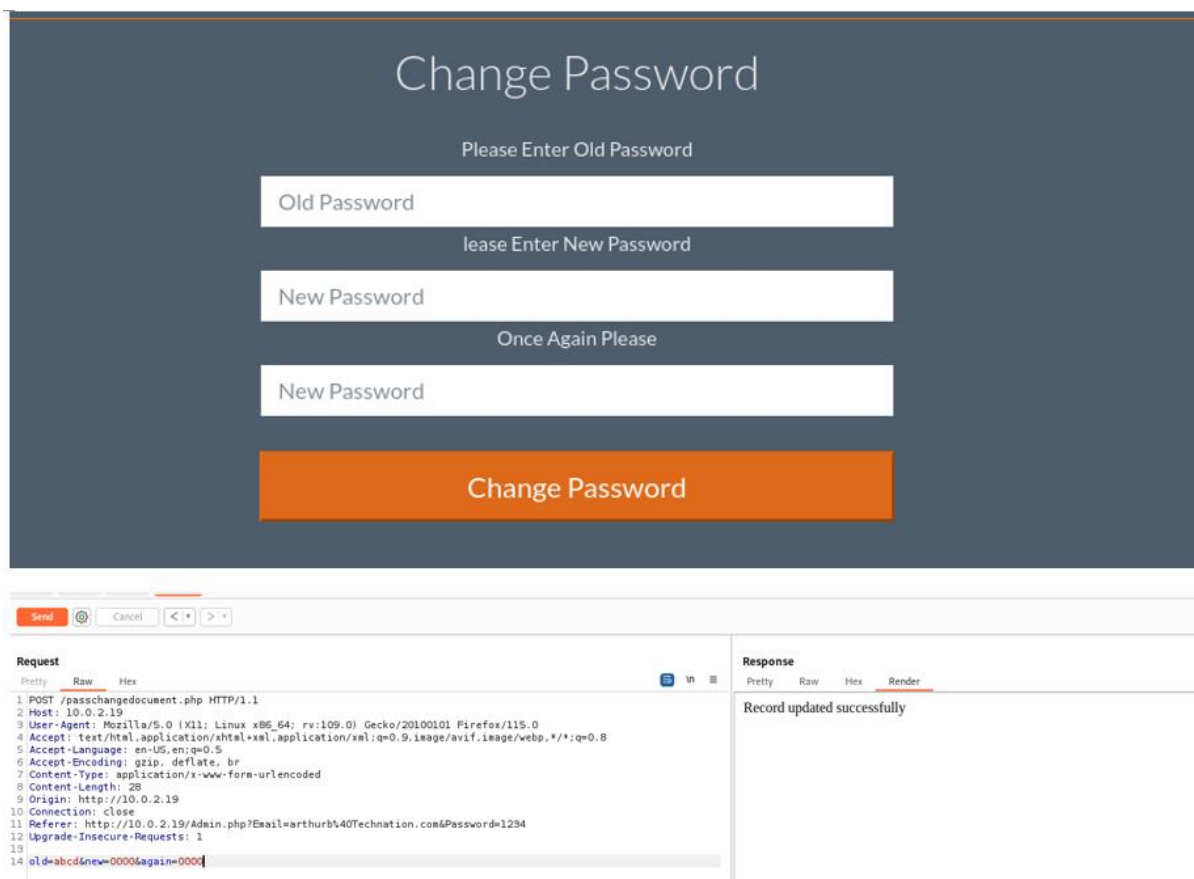
1. Use of strong passwords: Enforce the creation of strong passwords that are difficult to guess by potential attackers. Strong passwords should consist of a combination of letters, numbers, and special characters and be sufficiently long.
2. Limiting the number of failed login attempts: Implement a mechanism to lock the account after a certain number of failed login attempts within a short period of time. For example, after three or five failed login attempts, the account may be temporarily locked.

3. Multi-factor authentication (MFA): Implement multi-step authentication process, which requires not only a password but also an additional authentication factor such as a code generated by an app on the user's phone or sent via SMS.
4. Use CAPTCHA: Add a CAPTCHA mechanism to the login form, which requires users to confirm they are not robots by solving a simple CAPTCHA test.
5. Delete public access to `http://10.0.2.19/decoda9013smith21985.txt`

## Cross-site request forgery

In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. For example, this might be to change the email address on their account, to change their password, or to make a funds transfer. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality. In this case, we change password for user's account.

