Lifestyle Store

Detailed Developer Report

Security Status: Extremely Vulnerable

The website is extremely vulnerable. If the attacker wants he can take over the entire website by installing a backdoor which will enable him to View, Add, Edit, Delete files and folders (Shell Upload). The attacker can steal or modify databases on the server by exploiting the SQL Injection attacks.

Attacks with lesser damage can also be carried out which won't affect the website or servers but will harm the users of the website.

Attacks such as XSS, Open redirection will not only jeopardize the website but also the users and their security. The attacker can defame the website and collect confidential information about its users such as mobile number, address, username and more details of all customers (IDOR).

Anybody intercepting the requests and responses will be able to observe confidential data because the whole communication is unencrypted. The passwords and usernames are not encrypted. All in all the website has several high impact vulnerabilities which if not patched will cause great loss financially and harm the reputation also.

Vulnerability Statistics

Critical	Severe	Moderate	Low
15	24	6	7

Vulnerabilities

No	Severity	Vulnerability	Count
1.	Critical	SQL Injection	2
2.	Critical	Insecure Direct Object Reference(IDOR)	3
3.	Critical	Default or Weak Password	2
4.	Critical	Arbitrary File Inclusion	1
5.	Critical	Reflected and Stored Cross Site Scripting(XSS)	4
6.	Critical	Brute force Exploitation	2
7.	Critical	Command Execution Vulnerability	1
8.	Severe	Information Disclosure (Source Code Disclosure)	1
9.	Severe	Clear Text Submission of Password	10
10.	Severe	PII Leakage	1
11.	Severe	Cross Site Request Forgery(CSRF)	3
12.	Severe	Rate Limiting Flaws	8
13.	Severe	Open Redirection	1

Vulnerabilities

14.	Moderate	Information Disclosure due to Directory Listing	2
15.	Moderate	Using Components having known Vulnerabilities	4
16.	Low	Unencrypted Communication	1
17.	Low	Improper Data Validation (Server Side)	1
18.	Low	Information Disclosure due to Default Pages	3
19.	Low	Improper Error Handling	2

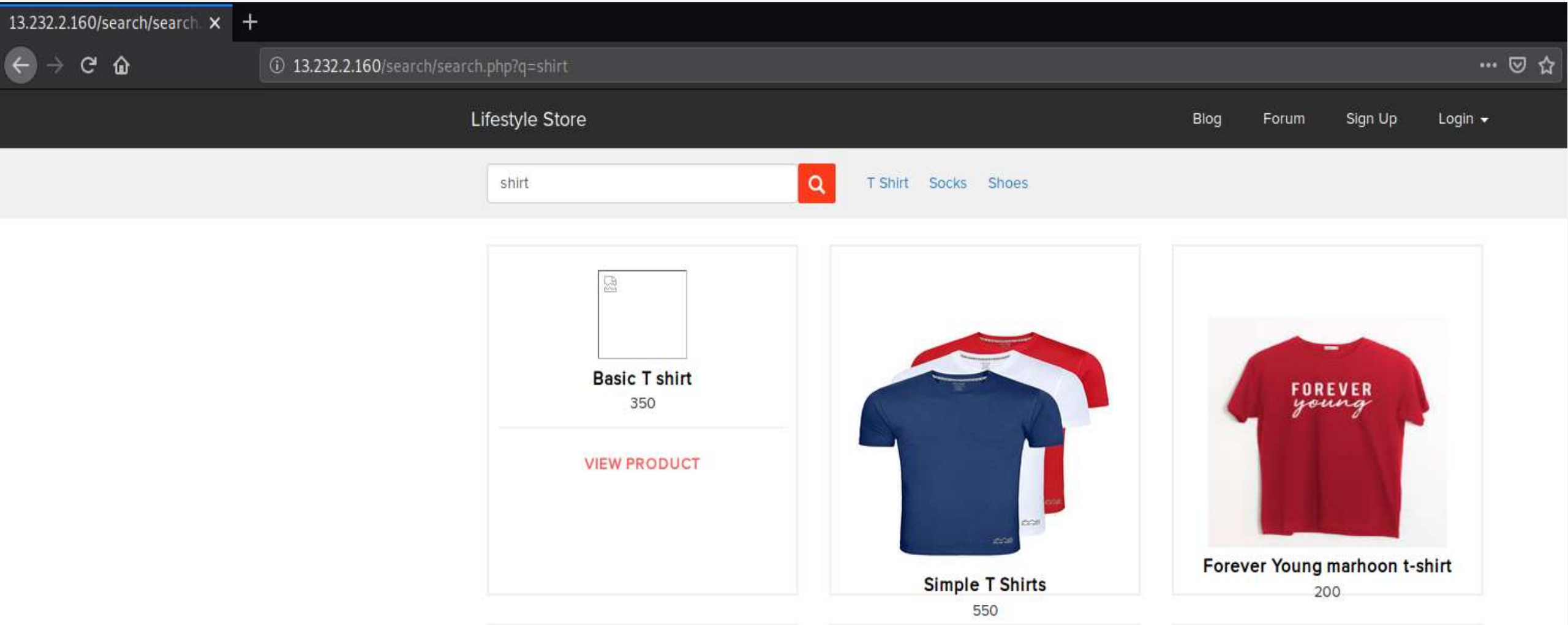
1. SQL Injection

SQL injection is the placement of malicious code in SQL statements, via web page input. A wide range of damaging attacks can often be delivered via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges within the database and taking control of the database server.
Upon testing the below mentioned URL's were found to be vulnerable, details are as below:

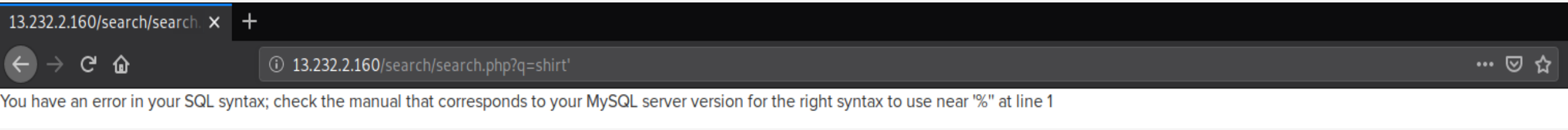
	Affected URL	Parameters	Payload
SQL Injection (critical)	http://13.232.2.160/search/search.php?q=shirt	q(GET Parameter)	q=shirt‘ or 1=1 --+
	http://13.232.2.160/products.php?cat=1	cat(GET parameter)	cat=1‘ or 1=1 --+

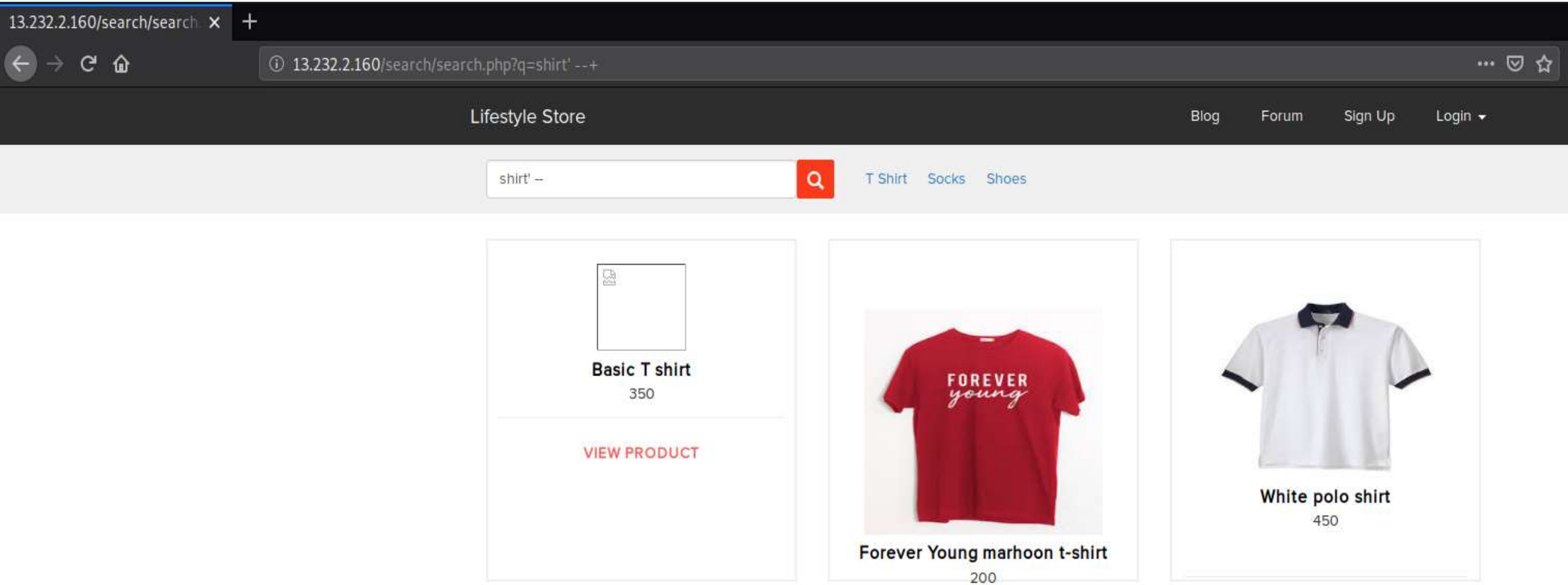
Observation

As we navigate to 13.232.2.160 we find ourselves a welcome page. After clicking on shop now we are directed to a page listing all items where we have a search bar.

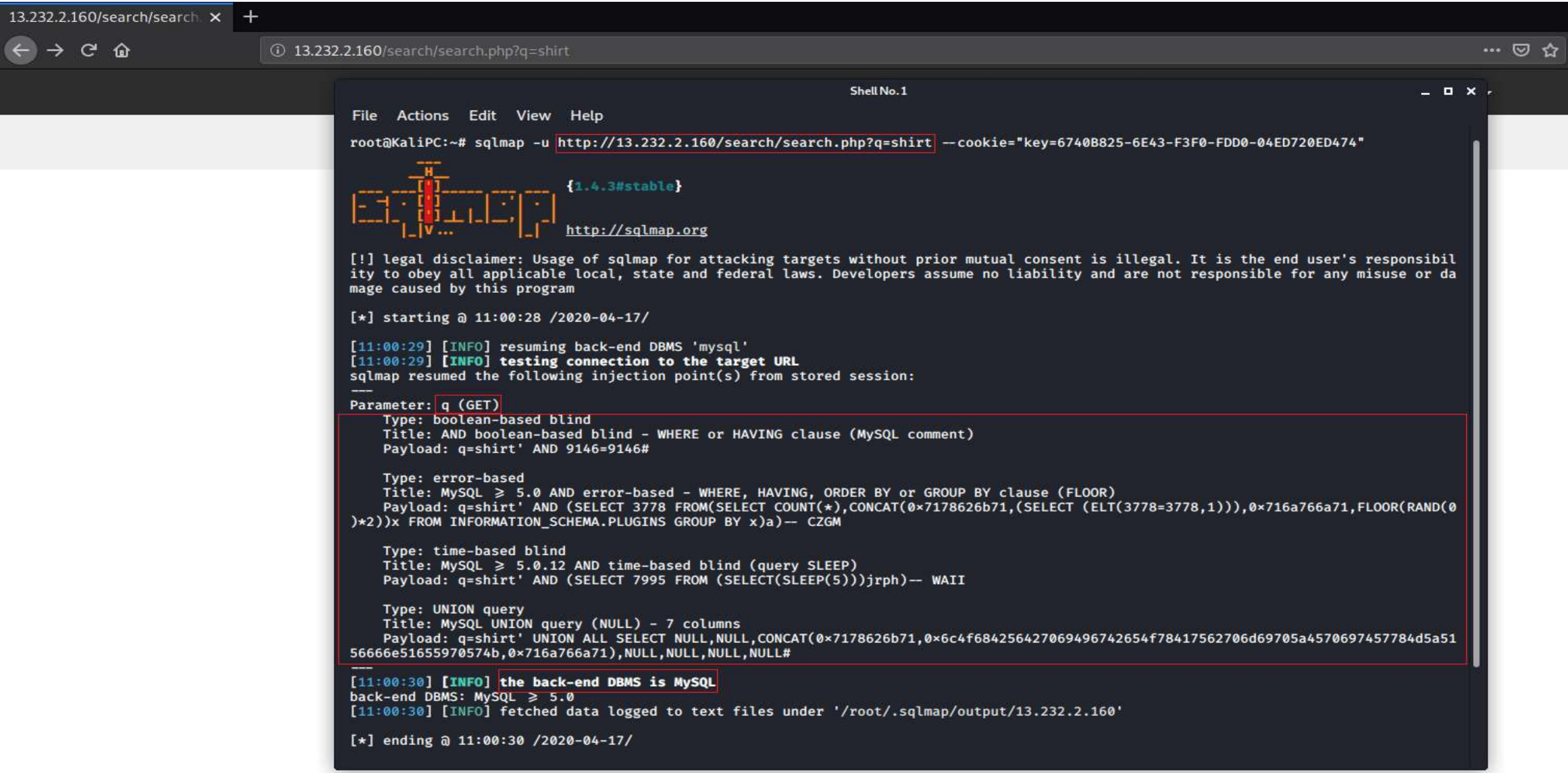


Upon tinkering with the search input, it was found that the error message was encountered for an invalid input. After that “ --+ ” expression was put which negates the effects of “ ‘ ” , henceforth confirming that the URL is vulnerable to SQL Injection.





Next , SQLmap was used to determine vulnerabilities in depth on the q(GET parameter) & cat(GET parameter) of which both yielded similar results.



Proof of Concept (PoC)

The databases, their tables, the contents etc. All the elements of the database in use were accessible through SQL Injection.

```
[11:12:57] [INFO] fetching database names
available databases [2]:
[*] hacking_training_project
[*] information_schema
```

The “hacking_training_project” database yielded these tables, information in these which when exploited pose a great threat to the organisation.

```
[11:20:22] [INFO] fetching tables for database: 'hacking_training_project'
Database: hacking_training_project
[10 tables]
+-----------------+
| brands          |
| cart_items      |
| categories      |
| customers       |
| order_items     |
| orders          |
| product_reviews |
| products        |
| sellers         |
| users           |
+-----------------+
```

```
Database: hacking_training_project
Table: sellers
[3 entries]
+---------+------+--------+--------+----------------------------+---------------------+-------------+---------------------+
| user_id | id   | city   | rating | website                    | created_at          | pan_number  | last_updated_at     |
+---------+------+--------+--------+----------------------------+---------------------+-------------+---------------------+
| 4       | 1    | Delhi  | 4      | www.chandanstore.com       | 2019-02-12 18:30:00 | AWQRD7856Q12 | 2019-02-12 18:30:00 |
| 6       | 2    | Ajmer  | 6      | www.radhikafancystore.com  | 2019-02-12 18:30:00 | JGNW147GT8K | 2019-02-12 18:30:00 |
| 7       | 3    | Batwan | 5      | www.batwanworldfashion.com | 2019-02-12 18:30:00 | LJSDLJ45785S | 2019-02-12 18:30:00 |
+---------+------+--------+--------+----------------------------+---------------------+-------------+---------------------+
```

Business Impact : Extremely High

Using this vulnerability, attacker can execute arbitrary SQL commands on this server and gain complete access to internal databases along with all customer data inside it.
Alongside is the screenshot of users table which shows user credentials being leaked ,although with hashing, dictionary-based and logic-based brute forcing passwords still remains an option to verify with.

```
Database: hacking_training_project
Table: users
[15 entries]
+------+-----------------------------+-----------------------+----------+--------------------------------------------------------
-----------------------------------------------------+---------------+--------------------------------------------------------
-+---------------------+-----------------------------------+--------------+---------------------+
| id   | email                       | name                  | type     | address
                                                     | user_name     | password
 | created_at          | unique_key                        | phone_number | last_updated_at     |
+------+-----------------------------+-----------------------+----------+--------------------------------------------------------
-----------------------------------------------------+---------------+--------------------------------------------------------
-+---------------------+-----------------------------------+--------------+---------------------+
| 1    | admin@lifestylestore.com    | admin                 | admin    | Scholiverse Educare Pvt. Ltd. B-610, Unitech Business Zone,
 Nirvana Country, South City 2, Gurgaon, India - 122018  | admin         | $2y$10$xkmdvrxSCxqdyWSrDx5YSe1NAwX.7pQ2nQmaTCovH4CFssxgyJTki
 | 2019-02-15 12:55:00 | 15468927955c66694cba1174.29688447 | 8521479630   | 2019-02-15 12:55:00 |
| 2    | donald@lifestylestore.com   | Donald Duck           | customer | B-34/ the duck lane, Disneyland
                                                     | Donal234      | $2y$10$PM.7nBSP5FMaldXiM/S3s./p5xR6GTKvjry7ysJtxOkBqOJURAHsO
 | 2019-02-15 12:56:17 | 778522555c6669996f5a24.34991684   | 9489625136   | 2019-02-15 12:56:17 |
| 3    | Pluto@lifestylestore.com    | Brutus                | customer | A-56 Sailor's ship, popeyeworld
                                                     | Pluto98       | $2y$10$xkmdvrxSCxqdyWSrDx5YSe1NAwX.7pQ2nQmaTCovH4CFssxgyJTki
 | 2019-02-15 12:58:03 | 19486318945c666a037b1432.99985767 | 8912345670   | 2019-02-15 12:58:03 |
| 4    | chandan@lifestylestore.com  | Chandan               | seller   | GF-213, Nehru road, old Delhi market, 120078
                                                     | chandan       | $2y$10$4cZBEIrgthXdvT1hwUlivuFELe03rR.GIcdp03NjrlS0VeiOKLVDa
 | 2019-02-15 12:58:59 | 12404594545c666a3b49e0f8.08173871 | 7854126395   | 2019-02-15 12:58:59 |
| 5    | popeye@lifestylestore.com   | Popeye the sailor man | customer | B-44 spinach house, Disneyworld
```

Recommendation

The most effective way to prevent SQL injection attacks is to use parameterized queries (also known as prepared statements) for all database access. This method uses two steps to incorporate potentially tainted data into SQL queries: first, the application specifies the structure of the query, leaving placeholders for each item of user input; second, the application specifies the contents of each placeholder. Because the structure of the query has already been defined in the first step, it is not possible for malformed data in the second step to interfere with the query structure. You should review the documentation for your database and application platform to determine the appropriate APIs which you can use to perform parameterized queries. It is strongly recommended that you parameterize every variable data item that is incorporated into database queries, even if it is not obviously tainted, to prevent oversights occurring and avoid vulnerabilities being introduced by changes elsewhere within the code base of the application.

You should be aware that some commonly employed and recommended mitigations for SQL injection vulnerabilities are not always effective:

- One common defense is to double up any single quotation marks appearing within user input before incorporating that input into a SQL query. This defense is designed to prevent malformed data from terminating the string into which it is inserted. However, if the data being incorporated into queries is numeric, then the defense may fail, because numeric data may not be encapsulated within quotes, in which case only a space is required to break out of the data context and interfere with the query. Further, in second-order SQL injection attacks, data that has been safely escaped when initially inserted into the database is subsequently read from the database and then passed back to it again. Quotation marks that have been doubled up initially will return to their original form when the data is reused, allowing the defense to be bypassed.

- Another often cited defense is to use stored procedures for database access. While stored procedures can provide security benefits, they are not guaranteed to prevent SQL injection attacks. The same kinds of vulnerabilities that arise within standard dynamic SQL queries can arise if any SQL is dynamically constructed within stored procedures. Further, even if the procedure is sound, SQL injection can arise if the procedure is invoked in an unsafe manner using user-controllable data.

References

https://www.owasp.org/index.php/SQL_Injection
https://en.wikipedia.org/wiki/SQL_injection
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
https://websec.ca/kb/sql_injection

Vulnerability Classification: CWE-89, CWE-94, CWE-116

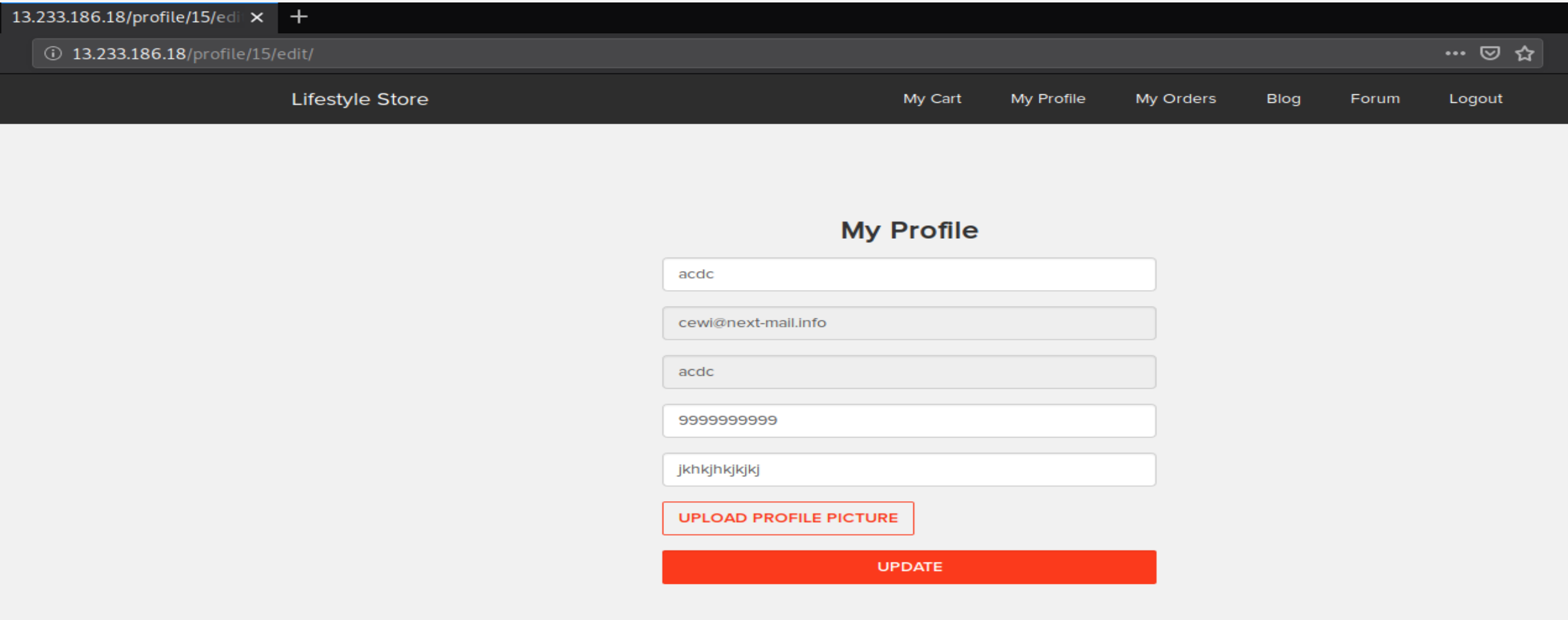
2. Insecure Direct Object Reference

Insecure direct object references (IDOR) are a type of access control vulnerability that arises when an application uses user-supplied input to access objects directly. Here the POST parameter (user_id) and GET parameter (customer) are vulnerable to the IDOR vulnerability which causes disclosure of customer details such as the order history and personal information from other user accounts.