Via Burp we intercept moment when user changes password and then change it to our new credentials.



## Path Traversal

to  
http://10.0.2.19/Admin.php?Email=danielg%40technation.com&Password=cccc&page=../..../  
../etc/passwd

TechNation — Mozilla Firefox

TechNation x 10.0.2.19/info.txt x +

10.0.2.19/Admin.php?Email=danielg%40technation.com&Password=cccc&pages../../../../\_etc/passwd

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB 0FFSec

TechNation

A Technology blog about everything that is new.

TechNation News Reviews Deals Admin Panel

# Welcome Daniel

Profile Write a post Site Admin

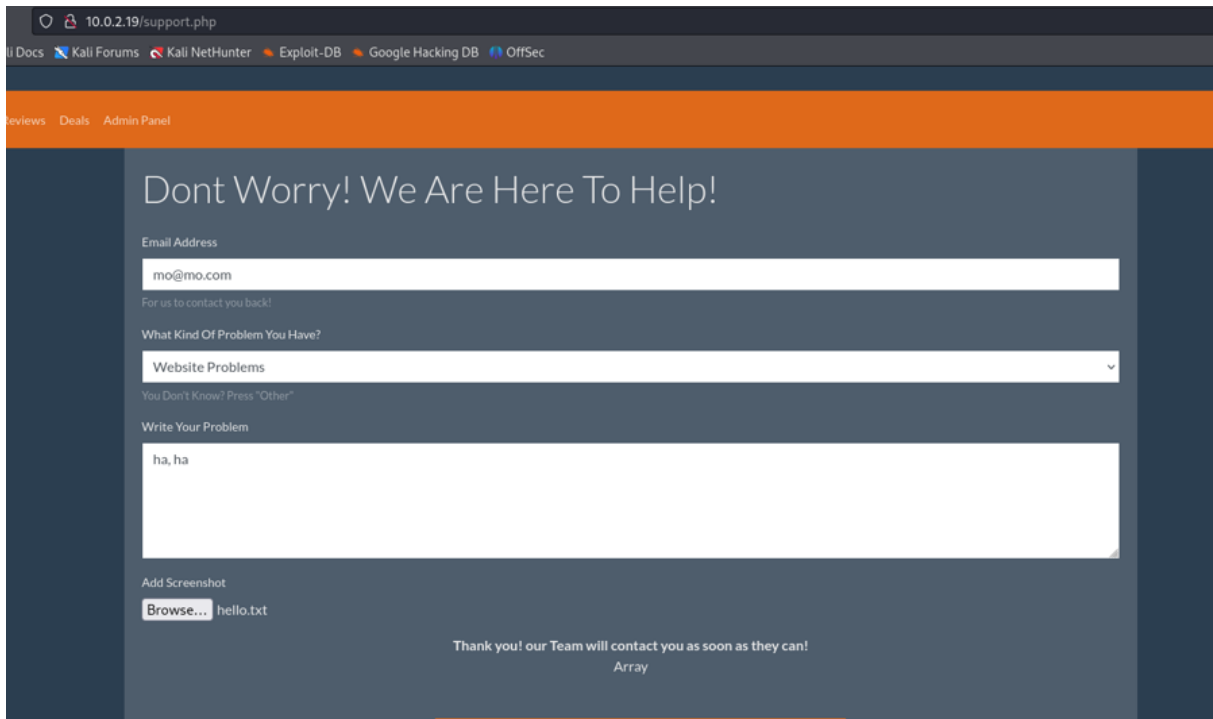
Welcome to the Admin Panel! Here you can Write Posts, and view your own profile! Change password to whatever you'd like! Are you the admin of the website? If you are you could also load a new webpage!

```
root:x:0:0:root:/bin:/usr/sbin/dmccomx:1:1:daemon:/usr/sbin:/usr/sbin/bin:x:2:2:bin:/bin:/usr/sbin/hologin sysxc:3:3:sys:/dev:/usr/sbin/hologin
syncx:4:4:sync:/bin:/bin/sync games:x:5:5:games:/usr/games:/usr/sbin/hologin man:x:6:12:man:/var/cache/man:/usr/sbin/hologin lpx:7:7:lp:/var/hpool
/lpx:/usr/sbin/hologin mail:x:8:8:mail:/var/mail:/usr/sbin/hologin newsc:x:9:9:news:/var/spool/news:/usr/sbin/hologin uucpx:10:10:uucp:/var/hpool/uucp:/usr/sbin
/hologin proxy:x:13:13:proxy:/bin:/usr/sbin/hologin www-data:x:33:33:www-data:/var/www:/usr/sbin/hologin backupx:34:34:backup:/var/backups:/usr/sbin
/hologin list:x:38:38:MailList Manager:/var/list:/usr/sbin/hologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/hologin gnats:x:41:41:Gnats Bug-Reporting System
[admin]:/var/irb:/usr/sbin/hologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/hologin systemd-network:x:100:102:systemd Network
Management...:/run/systemd:/usr/sbin/hologin systemd-resolve:x:101:103:systemd Resolver...:/run/systemd:/usr/sbin/hologin systemd-
timesync:x:102:104:systemd Time Synchronization...:/run/systemd:/usr/sbin/hologin messagebus:x:103:106:/nonexistent:/usr/sbin/hologin
syslog:x:104:110:/home:/syslog:/usr/sbin/hologin _aptc:x:105:65534:/nonexistent:/usr/sbin/hologin tcsc:x:106:111:TPM software stack...:/var/lib/tpm/bin/false
uid:x:107:114:/run/uid:/usr/sbin/hologin tcpdump:x:108:115:/nonexistent:/usr/sbin/hologin avahi-autopk:x:109:116:/Avahi/autopk/daemon...:/var/lib/avahi-
```

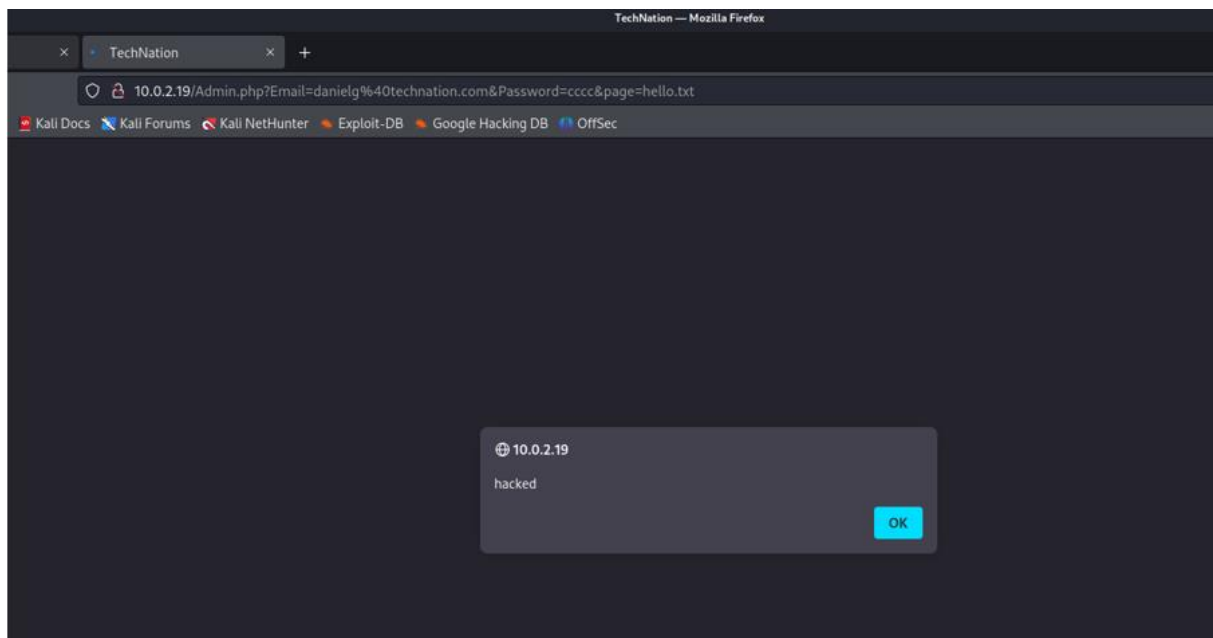
1. Validate the user input before processing it. Ideally, compare the user input with a whitelist of permitted values. If that isn't possible, verify that the input contains only permitted content, such as alphanumeric characters only.
2. After validating the supplied input, append the input to the base directory and use a platform filesystem API to canonicalize the path. Verify that the canonicalized path starts with the expected base directory.