	Affected URL	Parameters	Payload
Insecure Direct Object Reference (critical)	http://13.233.186.18/profile/16/edit/	user_id(POST Parameter)	Changing the user_id from 16 to 15
	http://13.233.186.18/orders/orders.php?customer=16	customer(GET parameter)	Changing the customer parameter from 16 to 15
	http://13.126.119.22/forum/index.php?u=/user/profile/2	u(GET parameter)	Changing the u=/user/profile/ 3 to 1 or 2

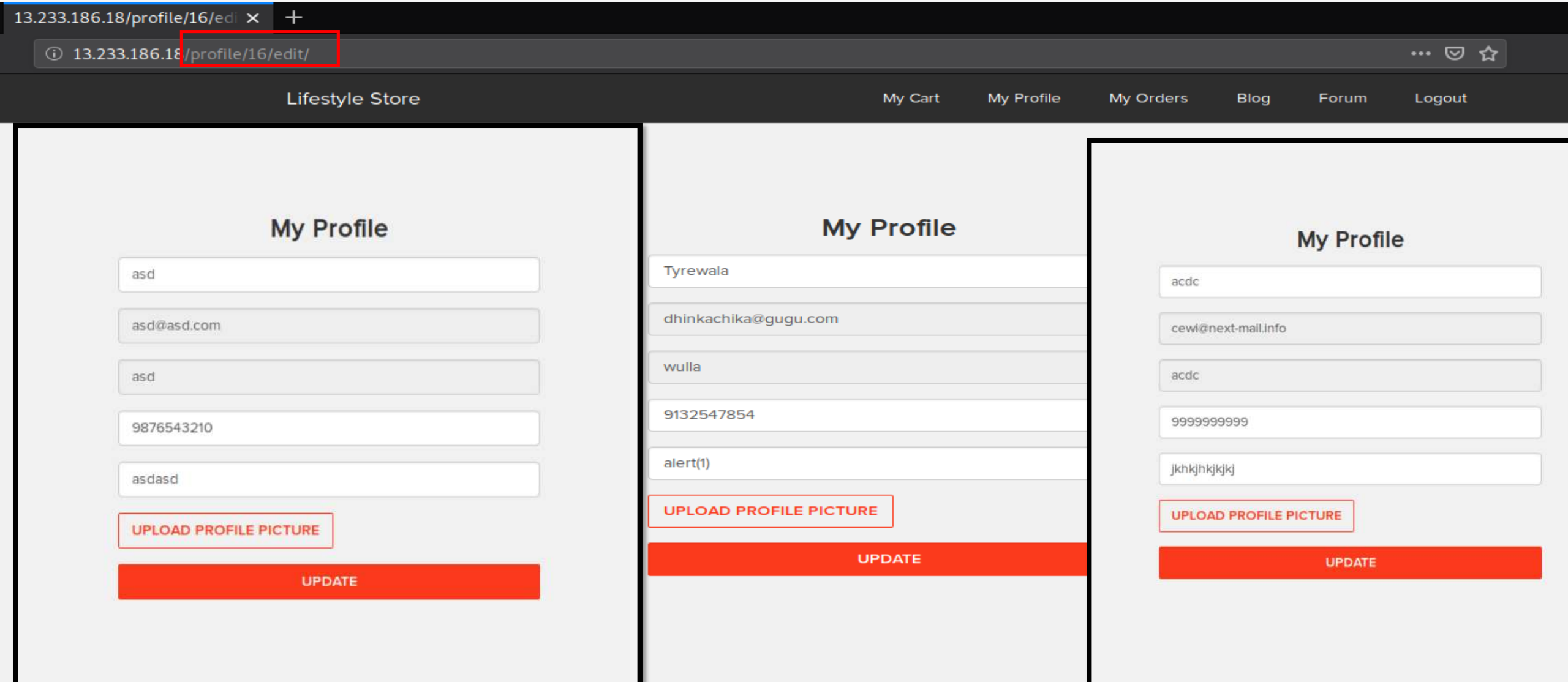
Observation

If we want to edit our profile we are directed to “/profile/15/edit/” directory in the url, we can observe a number in the url and hence upon tinkering with it we find we can navigate through all the edit sections of all the profiles.

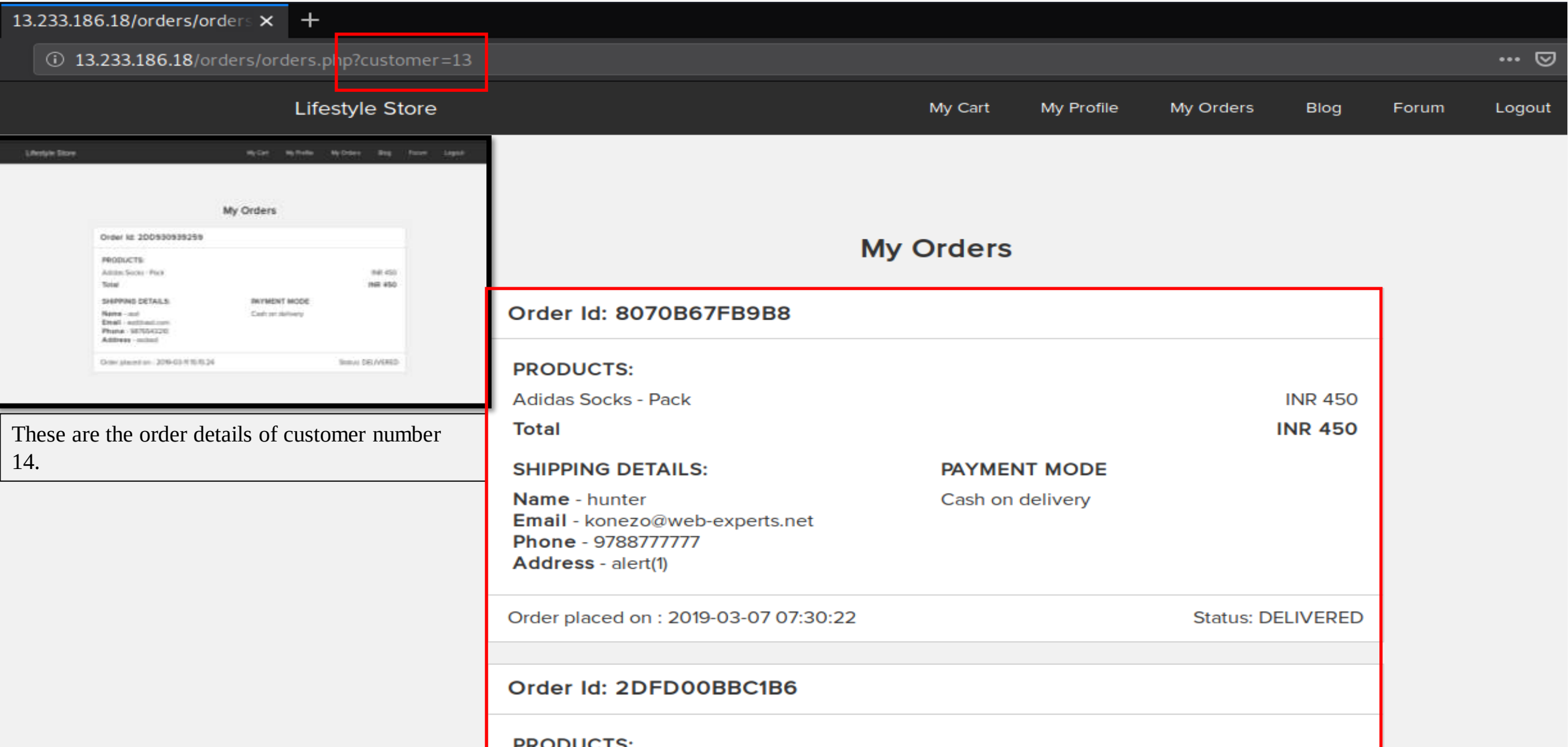


Proof of Concept (PoC)

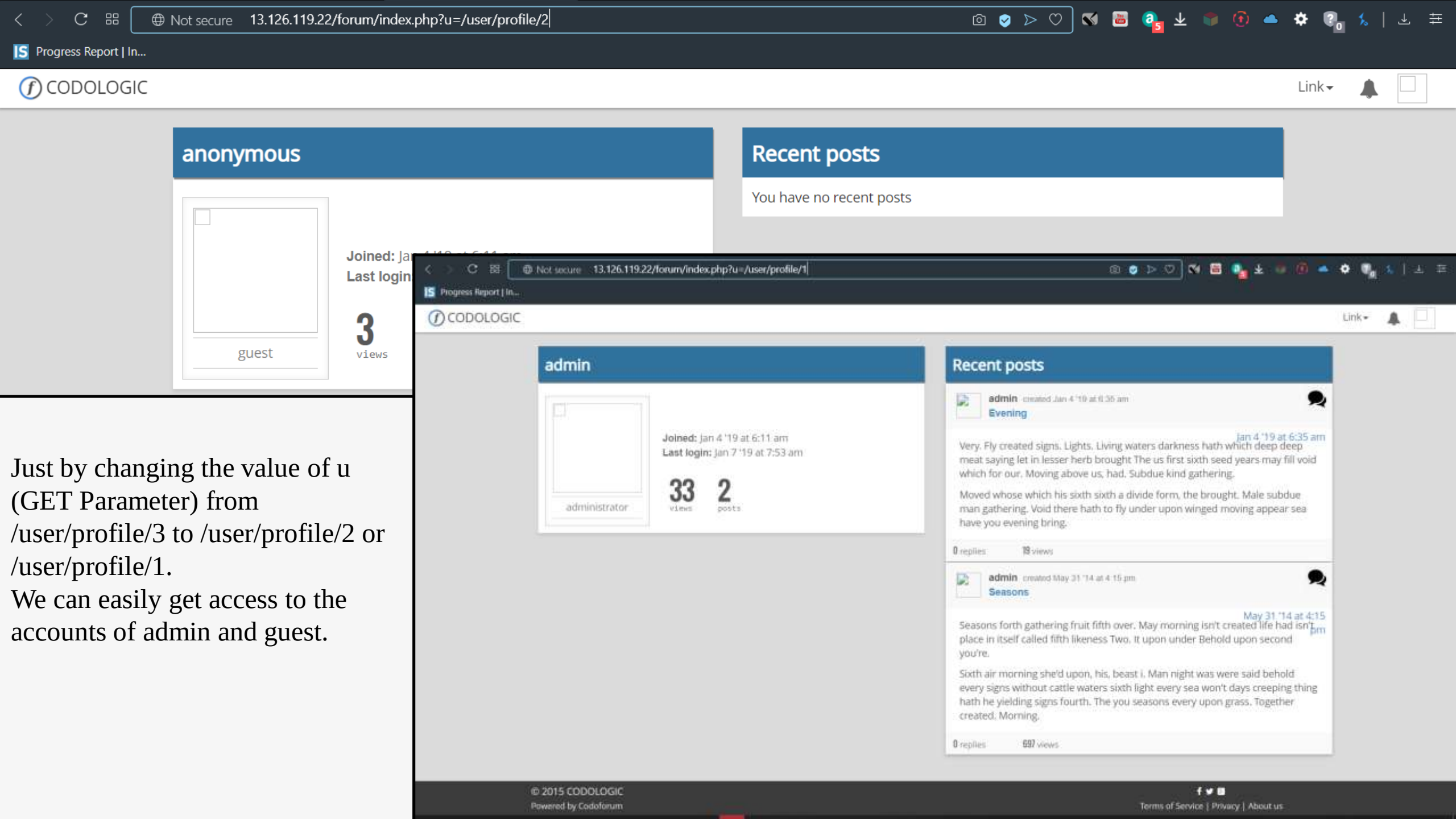
Below are the images of profiles that we can access just by changing the value of the POST parameter (user_id) and disclose personal information of other users.



When we go to my orders , we can notice a GET parameter “customer” in the URL and we see an associated number with it, we tinker with the number increasing it or decreasing it and find ourselves several order histories of other users which disclose very important personal information.



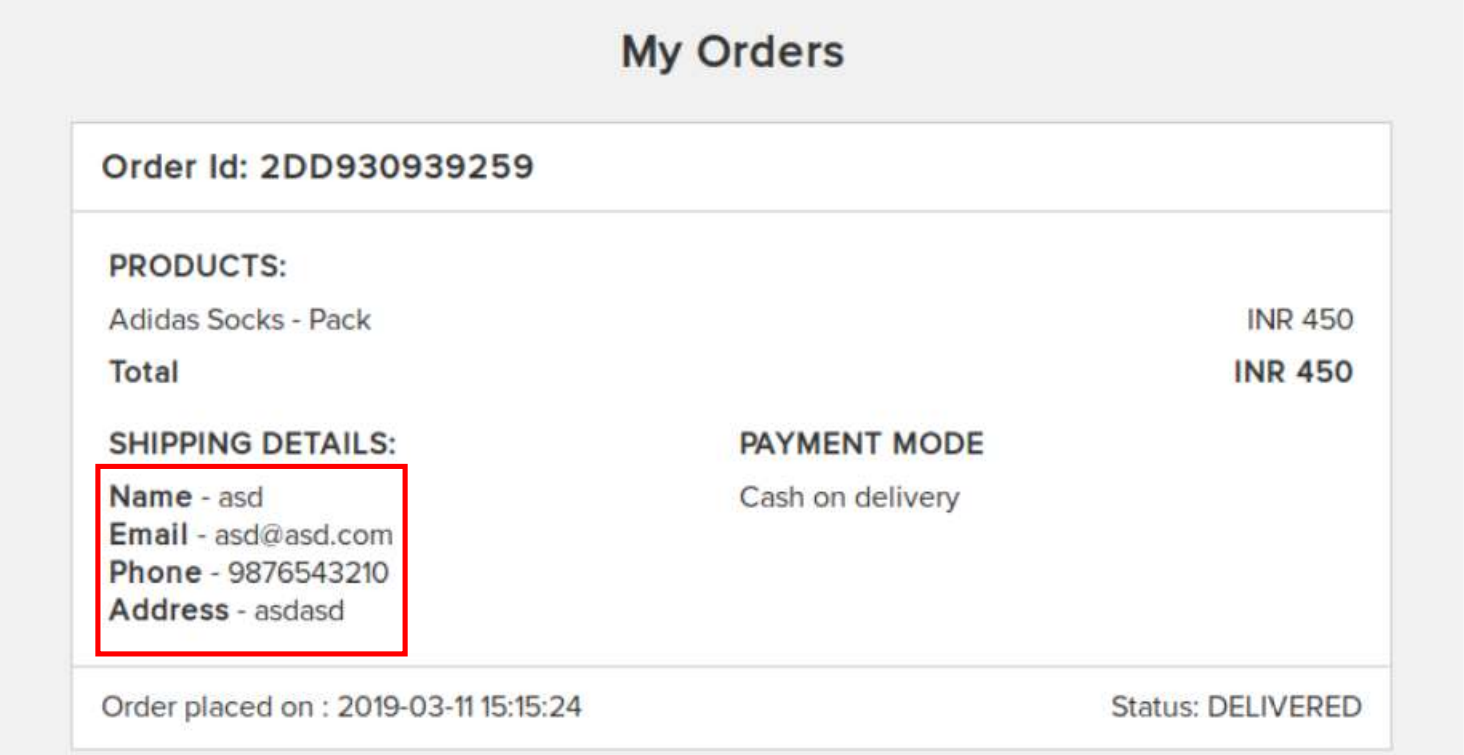
These are the order details of customer number 14.

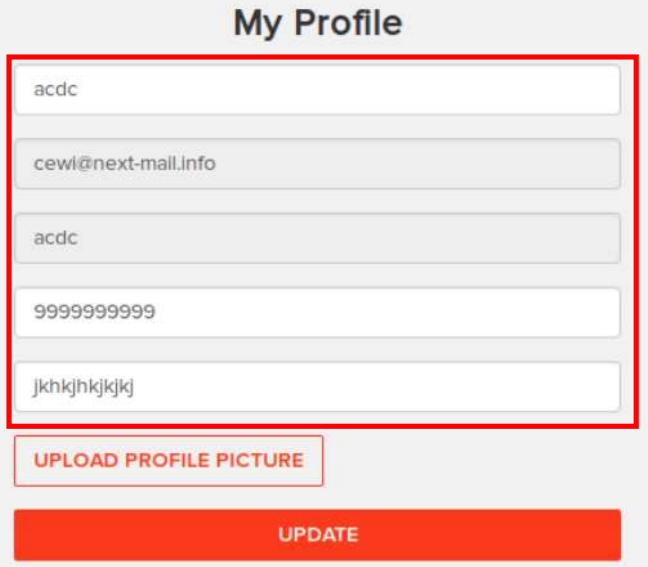


Just by changing the value of u (GET Parameter) from /user/profile/3 to /user/profile/2 or /user/profile/1.
We can easily get access to the accounts of admin and guest.

Business Impact : Extremely High

This vulnerability when exploited yields the personal details of a user (name, username, the phone number, address, email address), these details can be used to carry out other attacks which can jeopardize the existence of the user in all ways possible. If a user's information is exploited in anyway he/she can file a legal lawsuit against the company charging hefty amounts in compensation. This will cause major defamation and heavy losses for the company.





Recommendation

- Implement proper authentication and authorisation checks to make sure that the user has permission to the data he/she is requesting
- Use proper rate limiting checks on the number of request comes from a single user in a small amount of time
- Make sure each user can only access their own data.
- The application should verify that the current user has permission to access the object, every time access is attempted. If a user has access to record 3 and no others, any attempt to access record 4 should be rejected. Note that in some cases, just checking on read access is insufficient: if they can write to a record, even without being able to read it, it may allow further compromise. For example, if a PUT request made to /user/1 contains a password value, it is important to check that the same PUT request cannot be made to /user/2, even if it is known that GET /user/2 is rejected.
- Secondly, object references can be modified to be user-specific. This can prevent exposing information which can be commercially sensitive: if your client id field is 1045, and you signed up to a site yesterday, you can make a pretty reasonable guess that there are unlikely to be more than 1050 clients. This could be exposed through profile screens at paths such as /profile/1045/edit, or through a parameter being passed in a POST request, among other ways. If, however, the path was /profile/edit, and the client id was determined by the server based on the currently active user session, this information is not exposed.

References

- *https://www.owasp.org/index.php/Insecure_Configuration_Management*
- *https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References*

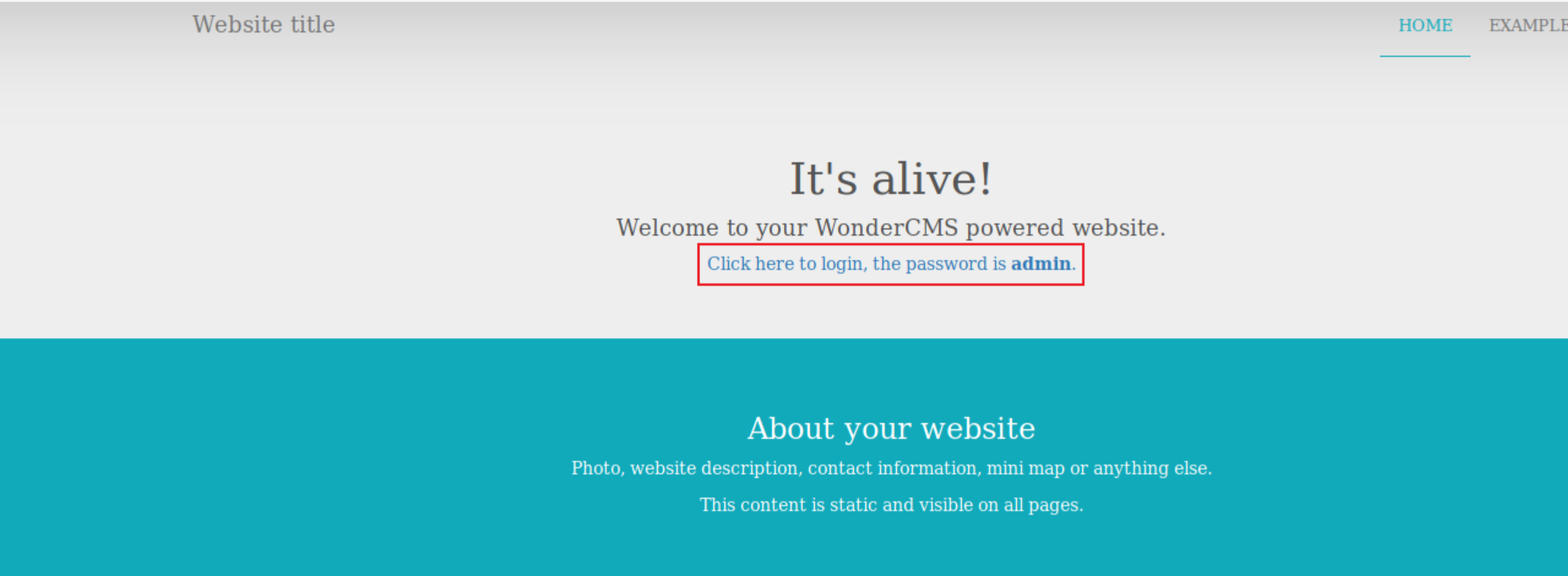
3. Default or Weak Password

Information Disclosure is when a webpage or web application fails to protect information and is visible in plain sight on the platform. When we navigate to the blog page we come across only a field which asks for a password and the password “admin” is visible on the page itself , giving direct access to the admin pane, which open up new array of vulnerabilities jeopardizing the whole website.

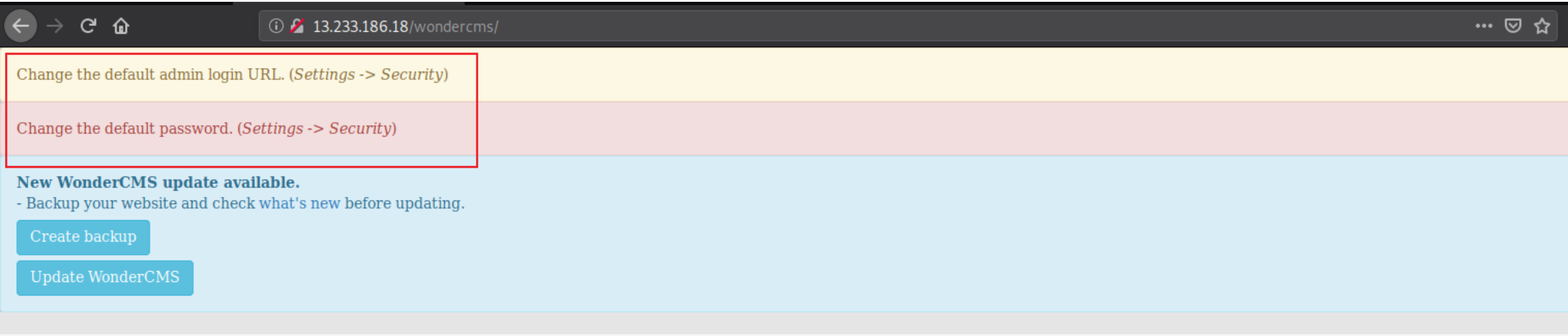
	Affected URL	Parameters	Payload
Default or Weak Password (critical)	http://13.233.186.18/wondercms/loginURL	password	admin
	http://13.233.186.18/login/seller.php	username, password	“sellernamewith123”

Observation

When we click the blog tab, we are directed to “URL/wondercms” in which the password to login to the admin panel is visible.