## Local File Inclusion

We send a file to the server with the JS code: `<script>alert('hacked')</script>`. After logging into the admin panel, we change the path to point to the selected file, and an alert appears.



The screenshot shows a web browser window with the address bar displaying `10.0.2.19/support.php`. The browser's bookmark bar includes `Kali Docs`, `Kali Forums`, `Kali NetHunter`, `Exploit-DB`, `Google Hacking DB`, and `OffSec`. The page has a navigation bar with `Reviews`, `Deals`, and `Admin Panel`. The main content area has a heading `Dont Worry! We Are Here To Help!` and a form with the following fields:   
- `Email Address`: `mo@mo.com`   
- `For us to contact you back!`   
- `What Kind Of Problem You Have?`: `Website Problems` (selected from a dropdown)   
- `You Don't Know? Press "Other"`   
- `Write Your Problem`: `ha, ha`   
- `Add Screenshot`: `Browse...` and `hello.txt`   
At the bottom, a message says: `Thank you! our Team will contact you as soon as they can! Array`



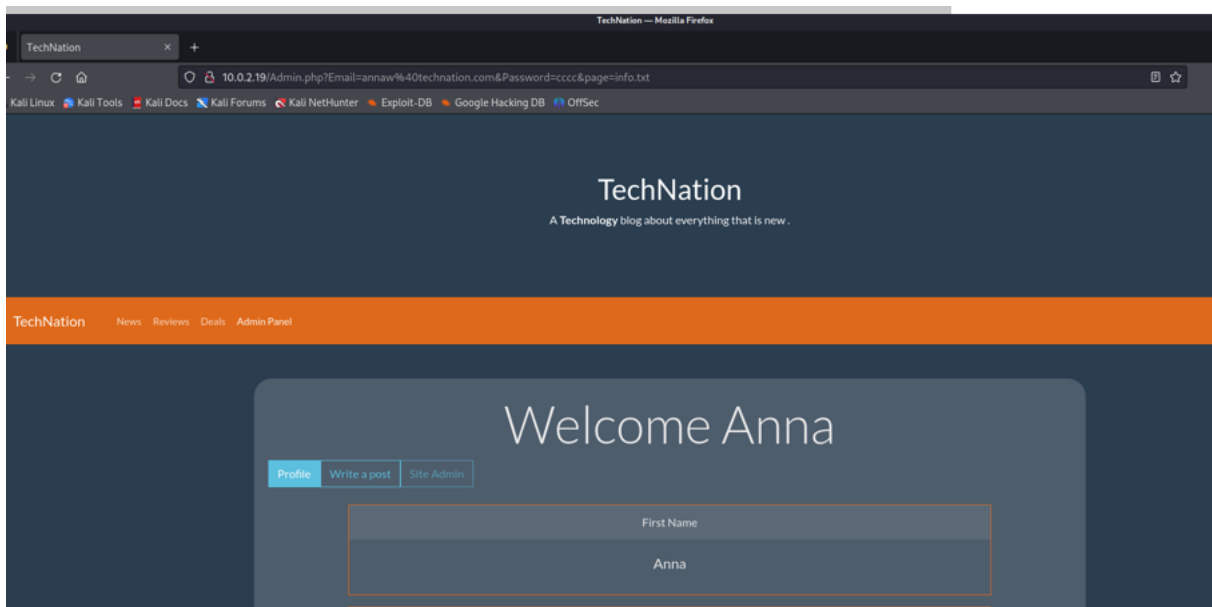
Here are a few ways to prevent LFI attacks:

1. ID assigation – save your file paths in a secure database and give an ID for every single one, this way users only get to see their ID without viewing or altering the path

2. Whitelisting – use verified and secured whitelist files and ignore everything else
3. Use databases – don't include files on a web server that can be compromised, use a database instead
4. Better server instructions – make the server send download headers automatically instead of executing files in a specified directory

## Insecure Direct Object References

This is a type of vulnerability where attackers can gain access to resources they normally wouldn't have access to by manipulating direct references to object identifiers. Here, by changing only the name in the browser's address bar, for example from 'danielg' to 'annaw' we get access to profile of another user.



To secure the application against these types of attacks, it's important to properly authenticate and authorize users and carefully control access to resources.

## Directory Listing

We use script l n Kali Linux: `nmap 10.0.2.19 --script=http-enum`

The purpose of this script is to identify potentially interesting resources, directories, parameters, and other elements that could be used for analysing configurations and potential security vulnerabilities. The http-enum script can assist in identifying hidden directories, administrator-level pages, or software version information, which may be useful for an attacker in finding vulnerabilities.



```
(monika@kali)-[~]
$ nmap 10.0.2.19 --script=http-enum
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-13 16:34 EST
Nmap scan report for 10.0.2.19
Host is up (0.00085s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT      STATE SERVICE
80/tcp    open  http
| http-enum:
|   /robots.txt: Robots file
|_  /css/: Potentially interesting directory w/ listing on 'apache/2.4.41 (ubuntu)'

Nmap done: 1 IP address (1 host up) scanned in 2.13 seconds
```

To secure against Nmap scanning with the http-enum script, several measures can be taken:

1. Firewall: Configure a firewall to block unauthorized scans or excessive queries from a single IP address. Limiting server access at the network level can help prevent attacks.
2. Access Restrictions: Implement access restriction policies on the server, such as limiting access to critical pages or resources only to authorized users.
3. Use IPS/IDS: Deploy an Intrusion Prevention System (IPS) or Intrusion Detection System (IDS) to detect and block unwanted network activity, such as scanning with Nmap.

## HTTP Verb Tampering

Executing command `nmap -p 80 --script http-methods 10.0.2.19`

will allow you to identify which HTTP methods are supported by the server on port 80. This can be useful for understanding how the server responds to various HTTP requests and may provide information about the server's configuration and potential security vulnerabilities.

```
(monika@kali)-[~]
$ nmap -p 80 --script http-methods 10.0.2.19
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-13 16:39 EST
Nmap scan report for 10.0.2.19
Host is up (0.00086s latency).

PORT      STATE SERVICE
80/tcp    open  http
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS

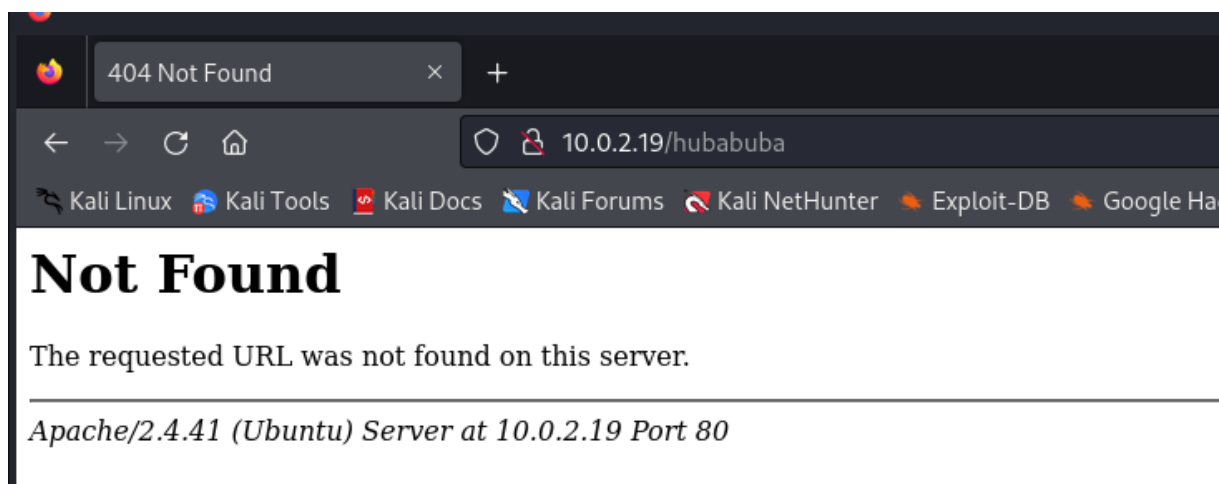
Nmap done: 1 IP address (1 host up) scanned in 0.47 seconds
```

To protect against this kind of probing, you can take several measures:

1. Access Control: Apply firewall rules to limit access to port 80 only to necessary IP addresses. This can help minimize the possibility of probing from untrusted sources.
2. Server Configuration: Configure the HTTP server to handle HTTP requests properly and not disclose unnecessary configuration information.
3. Limiting HTTP Methods: If there is no need to support all HTTP methods on the server, restrict their availability only to those that are necessary for the application's operation.
4. Use of Application Layer Security: Utilize security mechanisms available within the HTTP server, such as security modules, content filters, and access control lists, to prevent unwanted probing and attacks.