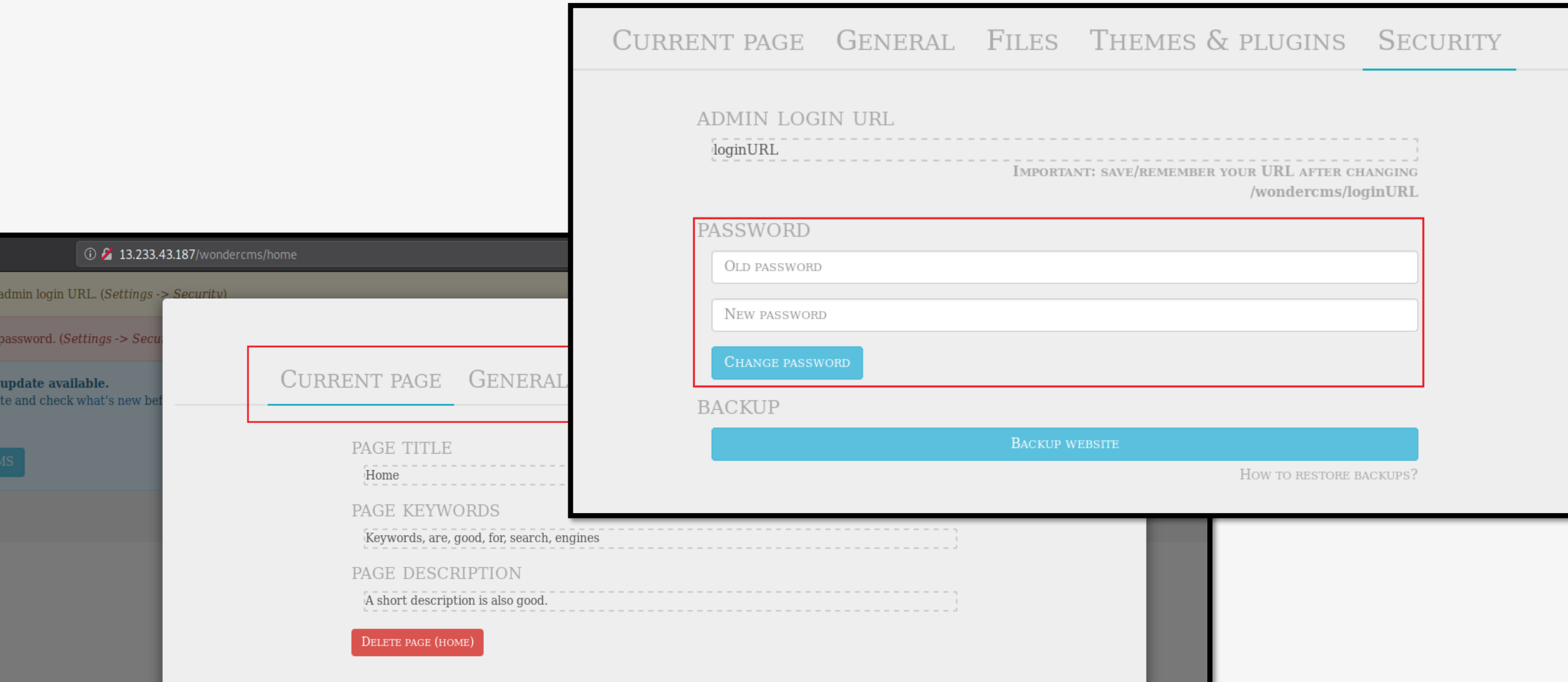


After logging in there are clear instructions on the page to change the default password and other security settings which will give any user who logs into the admin panel all the privileges of the admin.

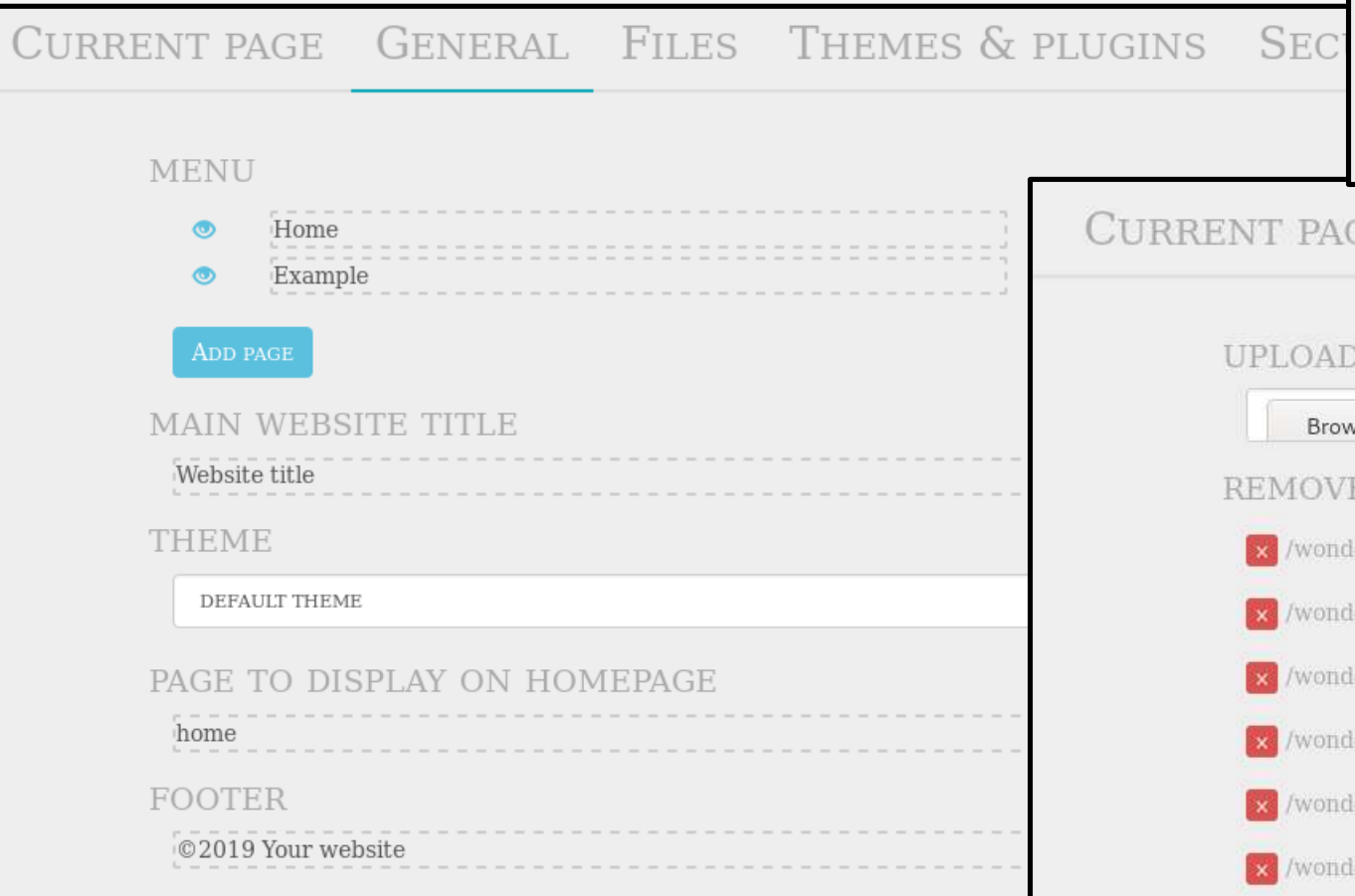


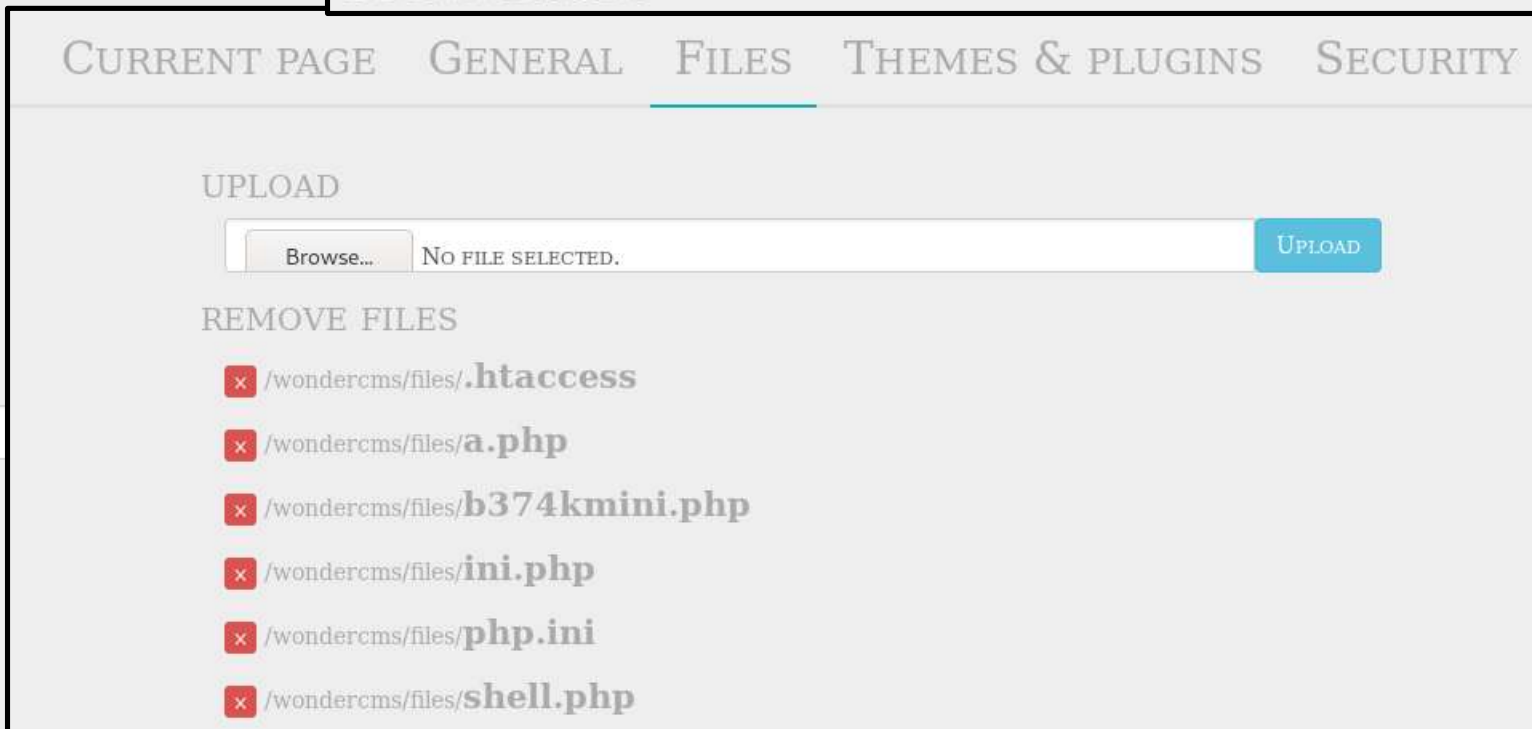
Proof of Concept (PoC)

We can clearly observe, settings such as editing, deleting of files or pages is accessible and installing or deleting themes & plugins can also be done now.



As we can see the options to change the themes & plugins are present and hence a vulnerable theme or plugin can be installed on the system and later exploited. In the ‘general’ tab a page can be added with some malicious code and exploited and also any arbitrary file can be uploaded via the files tab and take full control over the entire website.





Not secure 13.126.119.22/seller/dashboard.php

Progress Report | In...

Lifestyle Store

Dashboard Logout

This page is under construction

Business Impact : Extremely High

- Using this vulnerability, the attacker can get full access to the blog of the website with admin privileges.
- The attacker can change the password or even change the URL of the admin panel and restrict the admin to access it.
- The attacker can create new pages, delete any existing pages on the subdomain and edit these pages also.
- Files can be uploaded(without verification) and hence can be of malicious kind. So, files which can cause harm to the server can be uploaded.

Depending on the nature of the password-protected resource, an attacker can mount one or more of the following types of attacks:

- Access the contents of the password-protected resources.
- Access password-protected administrative mechanisms such as "dashboard", "management console" and "admin panel," potentially progressing to gain full control of the application.

Recommendation

- The default password should be changed and a strong password must be setup.
- The admin URL must never be accessible to normal users.
- Password must be at least 8 characters long containing numbers, alphanumerics, etc.
- All the default accounts should be hidden or removed better inaccessible.
- Password should not be reused.
- Do not use weak passwords, which are short, default, common or easy to guess. Implement a strong password policy.

References

https://www.owasp.org/index.php/Testing_for_weak_password_change_or_reset_functionalities_(OTG-AUTHN-009)
https://www.owasp.org/index.php/Default_Passwords
https://www.us-cert.gov/ncas/alerts/TA13-175A

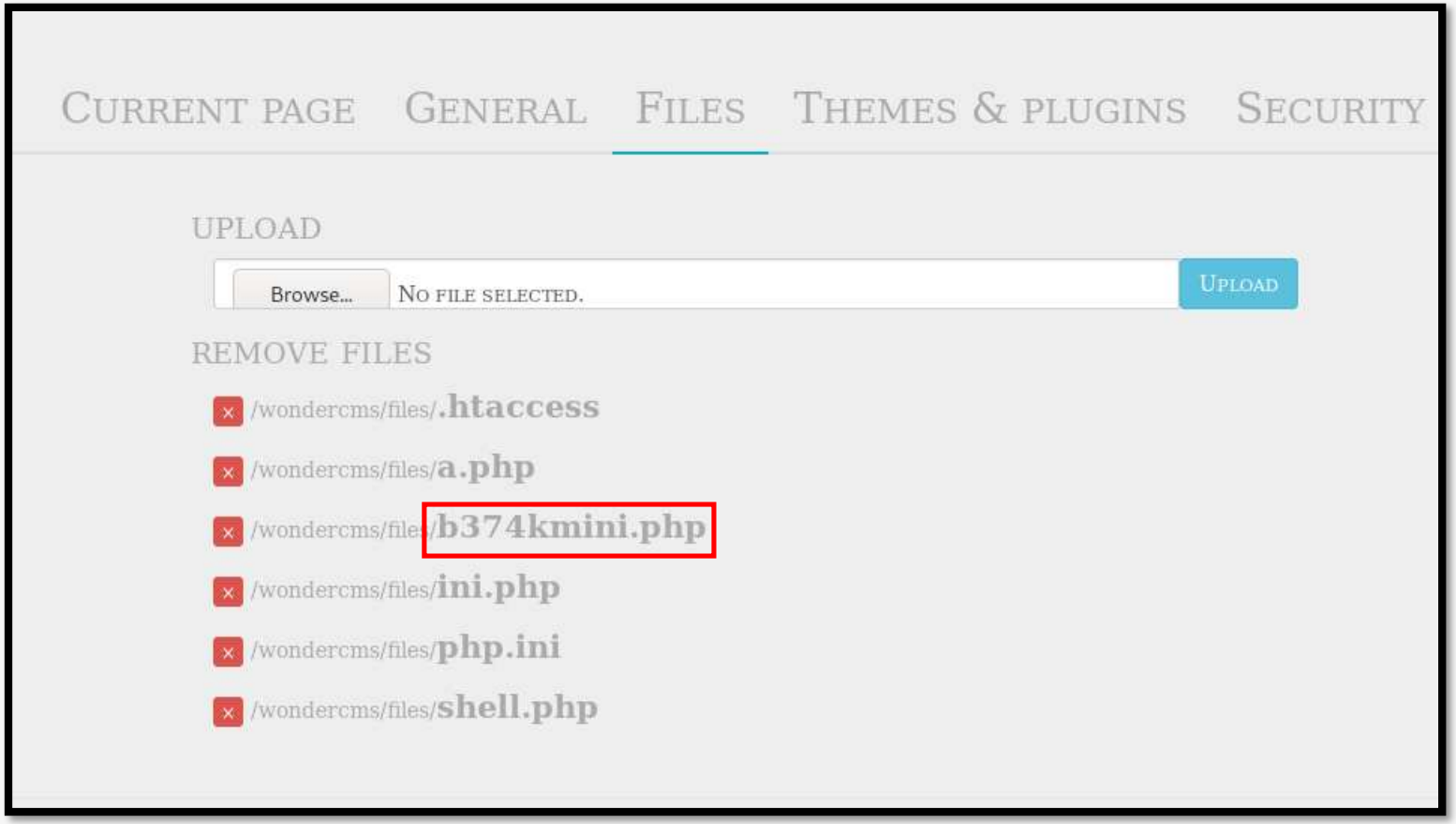
4. Arbitrary File Inclusion

Arbitrary File Inclusion vulnerability facilitates the upload of a malicious file which can compromise the well being of a website. In this case a PHP shell was successfully uploaded into the server and it opened a variety of functionalities and privileges which compromised the entire blog section.

Default or Weak Password (critical)	Affected URL	Parameters	Payload
	http://13.233.186.18/wondercms	File Upload Option in the CMS.	B374kmini.php(php shell file)

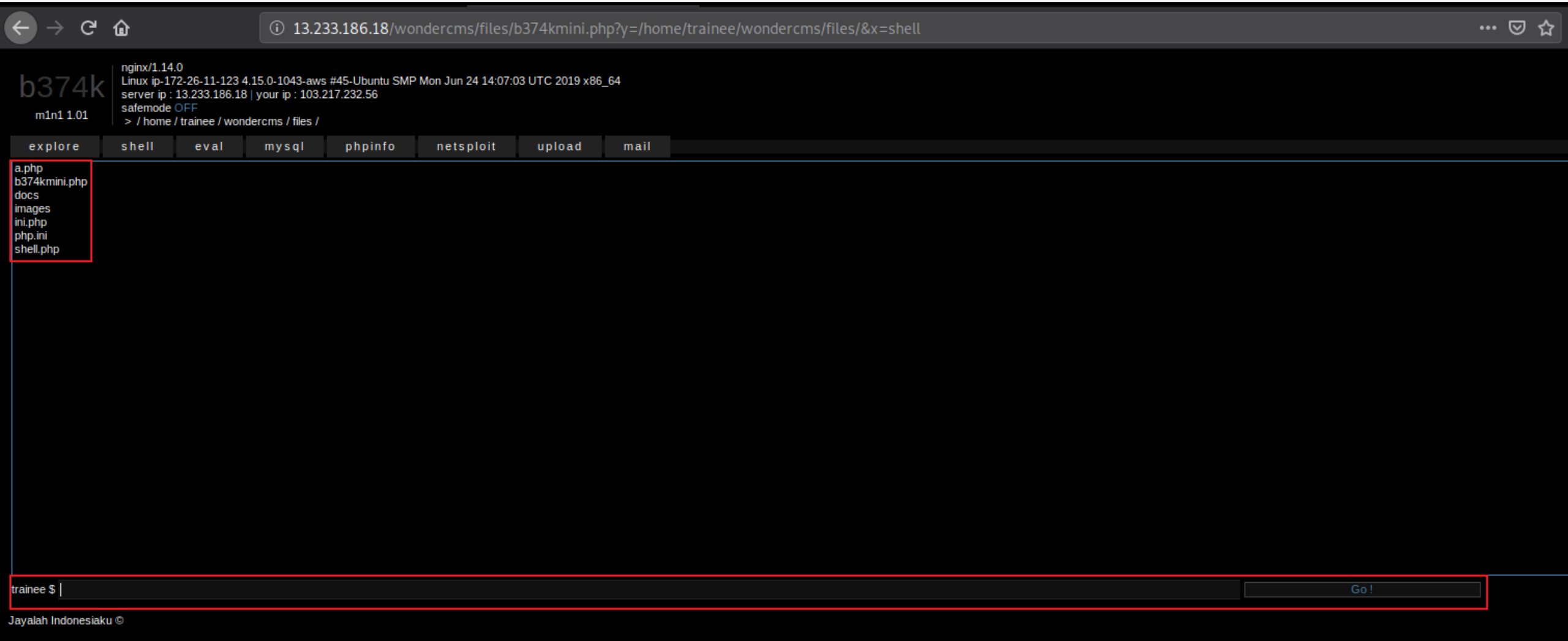
Observation

No file type filter was encountered during file upload. Hence a file of any type could be uploaded. A PHP shell was uploaded named “b374kmini.php” which gave privileges which can give complete access to the attacker remotely.



Proof of Concept (PoC)

Using the PHP shell , an onsite console could be exploited and all on server files can be accessed. The configuration file phpinfo() can also be accessed, MySQL commands can be executed or logged into. It also opens up a way to carry out OS Command Injection on the server. Basically the Operating System being Ubuntu; so only Linux based commands can be used.



Not secure 35.154.192.226/wondercms/home

Progress Report | In...

Change the default admin login URL. (*Settings -> Security*)

Change the default password. (*Settings -> Security*)

New WonderCMS update available.
– Backup your website and check what's new before updating.

Create backup

Update WonderCMS

File uploaded.

SETTINGS LOGOUT

Website title

HOME EXAMPLE

It's alive!

Welcome to your WonderCMS powered website.

Click here to login, the password is **admin**.

About your website

Business Impact : Extremely High

An attacker might alter or corrupt a database, steal customer records, use an API to launch a specific process or event, or launch a distributed denial of service (DDoS) attack. Once attackers have gained control of a server, they can use the underlying application to exploit any and all capabilities built into the software. The resulting damage is determined by the user authorizations and security protections an organization has in place. Attackers may even retain access to systems even after an organization has detected and fixed the underlying vulnerability.

It is important to check a file upload module's access controls to examine the risks properly. Uploaded files can be abused to exploit other vulnerable sections of an application when a file on the same or a trusted server is needed (can again lead to client-side or server-side attacks)

Uploaded files might trigger vulnerabilities in broken libraries/applications on the client side (e.g. iPhone MobileSafari LibTIFF Buffer Overflow), server side (e.g. ImageMagick flaw that called ImageTragick!).

Uploaded files might trigger vulnerabilities in broken real-time monitoring tools (e.g. Symantec antivirus exploit by unpacking a RAR file)

A malicious file such as a Unix shell script, a windows virus, an Excel file with a dangerous formula, or a reverse shell can be uploaded on the server in order to execute code by an administrator or webmaster later – on the victim's machine.

An attacker might be able to put a phishing page into the website or deface the website.

The file storage server might be abused to host troublesome files including malwares, illegal software, or adult contents. Uploaded files might also contain malwares' command and control data, violence and harassment messages, or steganographic data that can be used by criminal organisations.

Uploaded sensitive files might be accessible by unauthorised people.

File uploaders may disclose internal information such as server internal paths in their error messages.

Recommendation

The file types allowed to be uploaded should be restricted to only those that are necessary for business functionality.
Never accept a filename and its extension directly without having a whitelist filter.
The application should perform filtering and content checking on any files which are uploaded to the server. Files should be thoroughly scanned and validated before being made available to other users. If in doubt, the file should be discarded.
It is necessary to have a list of only permitted extensions on the web application. And, file extension can be selected from the list. For instance, it can be a “select case” syntax (in case of having VBScript) to choose the file extension in regards to the real file extension.
All the control characters and Unicode ones should be removed from the filenames and their extensions without any exception. Also, the special characters such as “;”, “:”, “>”, “<”, “/” ,”\”, additional “.”, “*”, “%”, “$”, and so on should be discarded as well.
If it is applicable and there is no need to have Unicode characters, it is highly recommended to only accept Alpha-Numeric characters and only 1 dot as an input for the file name and the extension; in which the file name and also the extension should not be empty at all (regular expression: [a-zA-Z0-9]{1,200}\.[a-zA-Z0-9]{1,10}).
Limit the filename length. For instance, the maximum length of the name of a file plus its extension should be less than 255 characters (without any directory) in an NTFS partition.
It is recommended to use an algorithm to determine the filenames. For instance, a filename can be a MD5 hash of the name of file plus the date of the day.
Uploaded directory should not have any “execute” permission and all the script handlers should be removed from these directories.
Limit the file size to a maximum value in order to prevent denial of service attacks (on file space or other web application’s functions such as the image resizer).
Restrict small size files as they can lead to denial of service attacks. So, the minimum size of files should be considered.
Use Cross Site Request Forgery protection methods.

Prevent from overwriting a file in case of having the same hash for both.
Use a virus scanner on the server (if it is applicable). Or, if the contents of files are not confidential, a free virus scanner website can be used. In this case, file should be stored with a random name and without any extension on the server first, and after the virus checking (uploading to a free virus scanner website and getting back the result), it can be renamed to its specific name and extension.
Try to use POST method instead of PUT (or GET!)
Log users' activities. However, the logging mechanism should be secured against log forgery and code injection itself.
In case of having compressed file extract functions, contents of the compressed file should be checked one by one as a new file.
If it is possible, consider saving the files in a database rather than on the filesystem.
If files should be saved in a filesystem, consider using an isolated server with a different domain to serve the uploaded files.
File uploaders should be only accessible to authenticated and authorised users if possible.
Write permission should be removed from files and folders other than the upload folders. The upload folders should not serve any
Ensure that configuration files such as ".htaccess" or "web.config" cannot be replaced using file uploaders. Ensure that appropriate settings are available to ignore the ".htaccess" or "web.config" files if uploaded in the upload directories.
Ensure that files with double extensions (e.g. "file.php.txt") cannot be executed especially in Apache.
Ensure that uploaded files cannot be accessed by unauthorised users.
Adding the "Content-Disposition: Attachment" and "X-Content-Type-Options: nosniff" headers to the response of static files will secure the website against Flash or PDF-based cross-site content-hijacking attacks. It is recommended that this practice be performed for all of the files that users need to download in all the modules that deal with a file download. Although this method does not fully secure the website against attacks using Silverlight or similar objects, it can mitigate the risk of using Adobe Flash and PDF objects, especially when uploading PDF files is permitted.
Flash/PDF (crossdomain.xml) or Silverlight (clientaccesspolicy.xml) cross-domain policy files should be removed if they are not in use and there is no business requirement for Flash or Silverlight applications to communicate with the website.

Browser caching should be disabled for the corssdomain.xml and clientaccesspolicy.xml files. This enables the website to easily update the file or restrict access to the Web services if necessary. Once the client access policy file is checked, it remains in effect for the browser session so the impact of non-caching to the end-user is minimal. This can be raised as a low or informational risk issue based on the content of the target website and security and complexity of the policy file(s).
CORS headers should be reviewed to only be enabled for static or publicly accessible data. Otherwise, the “Access-Control-Allow-Origin” header should only contain authorised addresses. Other CORS headers such as “Access-Control-Allow-Credentials” should only be used when they are required. Items within the CORS headers such as “Access-Control-Allow-Methods” or “Access-Control-Allow-Headers” should be reviewed and removed if they are not required.

References

https://portswigger.net/web-security/os-command-injection
https://www.netsparker.com/blog/web-security/command-injection-vulnerability/

Vulnerability Classification: CWE-434

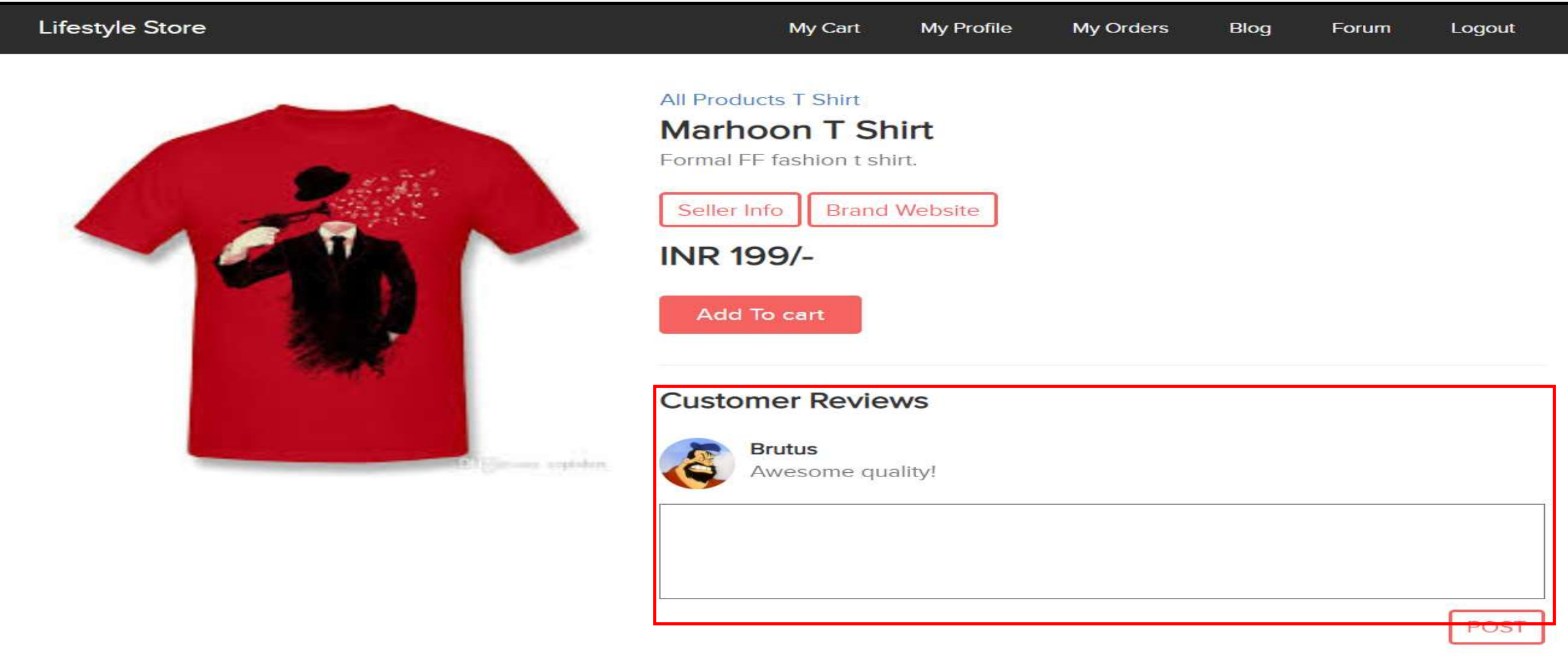
5. Reflected and Stored Cross Site Scripting(XSS)

Reflected cross-site scripting vulnerabilities arise when data is copied from a request and echoed into the application's immediate response in an unsafe way. An attacker can use the vulnerability to construct a request that, if issued by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application.

	Affected URL	Parameters	Payload
Reflected and Stored Cross Site Scripting (critical)	http://13.233.164.7/search/search.php (reflected)	q(GET Parameter)	abcd"><script>alert(1)</script>efgh
	http://13.233.164.7/profile/2/edit/ (stored)	form-data; name="address“ (POST Parameter)	<script>alert(1)</script>
	http://13.233.164.7/products/details.php?p_id=15* (all products having reviews/comment option) (stored)	body;comment=“” (POST Parameter)	<script>alert(1)</script>
	http://13.233.164.7/wondercms (title of the page, in edit current page option)	Page title	<script>alert(1)</script>

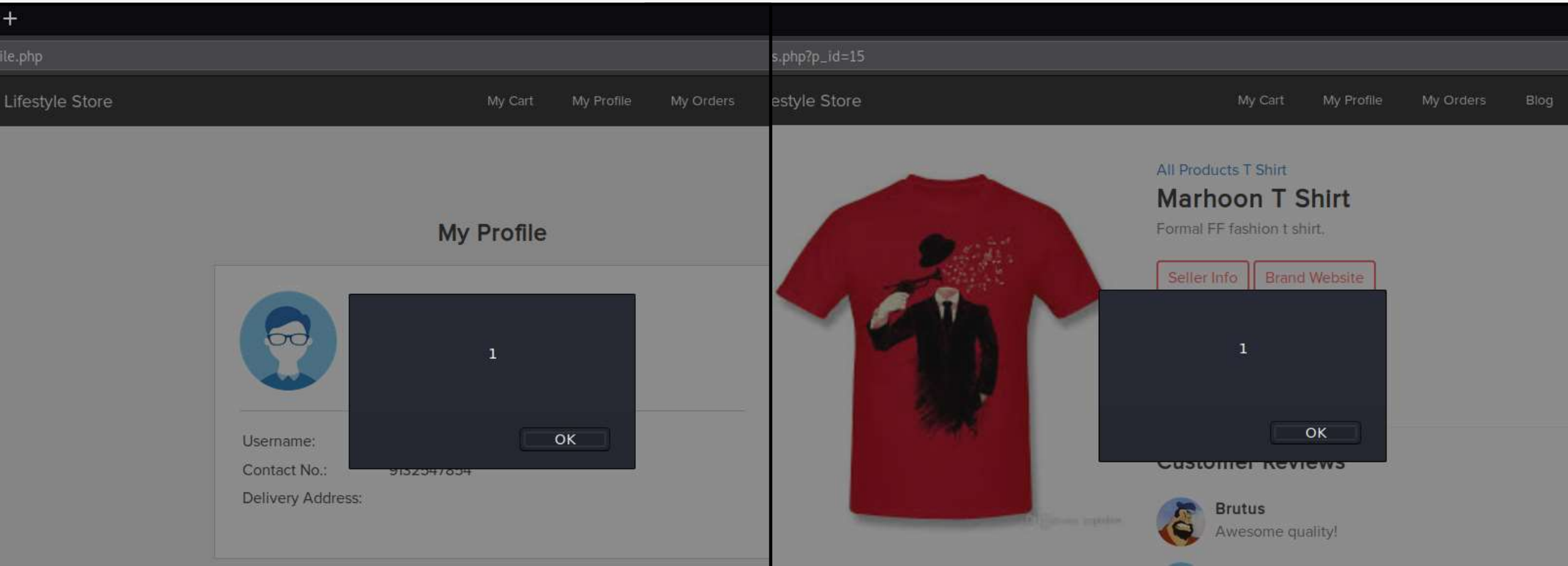
Observation

The space provided for the customer reviews executes HTML, CSS and JS commands. The attacker can execute Stored Cross Site Scripting in this area which will seem as though it is a part of the website. The user who trusts this website would also trust thinking that this is a part of the website, hence falling for the attack.

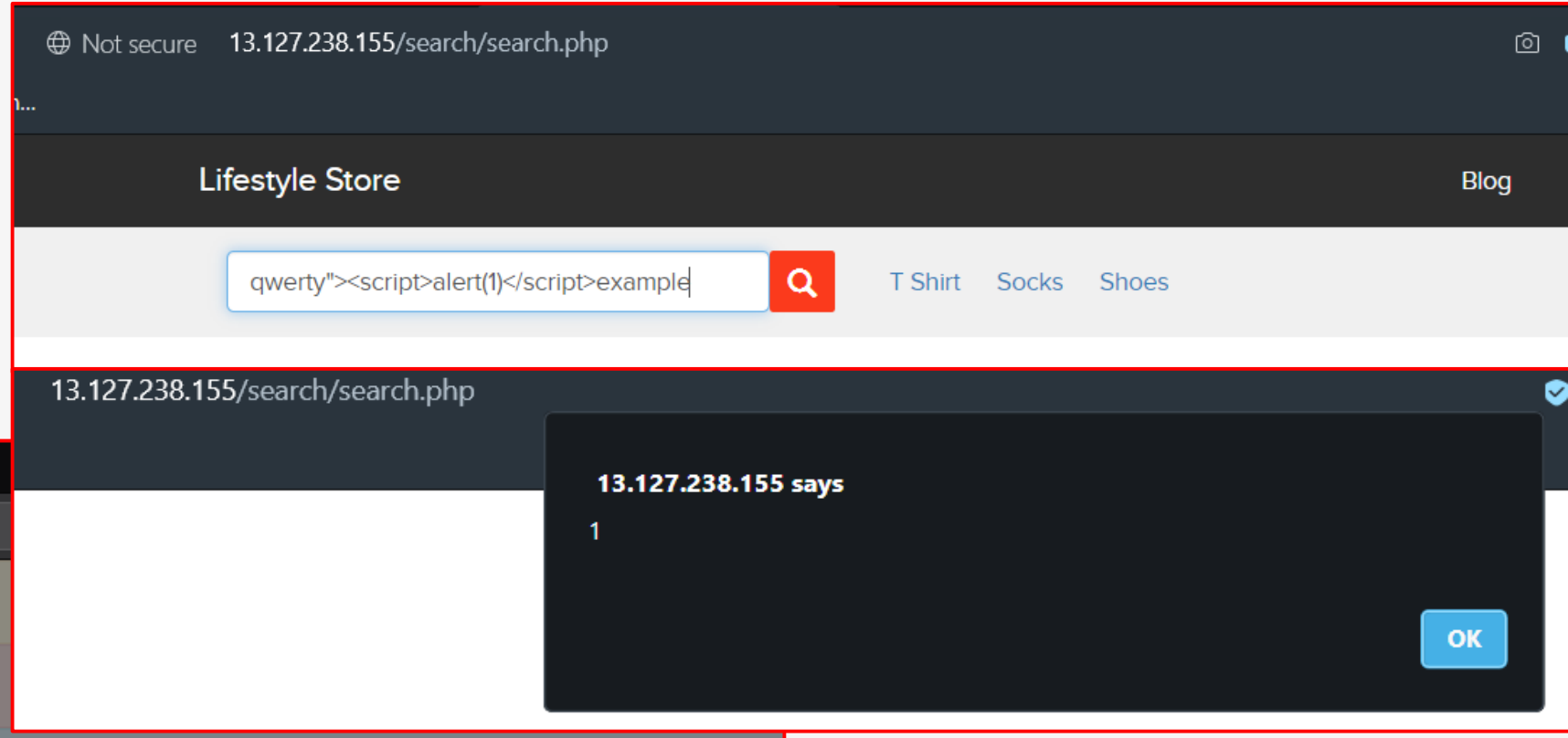


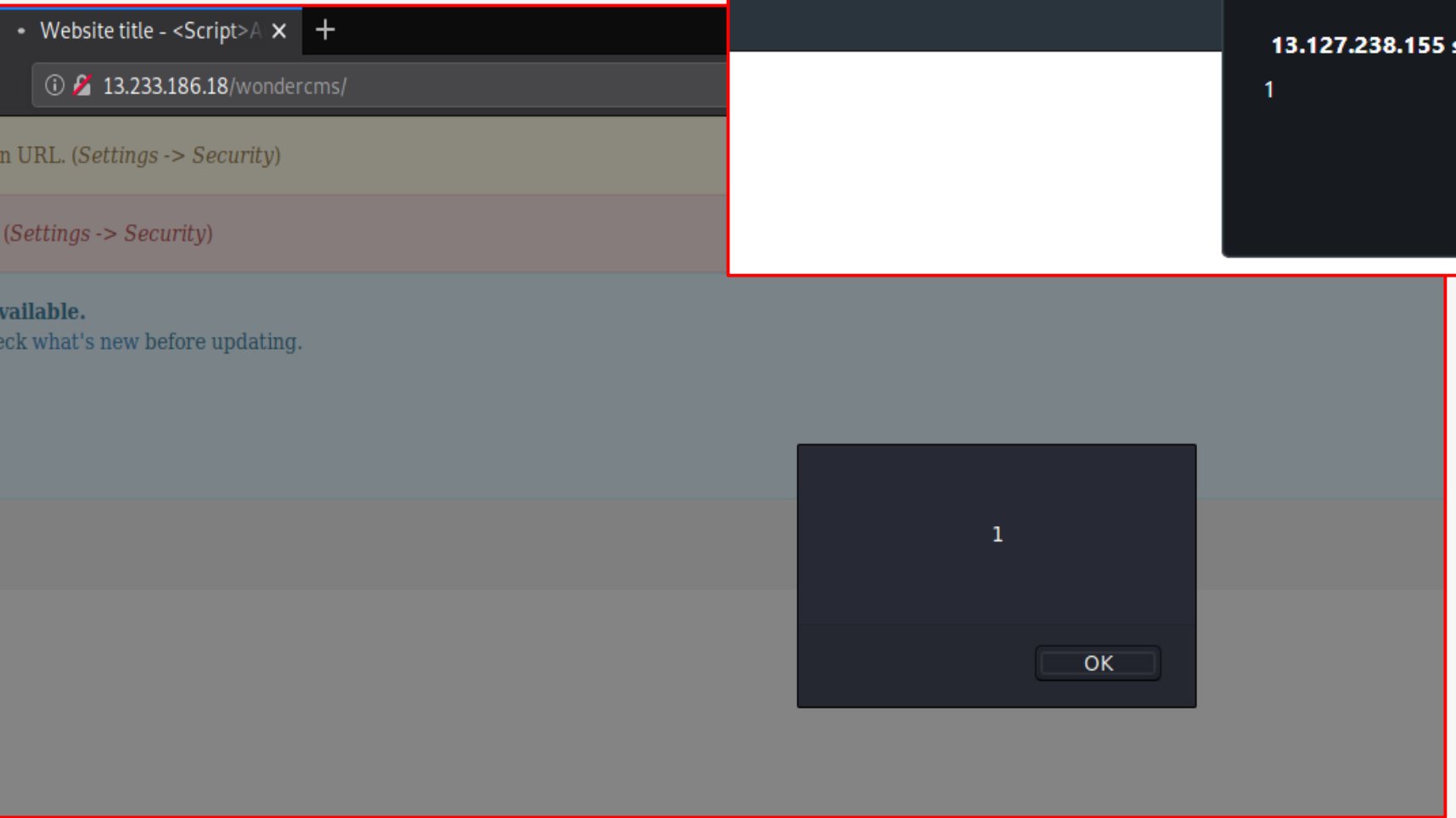
Proof of Concept (PoC)

Here are the two pop ups that we generated for Stored Cross Site Scripting which can direct to a malicious site. These can be removed only by the user and always generate when the item is accessed and when someone will edit their profile (only possible if we can bypass the credentials of a user which we have). These Stored XSS can be planted in the review/comment section of an item and in the address section of the edit profile option.



The edit page title option lets us put the same payload and gives us the pop up alert.
We can put the same payload <script>alert(1)</script> in the search bar in the main page. That too gives us a pop up. Hence all of these places are affected with Cross Site Scripting.





Even though temporary or reflected XSS is not that harmful but an attacker might find several other ways in addition to this to profit his terms.

Business Impact : Extremely High

The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes.

Users can be induced to issue the attacker's crafted request in various ways. For example, the attacker can send a victim a link containing a malicious URL in an email or instant message. They can submit the link to popular web sites that allow content authoring, for example in blog comments. And they can create an innocuous looking web site that causes anyone viewing it to make arbitrary cross-domain requests to the vulnerable application (using either the GET or the POST method).

The security impact of cross-site scripting vulnerabilities is dependent upon the nature of the vulnerable application, the kinds of data and functionality that it contains, and the other applications that belong to the same domain and organization. If the application is used only to display non-sensitive public content, with no authentication or access control functionality, then a cross-site scripting flaw may be considered low risk. However, if the same application resides on a domain that can access cookies for other more security-critical applications, then the vulnerability could be used to attack those other applications, and so may be considered high risk. Similarly, if the organization that owns the application is a likely target for phishing attacks, then the vulnerability could be leveraged to lend credibility to such attacks, by injecting Trojan functionality into the vulnerable application and exploiting users' trust in the organization in order to capture credentials for other applications that it owns. In many kinds of application, such as those providing online banking functionality, cross-site scripting should always be considered high risk.

Recommendation

In most situations where user-controllable data is copied into application responses, cross-site scripting attacks can be prevented using two layers of defenses:

- Input should be validated as strictly as possible on arrival, given the kind of content that it is expected to contain. For example, personal names should consist of alphabetical and a small range of typographical characters, and be relatively short; a year of birth should consist of exactly four numerals; email addresses should match a well-defined regular expression. Input which fails the validation should be rejected, not sanitized.
- User input should be HTML-encoded at any point where it is copied into application responses. All HTML metacharacters, including < > " ' and =, should be replaced with the corresponding HTML entities (< > etc).

In cases where the application's functionality allows users to author content using a restricted subset of HTML tags and attributes (for example, blog comments which allow limited formatting and linking), it is necessary to parse the supplied HTML to validate that it does not use any dangerous syntax; this is a non-trivial task.

References

https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
https://en.wikipedia.org/wiki/Cross-site_scripting
https://www.w3schools.com/html/html_entities.asp

Vulnerability Classifications: CWE-79, CWE-80, CWE-116, CWE-159

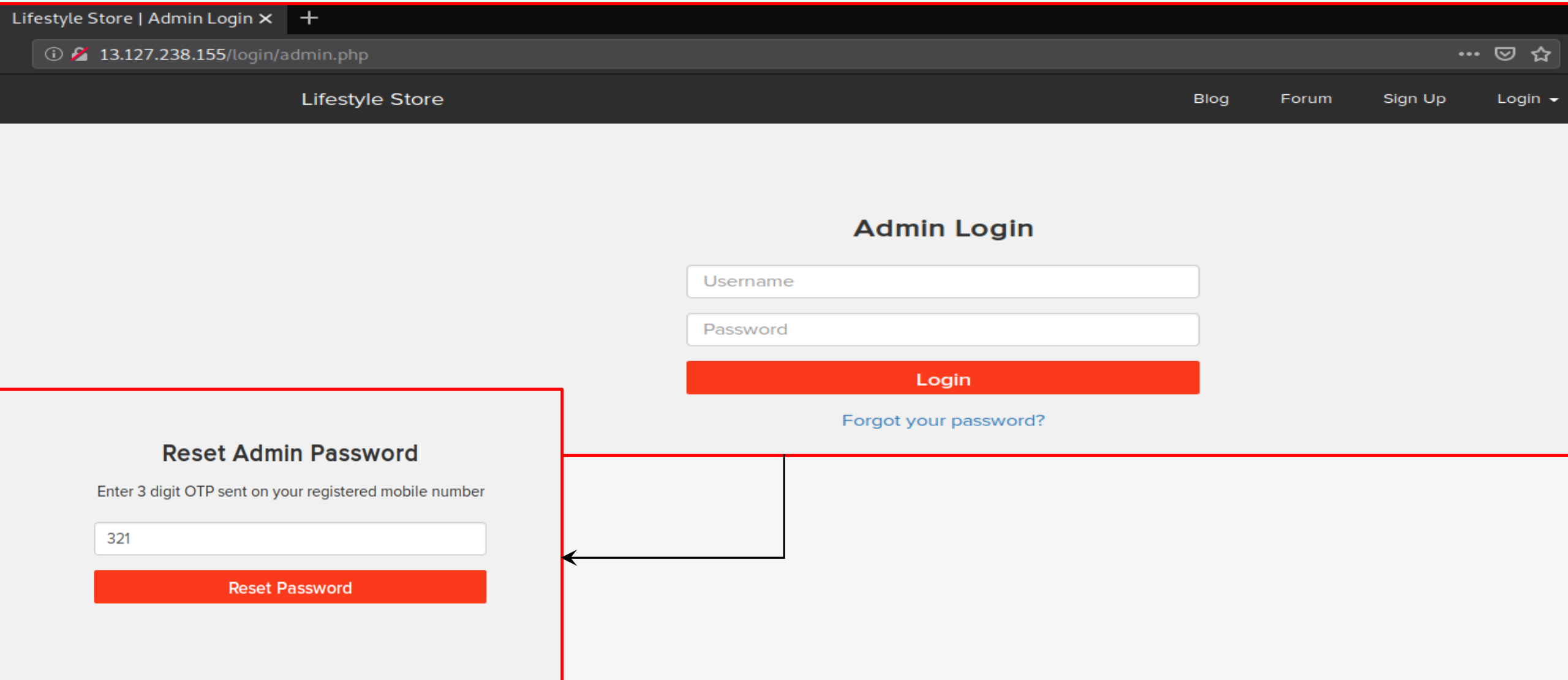
6. Brute force Exploitation

A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response. Considering a given method, number of tries, efficiency of the system which conducts the attack, and estimated efficiency of the system which is attacked the attacker is able to calculate approximately how long it will take to submit all chosen predetermined values.