## Server Version Disclosure

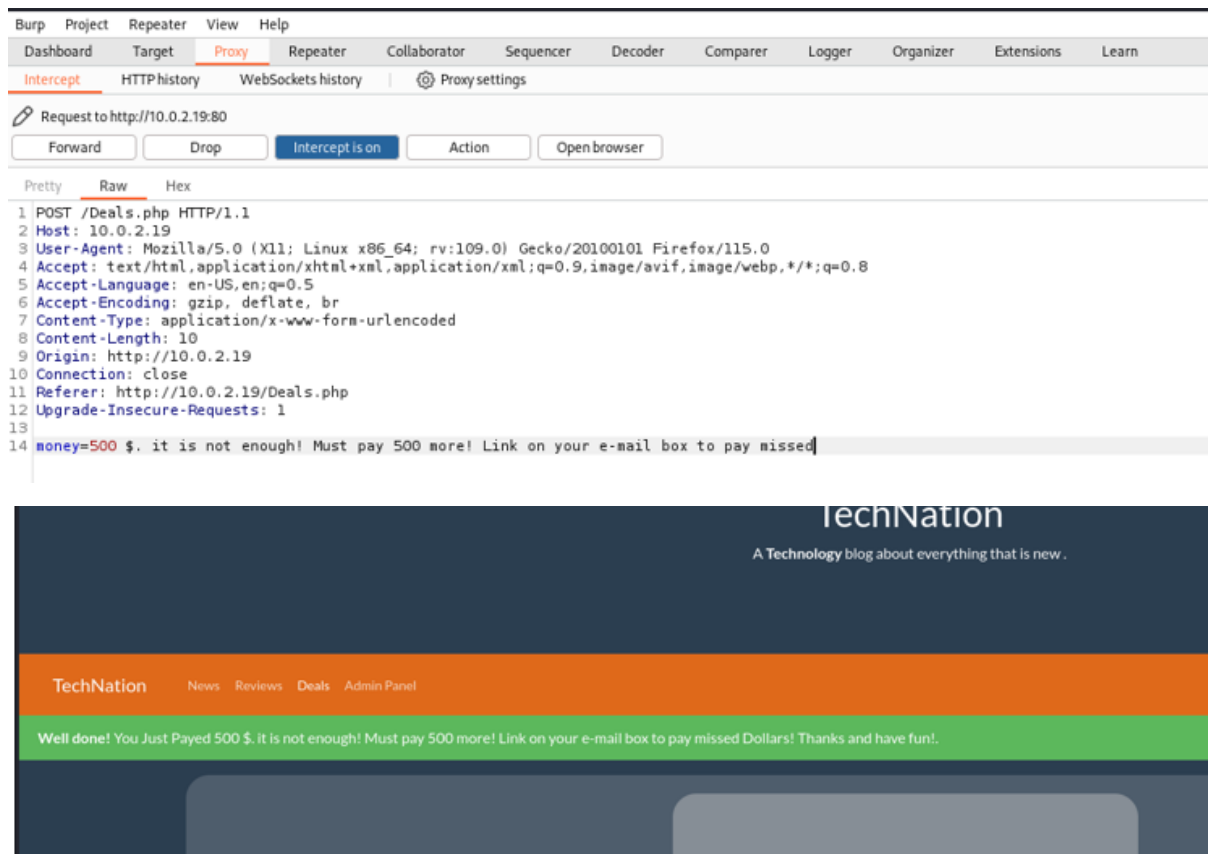
By entering an address on a domain that doesn't exist, we are shown a page displaying server version information. This leads to the possibility of finding an appropriate exploit and gaining unauthorized access to the server.



The proper server configuration will prevent such information from being displayed.

## Parameter Tampering

The Web Parameter Tampering attack is based on the manipulation of parameters exchanged between client and server in order to modify application data, such as user credentials and permissions, price and quantity of products, etc. In this case parameter money was manipulated.