	Affected URL	Parameters	Payload
Brute force Exploitation (critical)	http://13.233.164.7/cart/apply_coupon.php	coupon=“” (POST parameter)	“UL_1234”
	http://13.233.164.7/reset_password/admin.php?otp=716	otp=“” (GET Parameter)	“716”

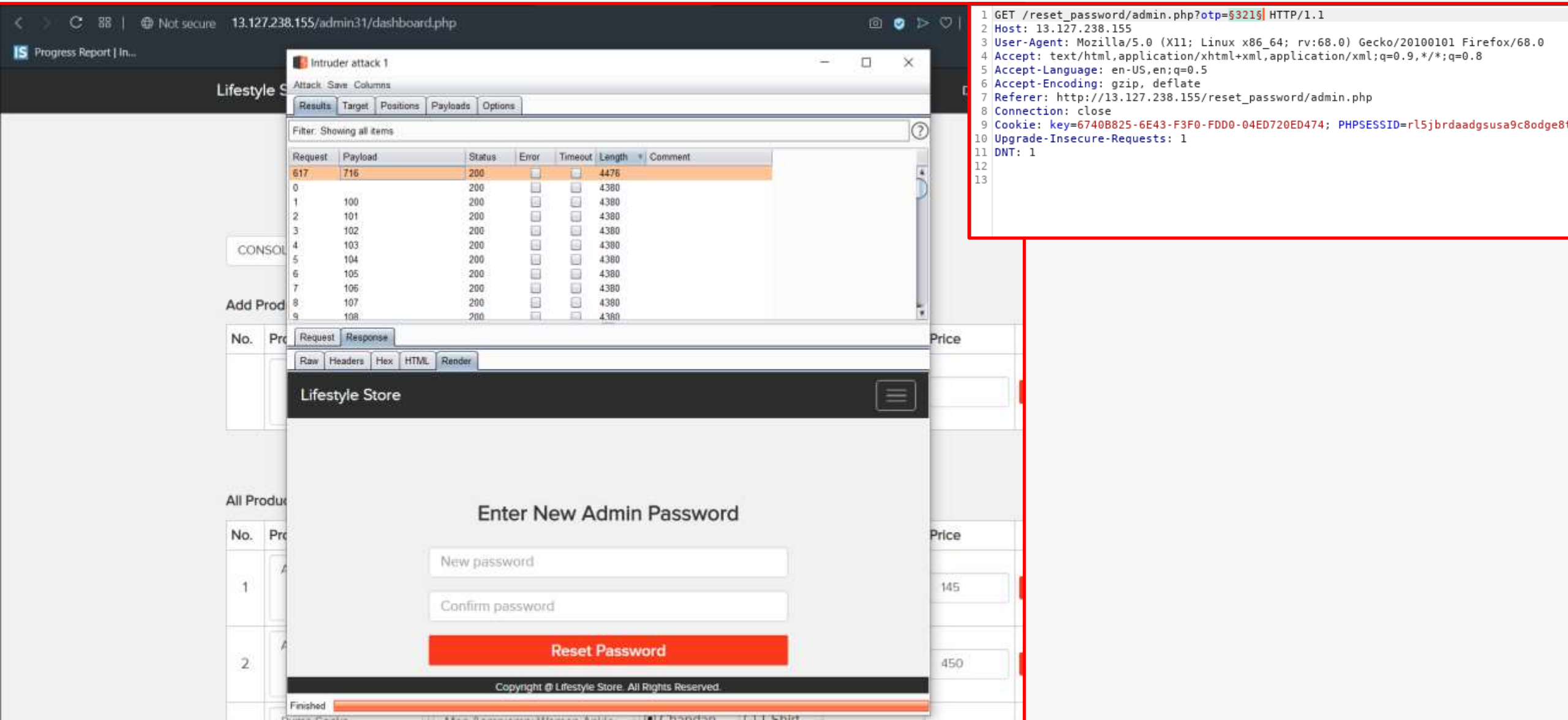
Observation

Here we have a login page for the admin of the website, which also has the forgot password option. By clicking on that, we are taken to the next page where we have to input the otp, which can be bruteforced.

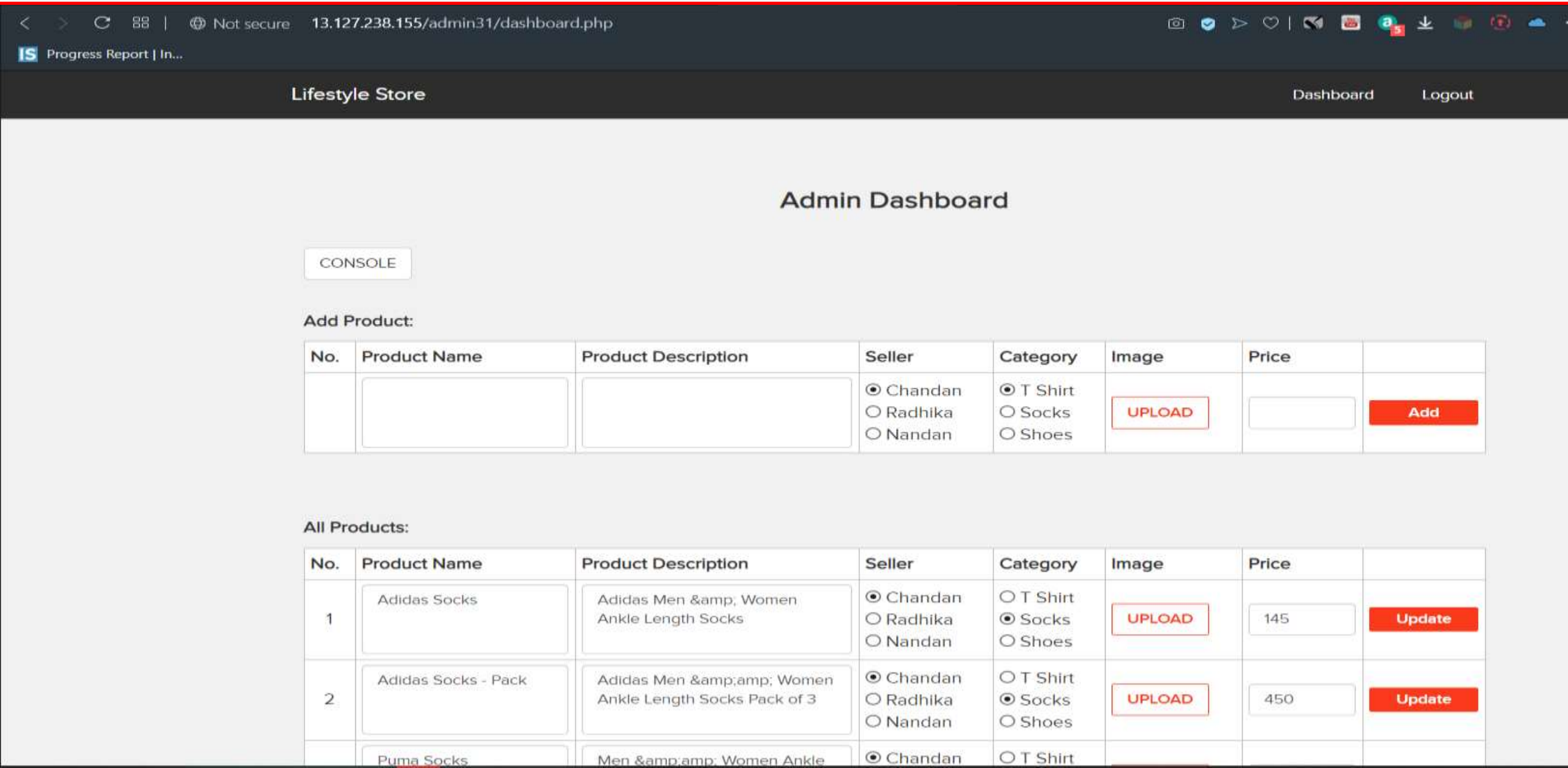


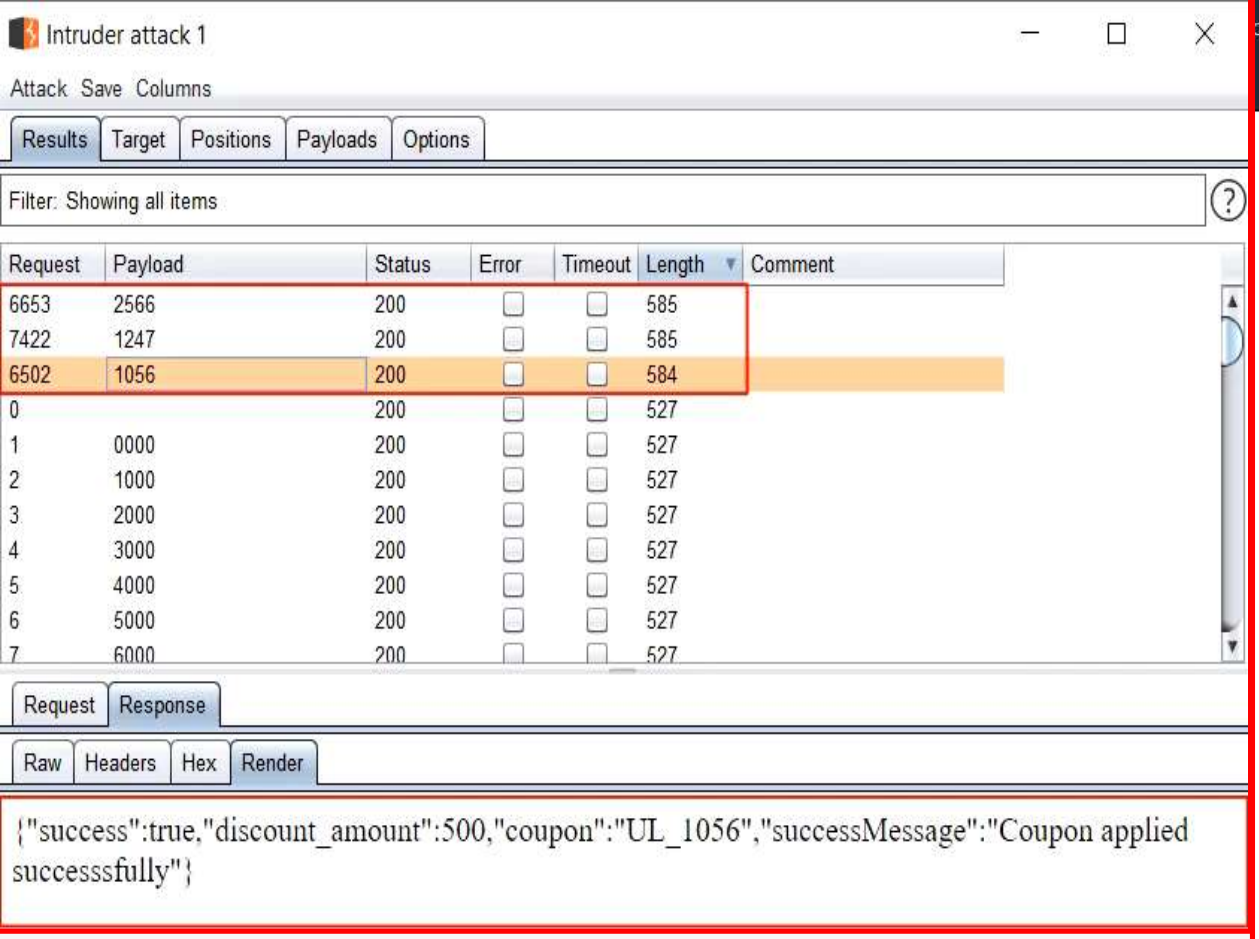
Proof of Concept (PoC)

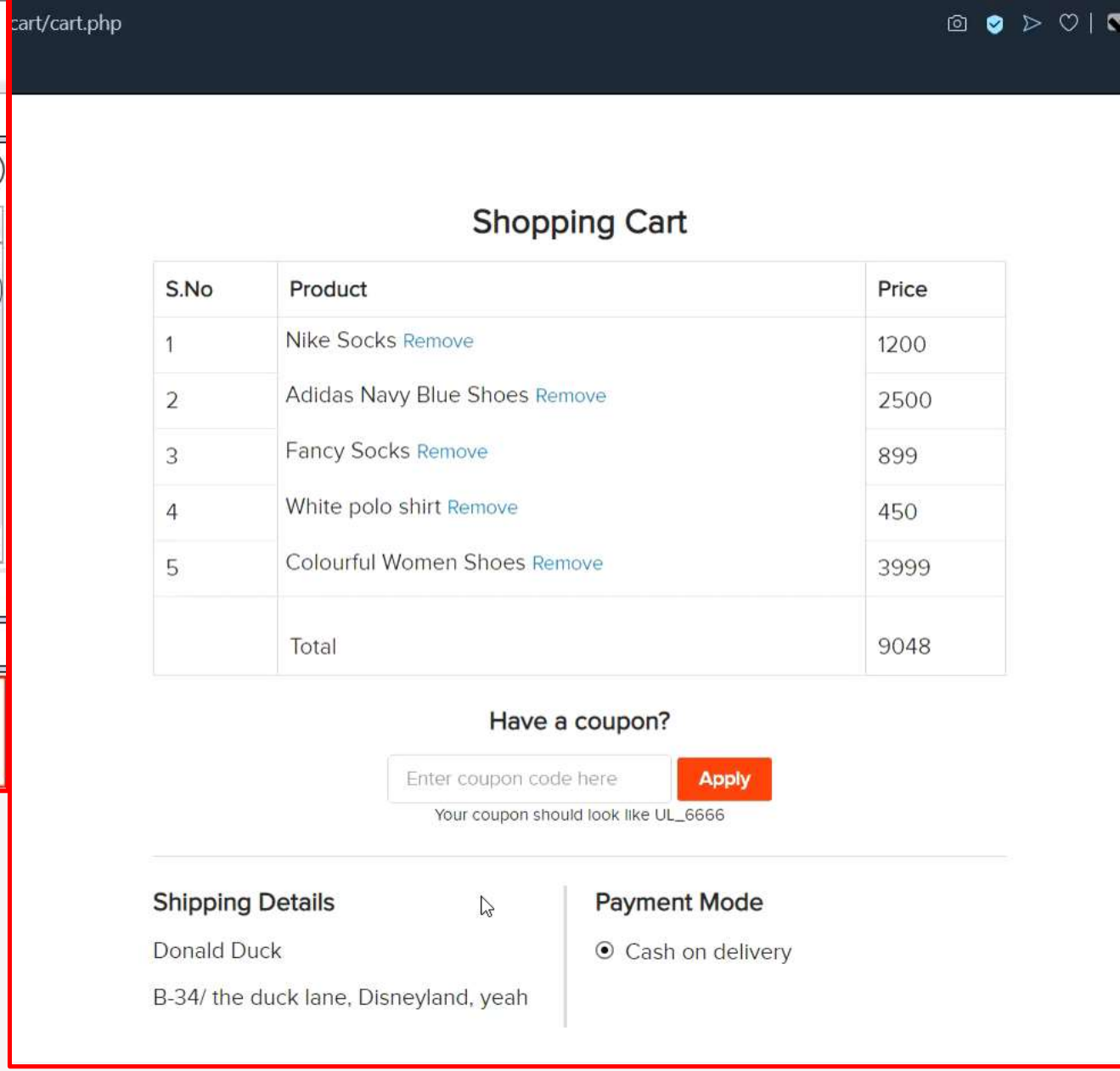
We were able to brute force the OTP and successfully logged into the admin account which gave us a wide variety of privileges.



Adding, categorizing, deleting & editing products, prices and sellers was successfully done.







As also with the coupon in the shopping cart, where we could enter the coupon and get a discount, by brute forcing the coupon serial we discovered three coupons.

The three coupons gave us three different cash discounts. They were 500, 1000 and 5000. These when applied yielded the true amounted discounts.

Business Impact : Extremely High

- Brute-force attacks are often used for attacking authentication and discovering hidden content/pages within a web application. These attacks are usually sent via GET and POST requests to the server. In regards to authentication, brute force attacks are often mounted when an account lockout policy in not in place.
- Attackers have access to hundreds of millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools. Session management attacks are well understood, particularly in relation to unexpired session tokens.
- The prevalence of broken authentication is widespread due to the design and implementation of most identity and access controls. Session management is the bedrock of authentication and access controls, and is present in all stateful applications. Attackers can detect broken authentication using manual means and exploit them using automated tools with password lists and dictionary attacks.
- Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.

Recommendation

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
- Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

References

https://www.hackingarticles.in/brute-force-website-login-page-using-burpsuite/
https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication

Vulnerability Classification: CWE-287, CWE-384

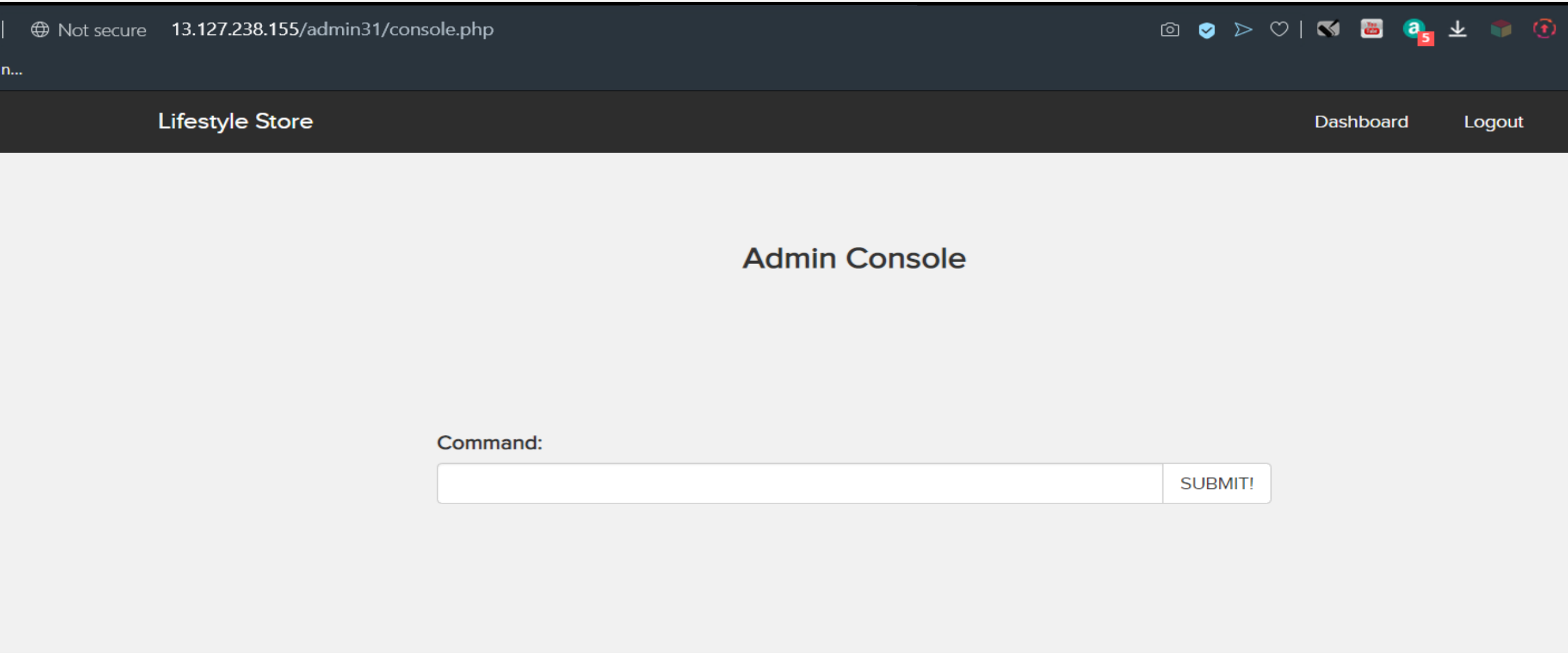
7. Command Execution Vulnerability

OS command injection (also known as shell injection) is a web security vulnerability that allows an attacker to execute arbitrary operating system (OS) commands on the server that is running an application, and typically fully compromise the application and all its data. Very often, an attacker can leverage an OS command injection vulnerability to compromise other parts of the hosting infrastructure, exploiting trust relationships to pivot the attack to other systems within the organization.

	Affected URL	Parameters	Payload
Command Excecution Vulnerability (critical)	http://13.127.238.155/admin31/console	--NA--	--NA--

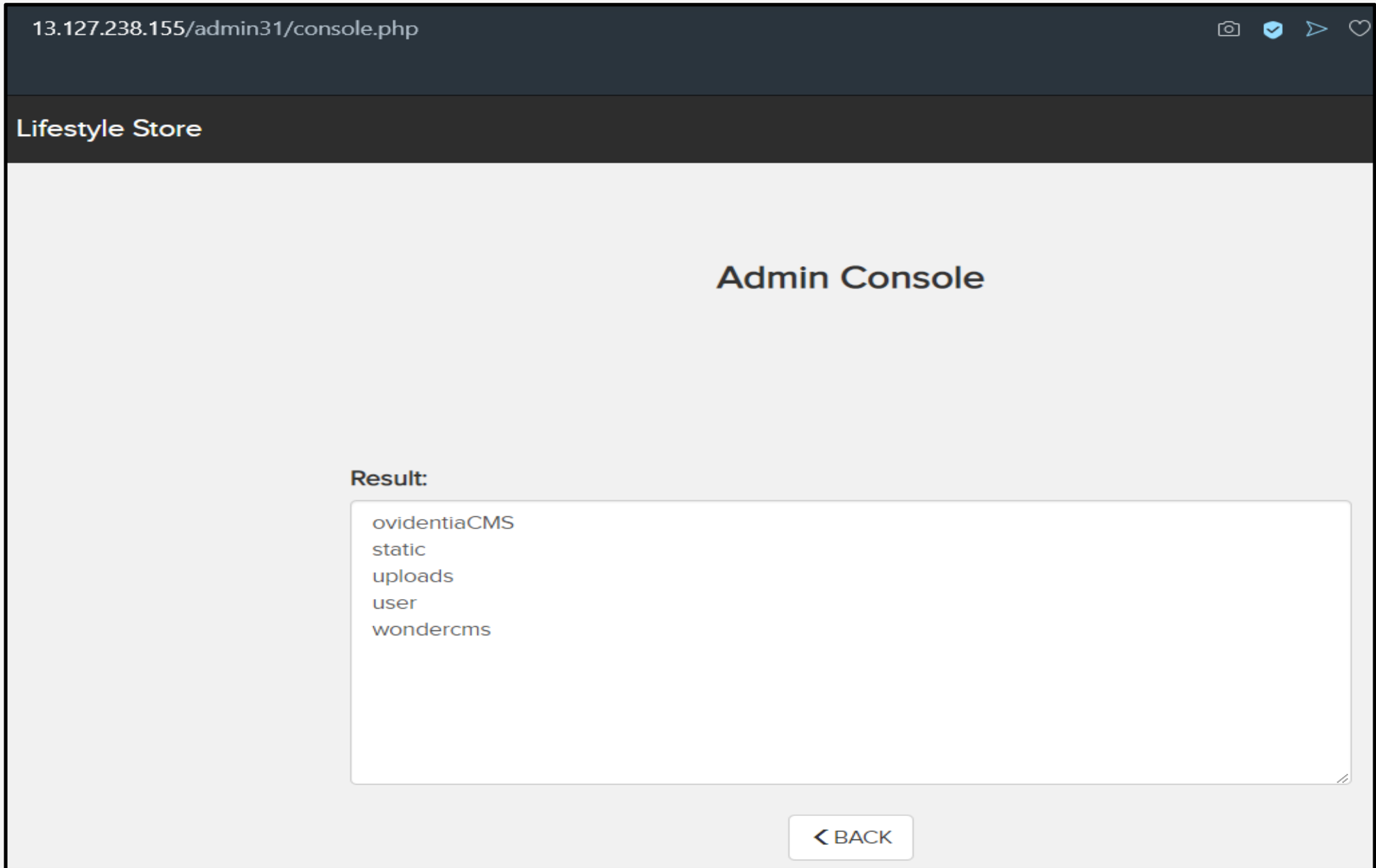
Observation

When we log into the Admin account we can see the first option as console. Upon clicking that button we find ourselves into a console which can execute OS commands (Linux Based).