To protect against parameter tampering, several measures can be taken:

1. **Data Validation:** Validate the input data passed through parameters. Ensure they adhere to the expected format and type to prevent improper modifications.
2. **Authentication and Authorization:** Utilize proper authentication and authorization mechanisms to control access to server functions and resources. Limit user permissions only to necessary actions.
3. **Session Management:** Use sessions to store session state information and authentication data. Ensure sessions are secure and protected against session hijacking or session fixation attacks.
4. **Data Encryption:** When transmitting sensitive data via parameters, employ encryption protocols such as HTTPS to prevent interception and manipulation of data during transmission.
5. **Parameter Access Limitation:** Implement appropriate access controls to prevent unauthorized users from manipulating parameters. Check user permissions before performing operations on parameters.

## Information Disclosure

Revealing the names of hidden directories, their structure, and their contents via a robots.txt file or directory listing – here using tool gobuster in Kali Linux.

```
(monika@kali)~[~/Pentest]
$ gobuster dir -u http://10.0.2.19 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt

Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url: http://10.0.2.19
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Timeout: 10s

Starting gobuster in directory enumeration mode

/icon (Status: 301) [Size: 305] [→ http://10.0.2.19/icon/]
/css (Status: 301) [Size: 304] [→ http://10.0.2.19/css/]
/javascript (Status: 301) [Size: 311] [→ http://10.0.2.19/javascript/]
/JS (Status: 301) [Size: 303] [→ http://10.0.2.19/JS/]
/Editor (Status: 301) [Size: 307] [→ http://10.0.2.19/Editor/]
/server-status (Status: 403) [Size: 274]
Progress: 220560 / 220561 (100.00%)

Finished
```

To defend against revealing the names of hidden directories, their structure, and their contents via a robots.txt file or directory listing, you can take the following measures:

1. **Proper Directory Configuration:** Ensure that your web server is properly configured to prevent directory listing. This means disabling directory browsing so that users cannot view the contents of directories directly.
2. **Robots.txt Management:** Carefully manage your robots.txt file to control what directories and files search engines are allowed to crawl and index. Avoid including sensitive directories or files in your robots.txt file.
3. **Use of Authentication:** Protect sensitive directories with authentication mechanisms such as password protection or user authentication. This adds an additional layer of security by requiring users to provide credentials before accessing the directory.
4. **Access Control:** Implement access controls at the server level to restrict access to sensitive directories based on user roles and permissions. Only authorized users should be allowed to access certain directories.

## **Conclusion and Security Improvement Suggestion**

Summary:

The website currently has several significant vulnerabilities, posing a serious threat to its security. Most of these vulnerabilities are active, meaning they are easily exploitable by potential attackers.

### **Improvement Suggestions:**

Regular Software Updates: Regularly update server and web application software to prevent exploitation of known security vulnerabilities.

Implementation of Data Filtering and Validation Mechanisms: Implement data filtering and validation mechanisms to prevent vulnerabilities such as Cross Site Scripting (XSS), SQL Injection, and Parameter Tampering.

Enhancement of Access Controls: Implement access controls to restrict access to sensitive functions and resources only to authorized users. Avoid exposing sensitive data through URLs (Credential Exposure via URL).

Removal of Unnecessary Functions and Resources: Ensure there are no unnecessary functions and resources that could be potential targets for attacks (e.g., Directory Listing, HTTP Verb Tampering).

Implementation of Incident Monitoring and Response: Implement incident monitoring and response system to detect and quickly respond to security breach attempts.

### **Website Security Assessment:**

Based on the identified vulnerabilities, the website is assessed to be at high risk of attacks and security compromise. Immediate actions are required to improve the website security.

### **Suggestions for the Website Owner:**

Immediate Vulnerability Patching: Urgently patch all identified vulnerabilities, applying the security improvement suggestions listed above.

Review of Server and Application Configuration: Conduct a review of server and application configuration to ensure all security measures are properly configured and implemented.

Staff Training: Organize training for personnel responsible for maintaining and managing the website on security best practices to minimize the risk of vulnerabilities related to human factors.

Regular Security Audits: Conduct regular security audits to monitor and prevent new vulnerabilities and respond to potential threats.