Proof of Concept (PoC)

By executing the ‘ls’ command we were able to list all the files stored in the folder ’files’ which show us a lot of information.



Business Impact : Extremely High

By exploiting a command injection vulnerability in a vulnerable application, attackers can add additional commands or inject their own operating system commands. This means that during a command injection attack, an attacker can easily take complete control of the host operating system of the web server. Therefore, developers should be very careful how to pass user input into one of those functions when developing web applications. During an OS command injection attack, the attacker can also set up an error based attack.

The impact is very high because the attacker will basically command the whole server and what's worse than that. The server will be under his/her command and the developer probably won't even get a hint of it.

Recommendation

The most effective way to prevent OS command injection vulnerabilities is to never call out to OS commands from application-layer code. In virtually every case, there are alternate ways of implementing the required functionality using safer platform APIs. If it is considered unavoidable to call out to OS commands with user-supplied input, then strong input validation must be performed. Some examples of effective validation include:
Validating against a whitelist of permitted values.
Validating that the input is a number.
Validating that the input contains only alphanumeric characters, no other syntax or whitespace.
Never attempt to sanitize input by escaping shell metacharacters. In practice, this is just too error-prone and vulnerable to being bypassed by a skilled attacker.

References

https://www.netsparker.com/blog/web-security/command-injection-vulnerability/

Vulnerability Classification: CWE-78

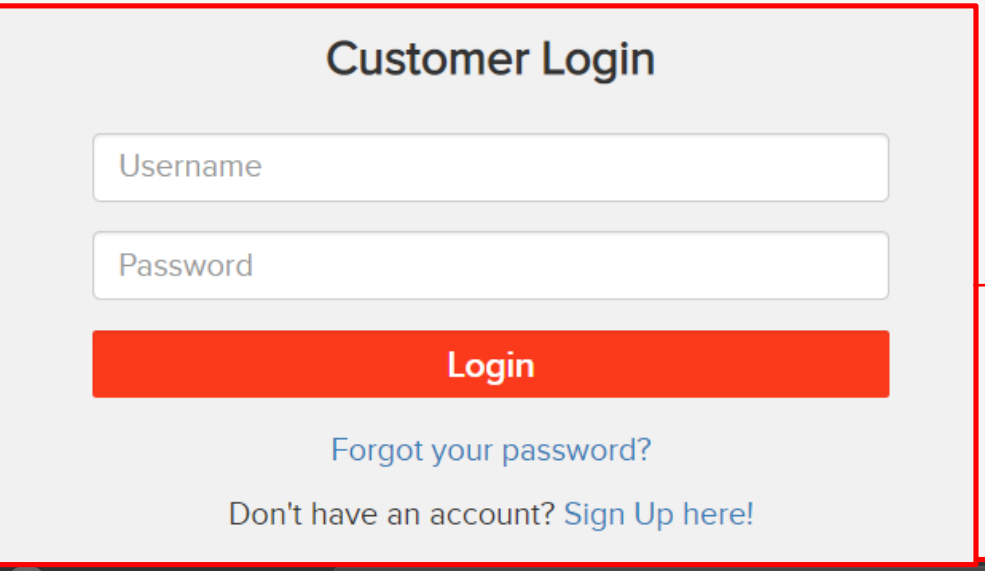
8. Information Disclosure [Source Code Disclosure]

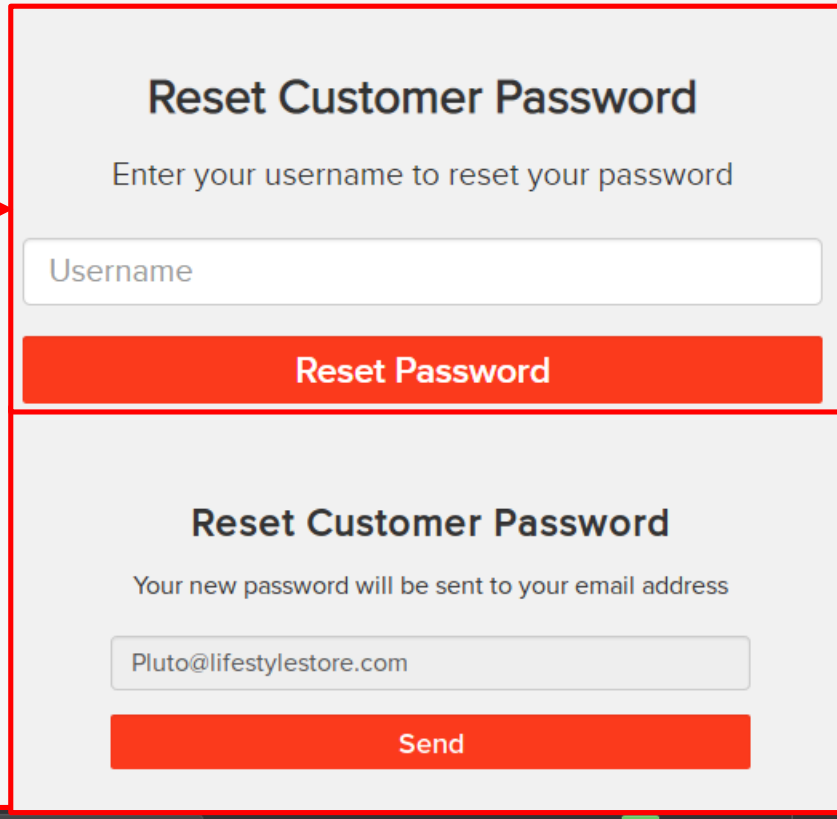
Information disclosure is when an application fails to properly protect sensitive and confidential information from parties that are not supposed to have access to the subject matter in normal circumstances. These type of issues are not exploitable in most cases, but are considered as web application security issues because they allows malicious hackers to gather relevant information which can be used later in the attack lifecycle, in order to achieve more than they could if they didn't get access to such information.

Information Disclosure (severe)	Affected URL	Parameters	Payload
	http://13.127.238.155/reset_password/customer.php?username=""	--NA--	--NA--

Observation

When we try for customer login we find a username and password input form below which we can see “forgot your password” option. Clicking on the options takes us to another form that takes username as input and as the usernames of other customers are visible in the login screen we can just put in the usernames and exploit this vulnerability.



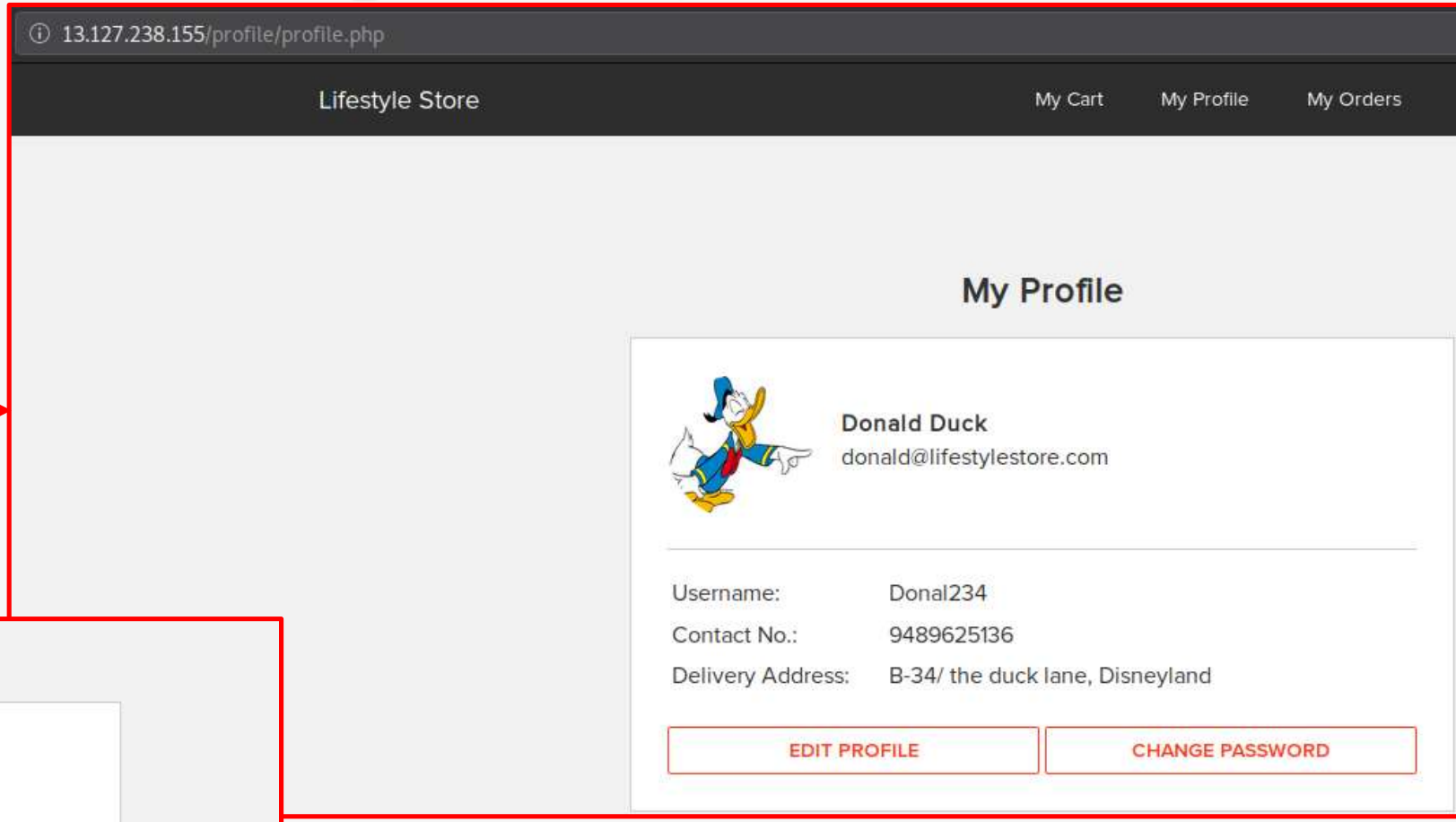


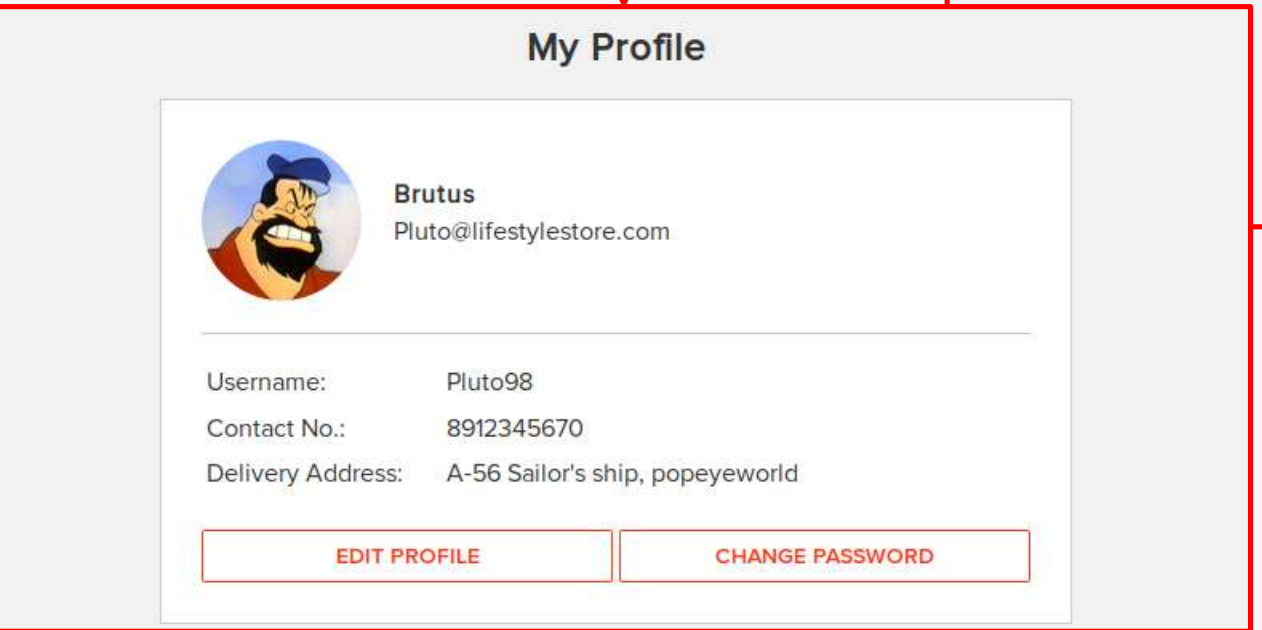
We can change the password as the send option triggers a server error which discloses source code and makes the reset button visible.

13.127.238.155/reset_password/customer.php?username=Pluto98

string(20) "hackinglab4@zoho.com" object(PHPMailer\PHPMailer\Exception)#6 (7) { ["message":protected]=> string(35) "SMTP Error: Could not authenticate." ["string":"Exception":private]=> string(0) "" ["code":protected]=> int(0) ["file":protected]=> string(69) "/var/www/hacking_project/vendor/phpmailer/phpmailer/src/PHPMailer.php" ["line":protected]=> int(1960) ["trace":"Exception":private]=> array(4) { [0]=> array(6) { ["file"]=> string(69) "/var/www /hacking_project/vendor/phpmailer/phpmailer/src/PHPMailer.php" ["line"]=> int(1774) ["function"]=> string(11) "smtpConnect" ["class"]=> string(29) "PHPMailer\PHPMailer\PHPMailer" ["type"]=> string(2) "->" ["args"]=> array(1) { [0]=> array(0) { } } } [1]=> array(6) { ["file"]=> string(69) "/var/www/hacking_project/vendor/phpmailer/phpmailer/src/PHPMailer.php" ["line"]=> int(1516) ["function"]=> string(8) "smtpSend" ["class"]=> string(29) "PHPMailer\PHPMailer\PHPMailer" ["type"]=> string(2) "->" ["args"]=> array(2) { [0]=> string(482) "Date: Thu, 23 Apr 2020 01:42:41 +0530 To: Pluto@lifestylestore.com From: Hackinglab Reply-To: No Reply Subject: Password reset request Message-ID: X-Mailer: PHPMailer 6.0.6 (https://github.com/PHPMailer/PHPMailer) MIME-Version: 1.0 Content-Type: multipart/alternative; boundary="b1_cpWwzgupE4VZ7yMsY92kmFERjJw7NFbGQrlFBX5pZM" Content-Transfer-Encoding: 8bit " [1]=> string(585) "This is a multi-part message in MIME format. --b1_cpWwzgupE4VZ7yMsY92kmFERjJw7NFbGQrlFBX5pZM Content-Type: text/plain; charset=us-ascii Copy and paste this url http://13.127.238.155/reset_password/verify.php?key=19486318945c666a037b1432.99985767 in browsers address bar to reset your password --b1_cpWwzgupE4VZ7yMsY92kmFERjJw7NFbGQrlFBX5pZM Content-Type: text/html; charset=us-asci Click here to reset your password --b1_cpWwzgupE4VZ7yMsY92kmFERjJw7NFbGQrlFBX5pZM-- " } } [2]=> array(6) { ["file"]=> string(69) "/var/www/hacking_project/vendor/phpmailer/phpmailer/src/PHPMailer.php" ["line"]=> int(1352) ["function"]=> string(8) "postSend" ["class"]=> string(29) "PHPMailer\PHPMailer\PHPMailer" ["type"]=> string(2) "->" ["args"]=> array(0) { } } [3]=> array(6) { ["file"]=> string(52) "/var/www/hacking_project /reset_password/customer.php" ["line"]=> int(51) ["function"]=> string(4) "send" ["class"]=> string(29) "PHPMailer\PHPMailer\PHPMailer" ["type"]=> string(2) "->" ["args"]=> array(0) { } } } ["previous":"Exception":private]=> NULL }

Proof of Concept (PoC)





We can clearly change the password and get into their accounts and carry out all sorts of attacks and exploit their information.

Business Impact : High

By exploiting this vulnerability , an attacker can simply take over the account of any customer by knowing his username. To his benefit few usernames are given right in the login screen (Information Exposure).
This vulnerability is a severe vulnerability which enables anyone to take over the account of a customer and hence will be able to act on their behalf with or without malicious intent.
This can prove to be dangerous for the organisation because, the credentials, order history and all the account specific actions are exploitable. The attacker can carry out further complex attacks using this account and their identity wouldn’t be revealed.
The customer can sue the organisation for this information leak and the organisation can face hefty losses and defacement.

Recommendation

This specific Information Disclosure is due to an error on the server side, hence the error shall be addressed immediately and the source code shall be debarred from being visible under any circumstances.
Always check whether each of the requests to create/edit/view/delete resources has proper access controls, preventing privilege escalation issues and making sure that all the confidential information remains confidential.
Configure the web server to suppress any exceptions that may arise and return a generic error page.

References

https://www.netsparker.com/blog/web-security/information-disclosure-issues-attacks/
https://cwe.mitre.org/data/definitions/209.html
https://cwe.mitre.org/data/definitions/728.html
https://cwe.mitre.org/data/definitions/389.html

Vulnerability Classification: CWE-209

9. Clear Text Submission of Password

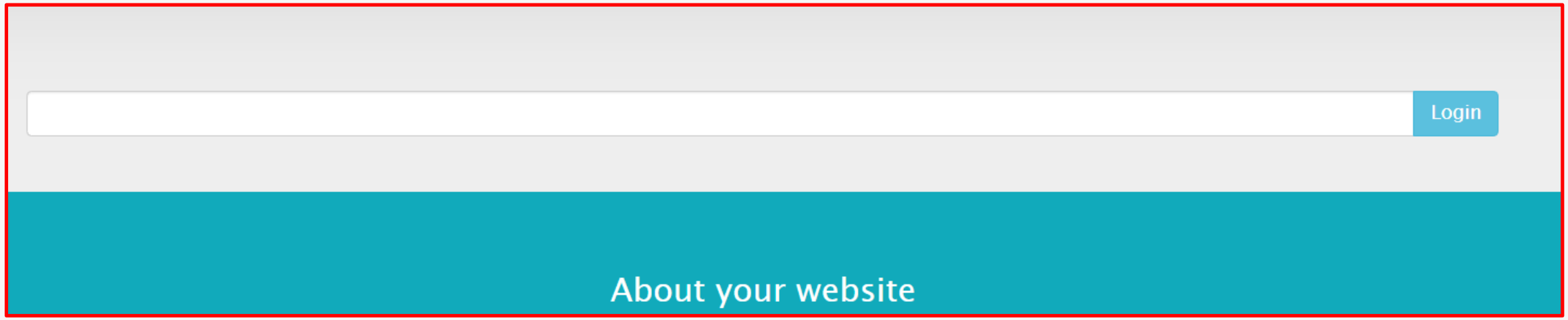
Some applications transmit passwords over unencrypted connections, making them vulnerable to interception. To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer.

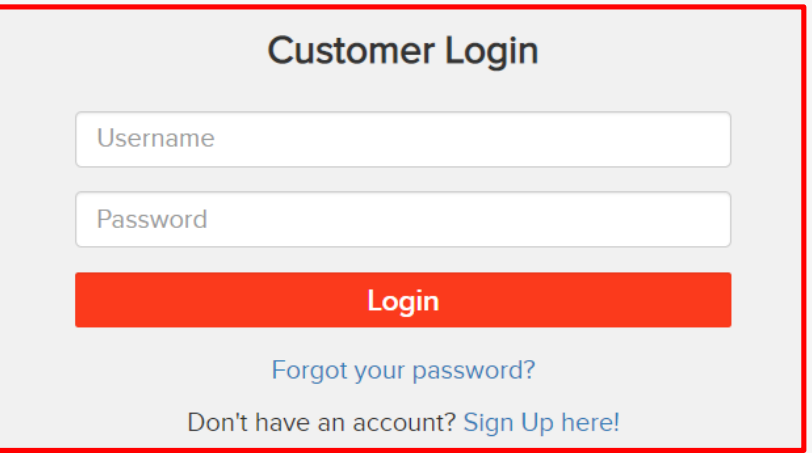
	Affected URL	Parameters	Payload
Clear Text Submission of Password (severe)	http://13.127.238.155/login/customer.php	Username, Password	--NA--
	http://13.127.238.155/login/seller.php	Username, Password	--NA--
	http://13.127.238.155/login/admin.php	Username, Password	--NA--
	http://13.127.238.155/login/wondercms/loginURL	Password	--NA--

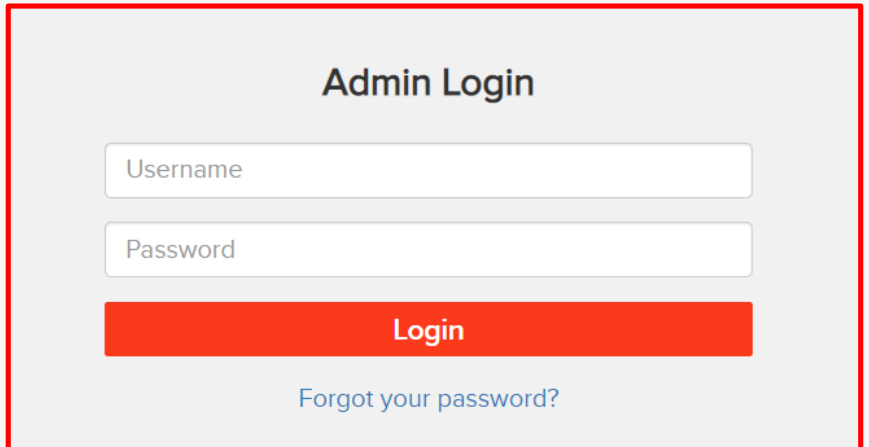
Clear Text Submission of Password (Severe)	http://13.127.238.155/profile/change_password_submit.php	Password, New Password	--NA--
	http://13.127.238.155/signup/customer.php	Username, Password	--NA--
	http://13.127.238.155/forum/index.php?u=/user/login	Username, Password	--NA--
	http://13.126.119.22/forum/index.php?u=/user/profile/5/edit#edit	Current Password, New Password, Confirm Password	--NA--
	http://13.126.119.22/forum/index.php?u=/user/register	Username, Password, Mail	--NA--
	http://13.234.35.186/ovidentiaCMS/index.php	Username, Password	--NA--

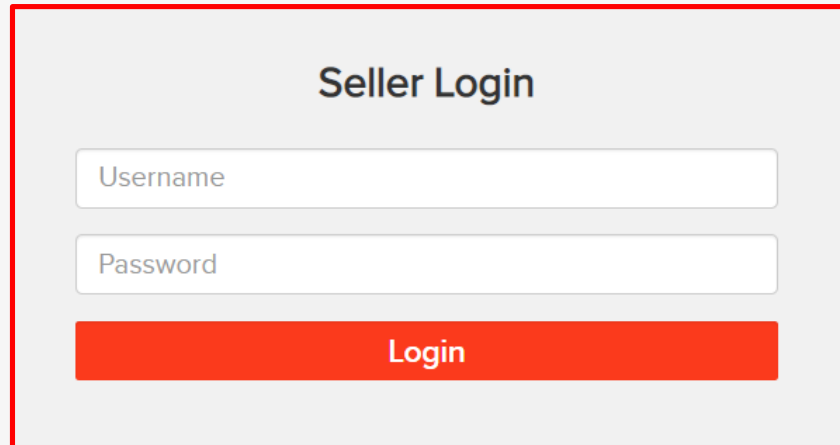
Observation

All the forms used to take input sensitive information such as the ‘Username’ and ‘Password’ are unencrypted and submitted in clear text. These are the login forms which are vulnerable. Any one who can capture requests can definitely see these usernames and passwords leaving the accounts of the holders compromised.









Proof of Concept (PoC)

Due to unencrypted usernames and passwords these elements can be viewed just by capturing the requests making the accounts vulnerable.

```
POST /login/submit.php HTTP/1.1
Host: 35.154.192.226
Content-Length: 117
Accept: */*
Origin: http://35.154.192.226
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36 OPR/67.0.3575.137
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://35.154.192.226/login/admin.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; PHPSESSID=4ogh1ik0tdvovqftr32efkvla5; X-XSRF-TOKEN=af939a3846ef73a67e9c0e6c400348d095488de9dc88544afaebe581daf66770
Connection: close

type=admin&username=admin&password=pass&X-XSRF-TOKEN=af939a3846ef73a67e9c0e6c400348d095488de9dc88544afaebe581daf66770
```

Raw | Params | Headers | Hex

```
POST /login/submit.php HTTP/1.1
Host: 35.154.192.226
Content-Length: 123
Accept: */*
Origin: http://35.154.192.226
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36 OPR/67.0.3575.137
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://35.154.192.226/login/customer.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; PHPSESSID=4ogh1ik0tdvovqftr32efkvla5; X-XSRF-TOKEN=117c5ca0737b9ca9fae1340c77a24e9d68b12380af0e228206fd756f520c9769
Connection: close

type=customer&username=Donal234&password=pass&X-XSRF-TOKEN=117c5ca0737b9ca9fae1340c77a24e9d68b12380af0e228206fd756f520c9769
```

Raw | Params | Headers | Hex

```
POST /login/submit.php HTTP/1.1
Host: 35.154.192.226
Content-Length: 126
Accept: */*
Origin: http://35.154.192.226
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36 OPR/67.0.3575.137
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://35.154.192.226/login/seller.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; PHPSESSID=4ogh1ik0tdvovqftr32efkvla5; X-XSRF-TOKEN=a25e4137b2d41a9e7be77e3a377733b1052c082e4859134646918370904898a7
Connection: close

type=seller&username=chandan&password=chandan123&X-XSRF-TOKEN=a25e4137b2d41a9e7be77e3a377733b1052c082e4859134646918370904898a7
```

Raw | Params | Headers | Hex

```
POST /signup/customer_submit.php HTTP/1.1
Host: 35.154.192.226
Content-Length: 197
Accept: */*
Origin: http://35.154.192.226
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36 OPR/67.0.3575.137
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://35.154.192.226/signup/customer.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; PHPSESSID=4ogh1ik0tdvovqftr32efkvla5; X-XSRF-TOKEN=1b0225745d873612036e075c3f2a21094940f1c2692df4b911dbb920b4a2aff0
Connection: close

name=bulla&email=emailid%40email.com&password=password&username=username&contact=9876543210&address=home%2Cunknown+lane&X-XSRF-TOKEN=1b0225745d873612036e075c3f2a21094940f1c2692df4b911dbb920b4a2aff0
```

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://35.154.192.226/wondercms/loginURL
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; PHPSESSID=4ogh1ik0tdvovqftr32efkvla5; X-XSRF-TOKEN=6684aaecfc2d12cf5446854a45fc2924af0c760ea6282480e49080d5aa1dfc8e
Connection: close

password=admin
```

Raw | Params | Headers | Hex

```
GET /forum/index.php?u=/Ajax/user/login/dologin&username=usrname&password=password&remember=true&token=9a9fbb19deaf60581334f35151a0fac0 HTTP/1.1
Host: 13.126.119.22
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36 OPR/67.0.3575.137
Referer: http://13.126.119.22/forum/index.php?u=/user/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; X-XSRF-TOKEN=eefcb31485619941lea888af77890fe15a3f75f0d18c834ffd1fb1fbb4bb0d0a; PHPSESSID=6pgq6hco7fd11doshtmlrqcgh7
Connection: close
```

Raw | Params | Headers | Hex

```
POST /forum/index.php?u=/user/register HTTP/1.1
Host: 13.126.119.22
Content-Length: 97
Cache-Control: max-age=0
Origin: http://13.126.119.22
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36 OPR/67.0.3575.137
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://13.126.119.22/forum/index.php?u=/user/register
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; PHPSESSID=3fg022cfjgsrtel3mhpdioc3i2; X-XSRF-TOKEN=eefcb31485619941lea888af77890fe15a3f75f0d18c834ffd1fb1fbb4bb0d0a
Connection: close

username=username&password=password&mail=email%40email.com&token=9a9fbb19deaf60581334f35151a0fac0
```

Business Impact : High

Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.
Vulnerabilities that result in the disclosure of users' passwords can result in compromises that are extremely difficult to investigate due to obscured audit trails. Even if the application itself only handles non-sensitive information, exposing passwords puts users who have re-used their password elsewhere at risk.

Recommendation

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. Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2.

References

https://cwe.mitre.org/data/definitions/319.html
https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A3-Sensitive_Data_Exposure
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Vulnerability Classification: CWE-319

10. Personal Identifiable Information Leakage

PII Leakage is : Any information about an individual maintained by an agency, including any information that can be used to distinguish or trace someone's identity, such as name, social security number, date, place of birth, mother's maiden name, or biometric records; any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information.

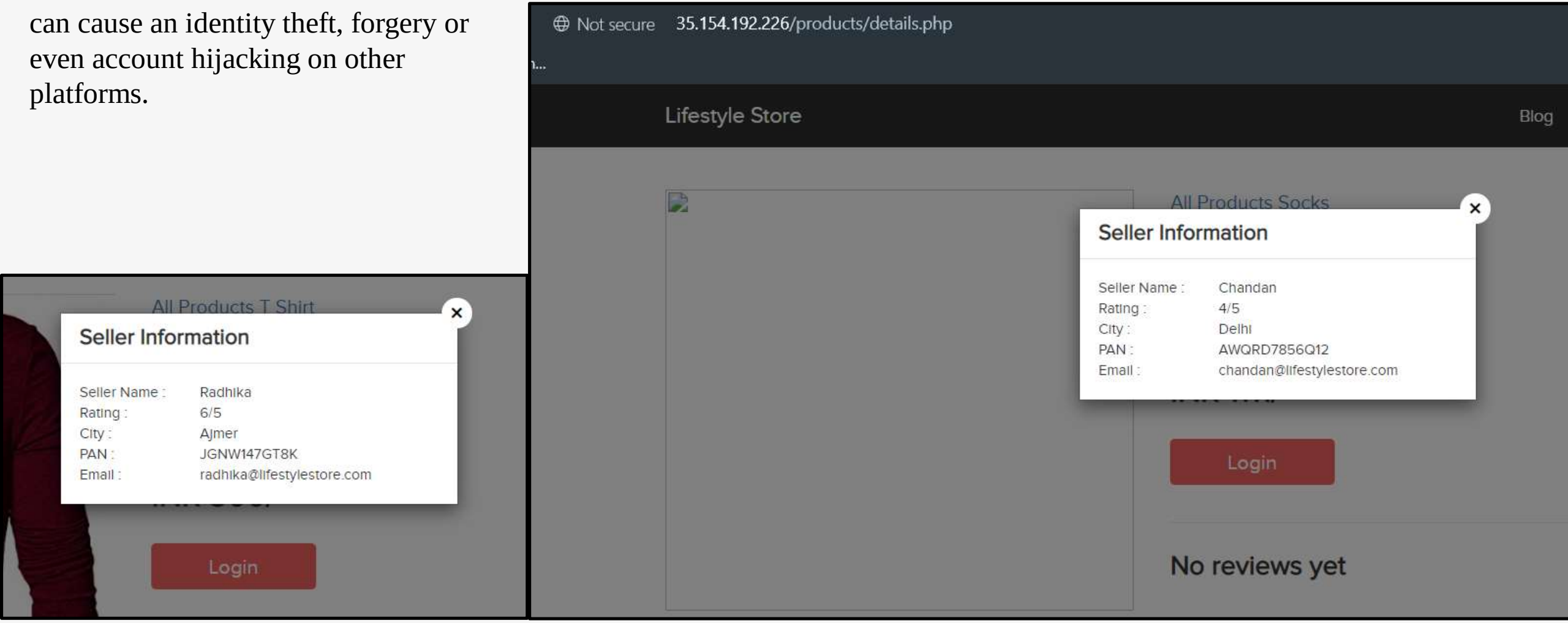
PII Leakage (severe)	Affected URL	Parameters	Payload
	http://13.233.186.18/products/details.php?p_id=11 (For all the items that have seller info())	---NA---	---NA---

Observation

As we surf through the products we find an option of seller information. Upon clicking it we can find various sorts of information about an individual with which he can be easily identified. This information leakage may cause trouble for the seller and the organisation. The seller may be harmed at various levels financially, personally or even physically.

Proof of Concept (PoC)

We can clearly see that seller name, his city of origin, his PAN number and email address have been revealed. This can cause an identity theft, forgery or even account hijacking on other platforms.



Business Impact : High

An attacker doesn't need a social security number and date of birth to cause harm to your employees and customers. Just having the target's first and last name, email address, physical address, phone number, and last four digits of a credit card is enough to do damage. The verification process used by most companies involves a couple of pieces of the person's information when completing a change to service, sending a paycheck to another address, or setting up a service like Door Dash.

Recommendation

Any sort of information that may help in identifying an individual and may yield in jeopardizing his/her life or finance in danger should be removed from the website immediately. If some information is needed for display purposes, a pseudo name or username which doesn't directly indicate a person's identification should be used.

References

https://www.lifelock.com/learn-identity-theft-resources-what-is-personally-identifiable-information.html
https://www.imperva.com/learn/data-security/personally-identifiable-information-pii/

11. Cross Site Request Forgery (CSRF)

Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.

	Affected URL	Parameters	Payload
Cross Site Request Forgery (CSRF) (severe)	http://13.233.186.18/profile/profile.php(For all accounts)	Referer	Desired website name
	http://13.233.186.18/cart/cart.php (for all customers)	Referer	Desired website name
	http://13.126.119.22/forum/index.php?u=/user/profile/5/edit#edit	Referer	Desired website name

Observation

On editing the profile we come across the change password option. When we input the new password we can intercept the request and change the referer which can direct to a malicious site where we can have a malicious code or page waiting for the user to press in order for him/her to fall into this request forgery.

Original request | Edited request | Response

Raw | Params | Headers | Hex

```
1  POST /profile/change_password_submit.php HTTP/1.1
2  Host: 13.233.186.18
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://hacker.com
8  Content-Type: application/x-www-form-urlencoded; charset=UTF-8
9  X-Requested-With: XMLHttpRequest
10 Content-Length: 37
11 Connection: close
12 Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; PHPSESSID=arel5b9vfad7fbtm6ojpr
13 DNT: 1
14
15 password=pass2&password_confirm=pass2
```

Original request | Edited request | Response

Raw | Headers | Hex | Render

```
1  HTTP/1.1 200 OK
2  Server: nginx/1.14.0 (Ubuntu)
3  Date: Tue, 21 Apr 2020 07:49:29 GMT
4  Content-Type: text/html; charset=utf-8
5  Connection: close
6  Expires: Thu, 19 Nov 1981 08:52:00 GMT
7  Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
8  Pragma: no-cache
9  X-FRAME-OPTIONS: DENY
10 Content-Length: 65
11
12 {"success":true,"successMessage":"Password updated succesfully."}
```

Proof of Concept (PoC)

Raw | Params | Headers | Hex

```
POST /orders/confirm.php HTTP/1.1
Host: 13.126.117.236
Content-Length: 0
Cache-Control: max-age=0
Origin: http://13.126.117.236
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application
Referer: www.x-hacker.com/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=e7jka91b7dmh4fgqqruldq9446; key=6740B825-6E43-F3F0-FDD0-04ED720ED474; X-XSRF-TOKEN=4725dd043bc30f52c3ee1cf1058ba7b6019673843192297f6b713169e8474218
Connection: close
```

Shopping Cart

S.No	Product	Price
1	Reebok Men Socks Remove	1111
	Total	1111

*forge - Notepad

File Edit Format View Help

```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="http://13.126.117.236/cart/remove.php" method="POST">
      <input type="hidden" name="product&#95;id" value="1" value="2" value="3" value="4" value="5" value="6" value="15"
value="16" value="17" value="21" value="12" value="10" value="19" />
      <input type="hidden" name="X&#45;XSRF&#45;TOKEN"
value="d344ae64ffc691407c6fe74ab7b67b10b74d17275a1f966d026cff97f26fa395" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

We can see that changing the referer link and creating our own html to alter the request consequences we were able to use remove.php to remove all the items whose item number values were specified in the script.

Receipt

Order Id: A7642F3D50EA

PRODUCTS:

Reebok Men Socks	INR 1111
Total	**INR 1111**

SHIPPING DETAILS:

Name - Popeye the sailor man
Email - popeye@lifestylestore.com
Phone - 9745612300
Address - B-44 spinach house, Disneyworld

PAYMENT MODE

Cash on delivery

Order placed on : 2020-04-27 01:03:35

Status: DELIVERED

Business Impact : High

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.

The impact of a successful CSRF attack is limited to the capabilities exposed by the vulnerable application and privileges of the user. For example, this attack could result in a transfer of funds, changing a password, or making a purchase with the user's credentials. In effect, CSRF attacks are used by an attacker to make a target system perform a function via the victim's browser, without the victim's knowledge, at least until the unauthorized transaction has been committed.

Recommendation

Check if your framework has built-in CSRF protection and use it

- If framework does not have built-in CSRF protection add CSRF tokens to all state changing requests (requests that cause actions on the site) and validate them on backend

Always use Same Site Cookie Attribute for session cookies

Implement at least one mitigation from Defense in Depth Mitigations section

- Use custom request headers
- Verify the origin with standard headers
- Use double submit cookies

Consider implementing user interaction based protection for highly sensitive operations

Remember that any Cross-Site Scripting (XSS) can be used to defeat all CSRF mitigation techniques!

- See the OWASP XSS Prevention Cheat Sheet for detailed guidance on how to prevent XSS flaws.

Do not use GET requests for state changing operations.

- If for any reason you do it, you have to also protect those resources against CSRF

References

https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
https://owasp.org/www-community/attacks/csrf

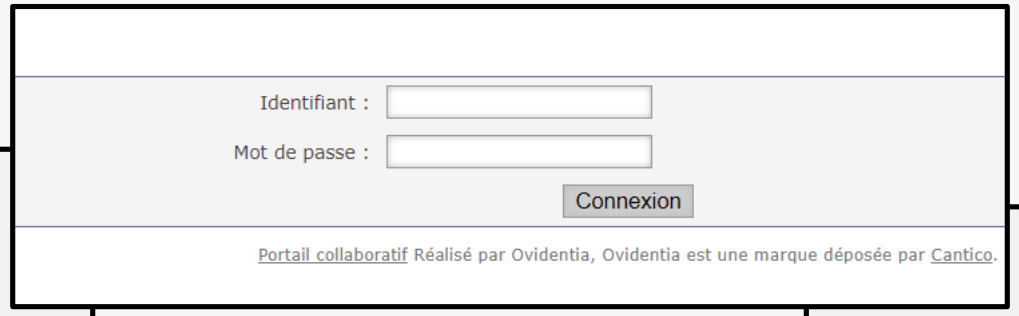
12. Rate Limiting Flaw

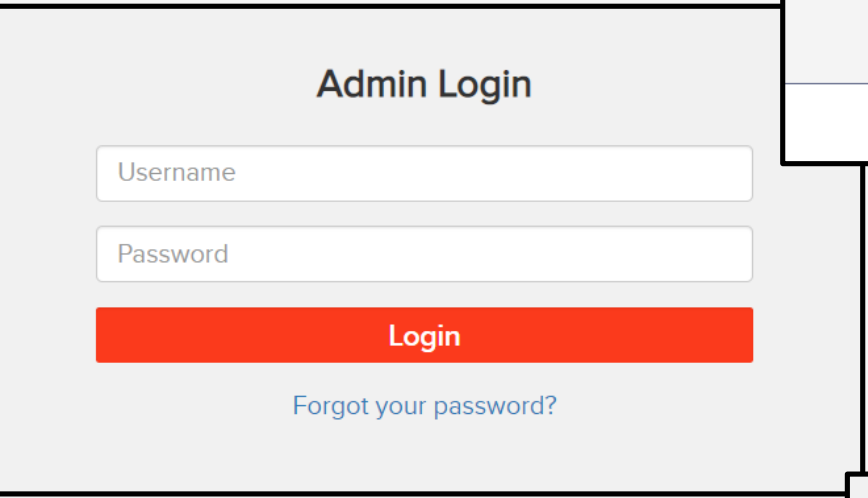
Rate limiting is used to control the amount of incoming and outgoing traffic to or from a network. For example, let's say you are using a particular service's API that is configured to allow 100 requests/minute. If the number of requests you make exceeds that limit, then an error will be triggered. If the system isn't configured to rate limiting it is said to have the **Rate Limiting Flaw**.

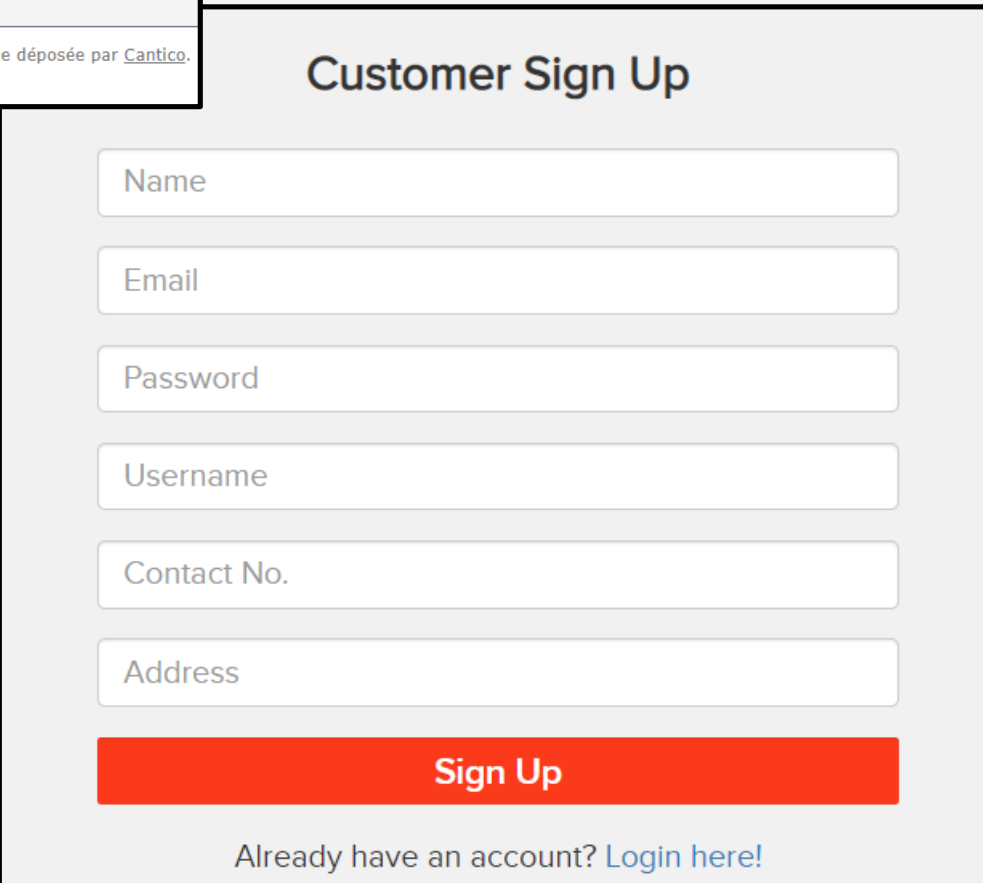
	Affected URL	Parameters	Payload
Rate Limiting Flaw (severe)	http://13.127.238.155/login/customer.php	Username, Password	--NA--
	http://13.127.238.155/login/seller.php	Username, Password	--NA--
	http://13.127.238.155/login/admin.php	Username, Password	--NA--
	http://13.127.238.155/login/wondercms/loginURL	Password	--NA--
	http://13.127.238.155/signup/customer.php	Username, Password	--NA--
	http://13.127.238.155/forum/index.php?u=/user/login	Username, Password	--NA--
	http://13.126.119.22/forum/index.php?u=/user/register	Username, Password Email	--NA--

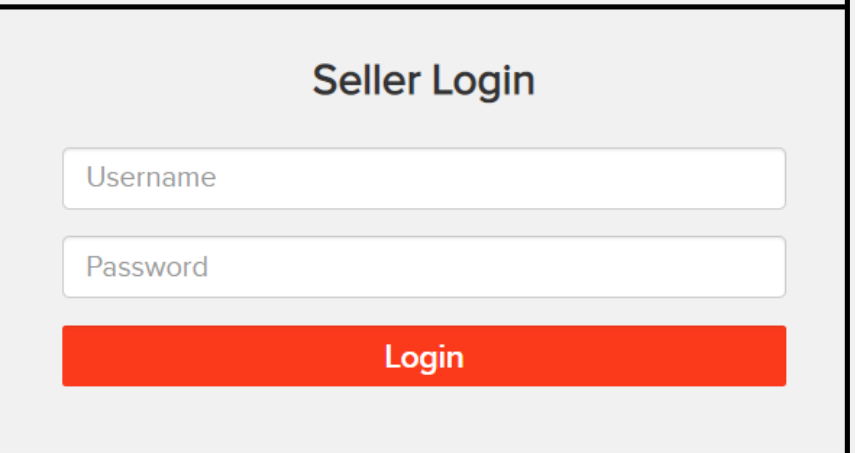
Observation

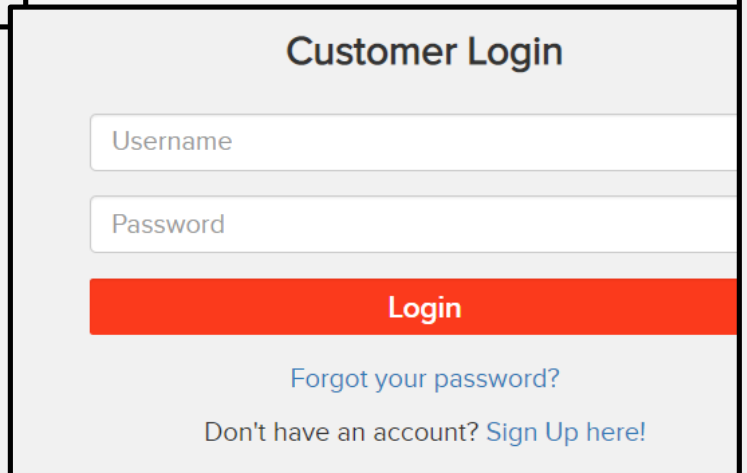
All the login pages on this website are susceptible to brute forcing because of rate limiting flaw. No rate limiter was encountered at any point of the testing. All the payloads were successfully tested on the vulnerable positions and no error occurred.











Proof of Concept (PoC)

We can clearly see by inputting a heavy number of payloads in the order of thousands, there are no request prohibitions being encountered. Hence all these login pages need to be configured with a rate limiter of maximum 100 requests/min.

```
GET /forum/index.php?u=/Ajax/user/login/dologin&username=$usrname$&password=$password$&remember=true&token=9a9fbb19deaf60581334f35151a0fac0 HTTP/1.1
Host: 13.126.119.22
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, l
```

Target | Positions | Payloads | Options

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines

Attack type: Pitchfork

```
POST /login/submit.php HTTP/1.1
Host: 52.66.125.76
Content-Length: 118
Accept: */*
Origin: http://52.66.125.76
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://52.66.125.76/login/admin.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; PHPSESSID=ic6krfg7kls16hrd
Connection: close

type=admin&username=$admin$&password=$admin$&X-XSRF-TOKEN=39b29ae6b619ed11e3
```

```
type=seller&username=$donal234$&password=$pass$&X-XSRF-TOKEN=c32c8d84bb305bc
```

```
type=customer&username=$donal234$&password=$pass$&X-XSRF-TOKEN=912c06c660a10
```

Filter: Showing all items

Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
3414	frazer	zephyr	200			392	
3415	frazier	zeppelin	200			392	
3416	fred	zeus	200			392	
3417	freda	zhongguo	200			392	
3418	freddi	ziggy	200			392	
3419	freddie	zimmerman	200			392	
3420	freddy	zjaaadc	200			392	
3421	fredek	zmodem	200			392	
3422	fredelia	zombie	200			392	
3423	frederic	zorro	200			392	
3424	frederica	zxcvbnm	200			392	

Request | Response

Raw | Params | Headers | Hex

```
GET
/forum/index.php?u=/Ajax/user/login/dologin&username=!root&password=!@#$%&remember=true&token=9a9fbb19deaf60581334f
35151a0fac0 HTTP/1.1
Host: 13.126.119.22
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163
Safari/537.36 OPR/67.0.3575.137
Referer: http://13.126.119.22/forum/index.php?u=/user/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474;
X-XSRF-TOKEN=eefcb31485619941lea888af77890fe15a3f75f0d18c834ffd1fb1fbb4bb0d0a; PHPSESSID=6pgq6hco7fd11doshtmlrqcgh7
Connection: close
```

Business Impact : High

When the number of requests sent to the server are in huge numbers , such as in the range of 100 thousands the server faces delay in response time and generally slows down. This shall effect other users of the system and ruin their experience of the website. The signup function for the customers is also not rate limited which leaves it vulnerable to multiple account creations; therefore filling up server space. Hence, a wastage of resources will occur. The reasoning behind implementing rate limits is to allow for a **better flow of data** and to **increase security** by mitigating attacks such as DDoS.

Another impact of rate limiting flaw is that , it is now highly susceptible to brute force attacks which will be able to break passwords making the all account logins vulnerable to attack.

Recommendation

There are various methods and parameters that can be defined when setting rate limits. The rate limit method that should be used will depend on what you want to achieve as well as how restrictive you want to be. The section below outlines **three different types** of rate limiting methods that you can implement.

User rate limiting: The most popular type of rate limiting is user rate limiting. This associates the number of requests a user is making to their API key or IP (depending on which method you use). Therefore, if the user exceeds the rate limit, then any further requests will be denied until they reach out to the developer to increase the limit or wait until the rate limit timeframe resets.

Geographic rate limiting: To further increase security in certain geographic regions, developers can set rate limits for particular regions and particular time periods. For instance, if a developer knows that from midnight to 8:00 am users in a particular region won't be as active, then they can define lower rate limits for that time period. This can be used as a preventative measure to help further reduce the risk of attacks or suspicious activity.

Server rate limiting: If a developer has defined certain servers to handle certain aspects of their application then they can define rate limits on a server-level basis. This gives developers the freedom to decrease traffic limits on server A while increasing it on server B (a more commonly used server).

References

https://www.keycdn.com/support/rate-limiting
https://www.nginx.com/blog/rate-limiting-nginx/
https://en.wikipedia.org/wiki/Rate_limiting

13. Open Redirection

Open redirection vulnerabilities arise when an application incorporates user-controllable data into the target of a redirection in an unsafe way. An attacker can construct a URL within the application that causes a redirection to an arbitrary external domain. This behavior can be leveraged to facilitate phishing attacks against users of the application. The ability to use an authentic application URL, targeting the correct domain and with a valid SSL certificate (if SSL is used), lends credibility to the phishing attack because many users, even if they verify these features, will not notice the subsequent redirection to a different domain.

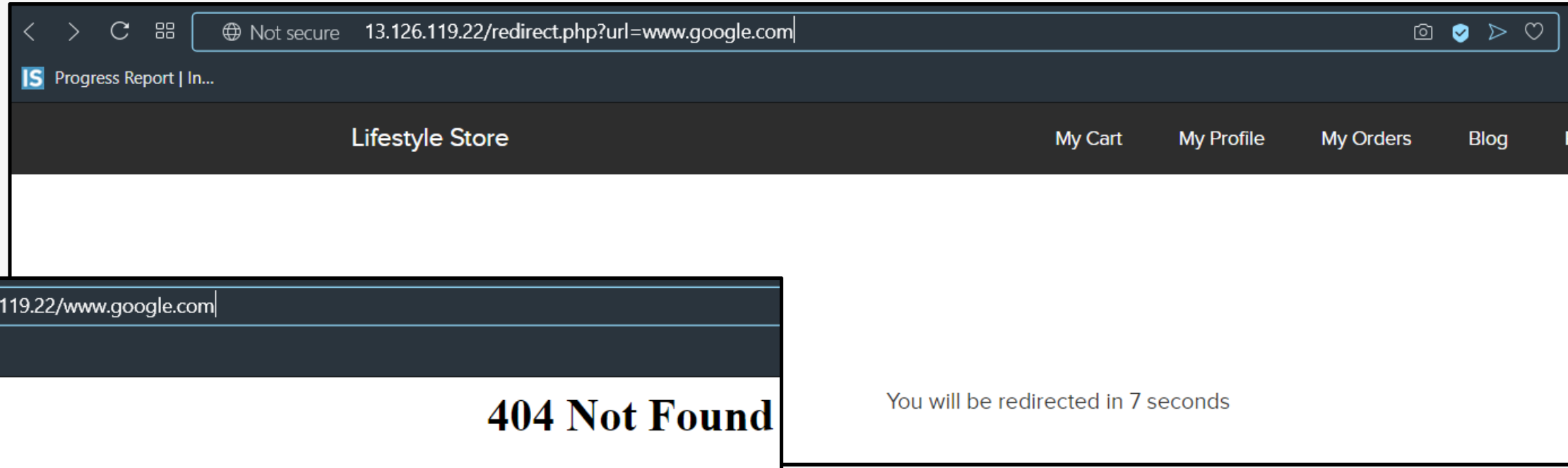
Open Redirection (severe)	Affected URL	Parameters	Payload
	http://13.126.119.22/redirect.php?url="websitename"	url	www.google.com

Observation

We can observe that by clicking the brand website button we are directed to the brand website.

Proof of Concept (PoC)

The parameter “url” is vulnerable to Open Redirection and takes us to the desired website when entered in the url.



Business Impact : High

Open redirection vulnerabilities arise when an application incorporates user-controllable data into the target of a redirection in an unsafe way. An attacker can construct a URL within the application that causes a redirection to an arbitrary external domain. This behavior can be leveraged to facilitate phishing attacks against users of the application. The ability to use an authentic application URL, targeting the correct domain and with a valid SSL certificate (if SSL is used), lends credibility to the phishing attack because many users, even if they verify these features, will not notice the subsequent redirection to a different domain.

The user may be redirected to an untrusted page that contains malware which may then compromise the user's machine. This will expose the user to extensive risk and the user's interaction with the web server may also be compromised if the malware conducts keylogging or other attacks that steal credentials, personally identifiable information (PII), or other important data.
The user may be subjected to phishing attacks by being redirected to an untrusted page. The phishing attack may point to an attacker controlled web page that appears to be a trusted web site. The phishers may then steal the user's credentials and then use these credentials to access the legitimate web site.

Recommendation

If possible, applications should avoid incorporating user-controllable data into redirection targets. In many cases, this behavior can be avoided in two ways:

Remove the redirection function from the application, and replace links to it with direct links to the relevant target URLs.

Maintain a server-side list of all URLs that are permitted for redirection. Instead of passing the target URL as a parameter to the redirector, pass an index into this list.

If it is considered unavoidable for the redirection function to receive user-controllable input and incorporate this into the redirection target, one of the following measures should be used to minimize the risk of redirection attacks:

The application should use relative URLs in all of its redirects, and the redirection function should strictly validate that the URL received is a relative URL.

The application should use URLs relative to the web root for all of its redirects, and the redirection function should validate that the URL received starts with a slash character. It should then prepend http://yourdomainname.com to the URL before issuing the redirect.

The application should use absolute URLs for all of its redirects, and the redirection function should verify that the user-supplied URL begins with http://yourdomainname.com/ before issuing the redirect.

References

https://portswigger.net/kb/issues/00500100_open-redirection-reflected
https://cwe.mitre.org/data/definitions/601.html

Vulnerability Classification: CWE-601

14. Information Exposure

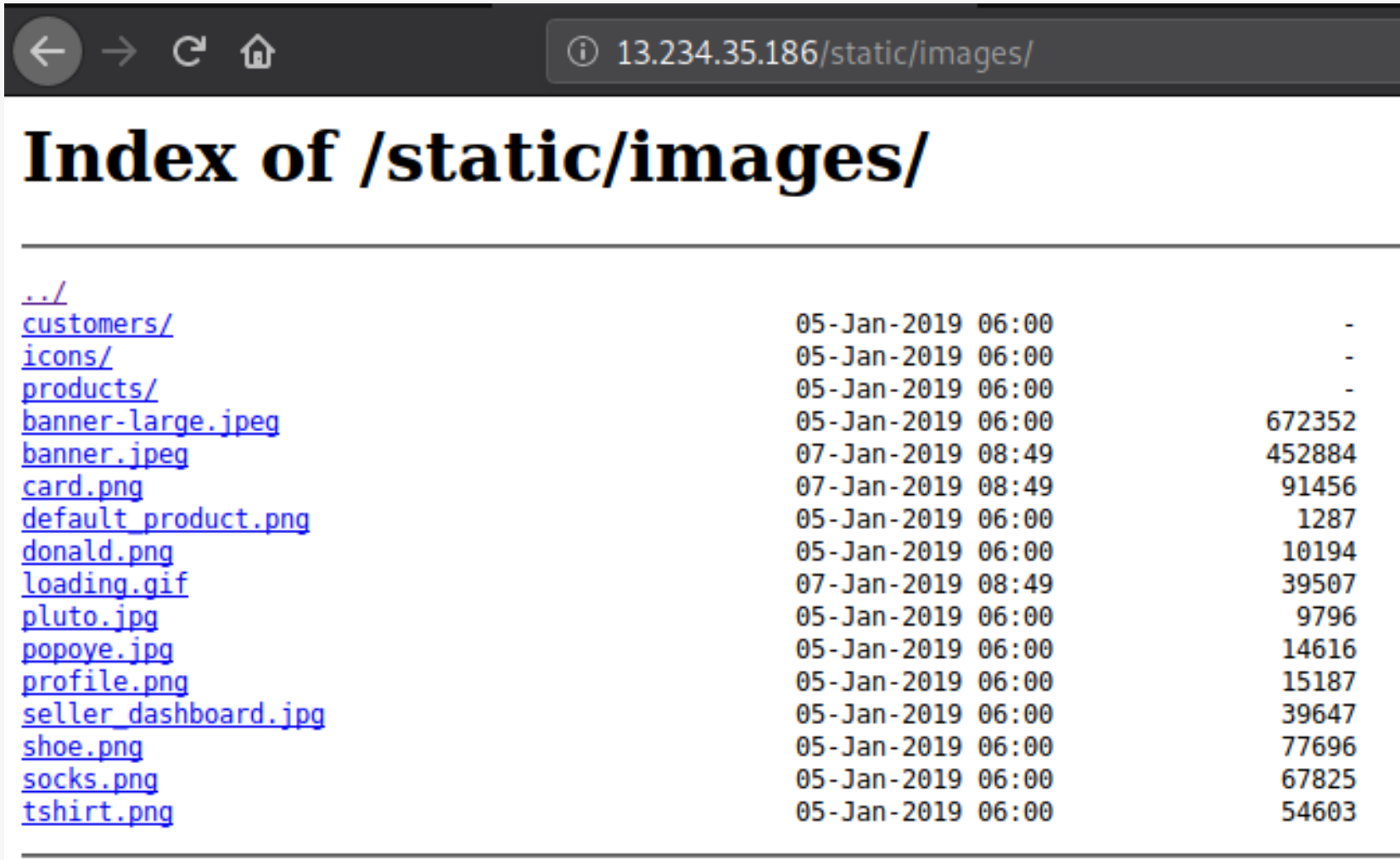
Filename and path disclosure leads to directory listing in web servers. This functionality is provided by default on web servers. When there is no default web page to show the web server shows the user a list of files and directories present on the website.

	Affected URL	Parameters	Payload
Information Exposure (moderate)	http://13.126.119.22/static/images/	---NA---	---NA---
	http://13.126.119.22/userlist.txt	--NA--	--NA--

Observation

By using dirbuster we were able to find directories with http response of 200 which was accessible to anyone who logged into this website. By using these directories , an attacker can be aware of the site structure the website uses , facilitating him/her with more complex ideas to attack the target.

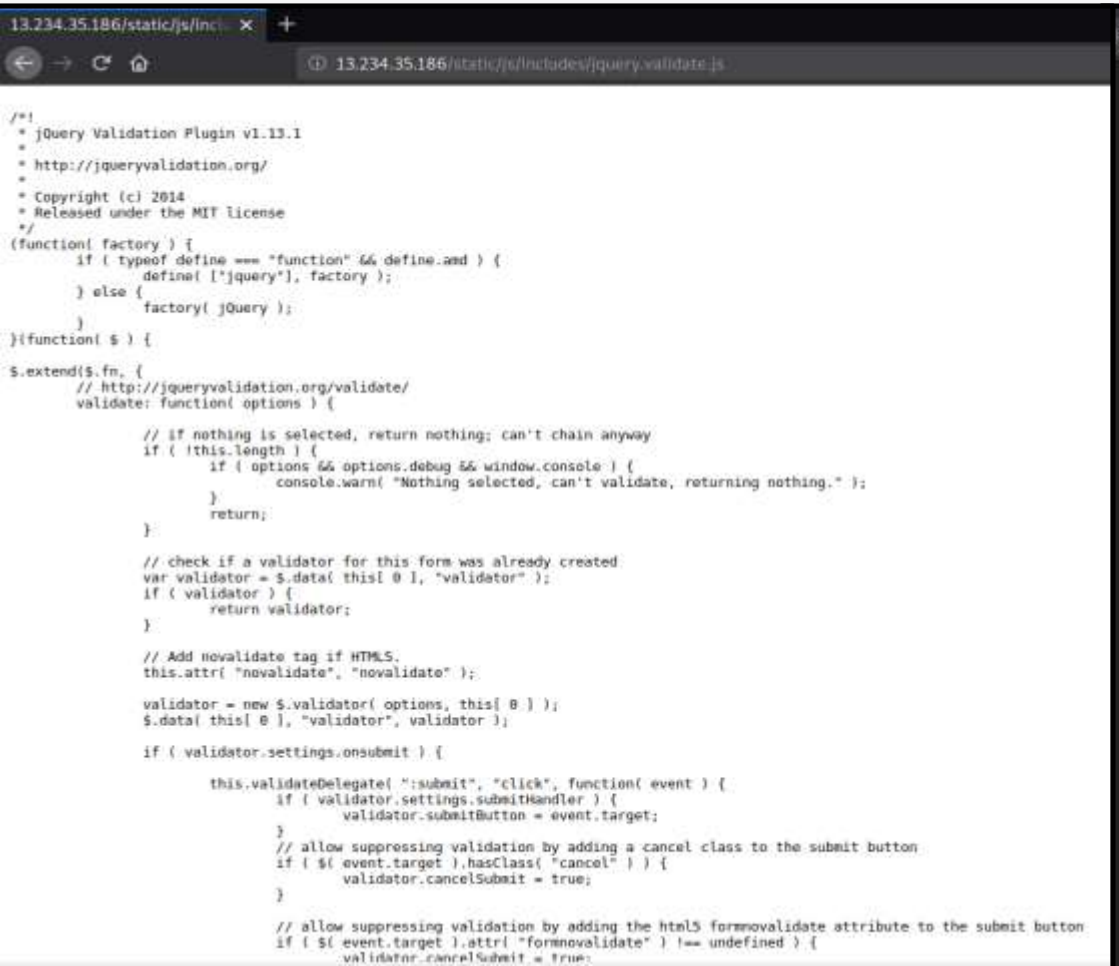
As we can see all the images stored on the server are vulnerable , the profile images , product images, icons etc.

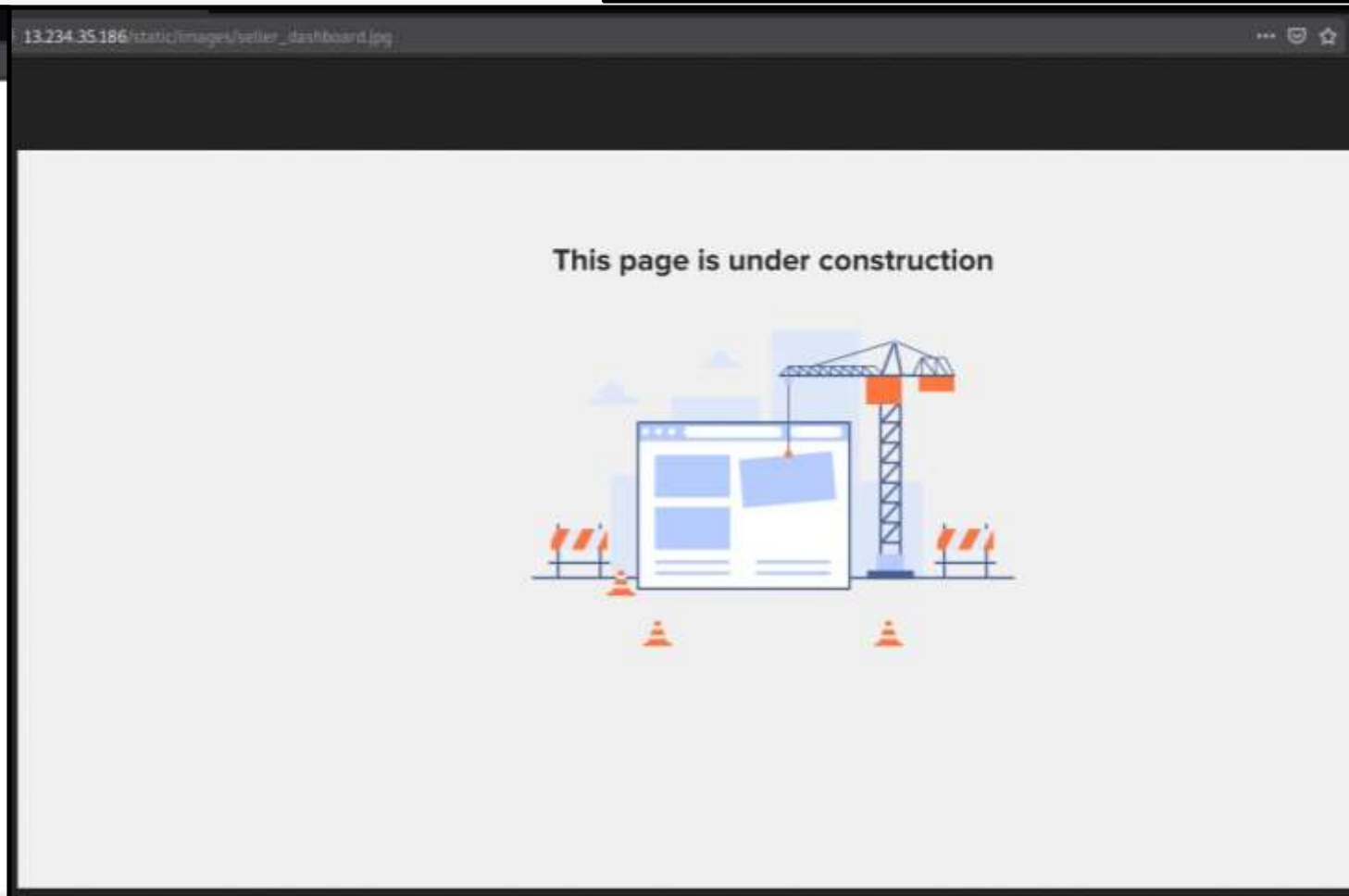


Proof of Concept (PoC)

We even come across source codes of the pages and of the plugins used which could be used to fetch out the vulnerabilities in the code and hence exploit. The pictures used in the website can be fetched out with ease.

Userlist.txt was found to have all the usernames and passwords of the sellers making their accounts vulnerable to attacks.





Business Impact : Moderate

Nowadays many are aware that such functionality should be disabled, so it is not common to see it. Though let's assume that after scanning the ports of a web server, the attacker found a default installation of the web server running on the port 8081.
Such "defaults" are typically overlooked by web server administrators, which also means they are much less secure. The installation may also have directory listing enabled, allowing the attacker to navigate through the directories and access source code, backup and possible database files of the web application.

Recommendation

Information disclosure security issues might seem like trivial issues, but they are not. They allow the malicious hackers to gain insightful and confidential information about the target they want to attack just by performing basic testing, and sometimes just by looking for information in public pages.

Therefore you should always address these issues, especially when you consider that they are really easy to mitigate against. The following are some guidelines to follow so you can make sure that your web applications are well protected against the most obvious information disclosure issues:

- Make sure that your web server does not send out response headers or background information that reveal technical details about the backend technology type, version or setup.
- Make sure that all the services running on the server's open ports do not reveal information about their builds and versions.
- Always make sure that proper access controls and authorizations are in place in order to disallow access for attackers on all web servers, services and web applications.
- Make sure that all exceptions are well handled when the web application fails and no technical information is reported in the errors.
- Do not hard code credentials, API keys, IP addresses, or any other sensitive information in the code, including first names and last names, not even in the form of comments.
- Configure the correct MIME types on your web server for all the different files being used in your web applications.
- Sensitive data, files and any other item of information that do not need to be on the web servers should never be uploaded on the web server.

- Always check whether each of the requests to create/edit/view/delete resources has proper access controls, preventing privilege escalation issues and making sure that all the confidential information remains confidential.
- Make sure that your web application processes user input correctly, and that a generic response is always returned for all the resources that don’t exist/are disallowed in order to confuse attackers.
- Enough validations should be employed by the backend code in order to catch all the exceptions and prevent the leakage of valuable information.
- Configure the web server to suppress any exceptions that may arise and return a generic error page.
- Configure the web server to disallow directory listing and make sure that the web application always shows a default web page.

References

https://www.netsparker.com/blog/web-security/information-disclosure-issues-attacks/
https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A3-Sensitive_Data_Exposure

15. Using Components having known vulnerabilities

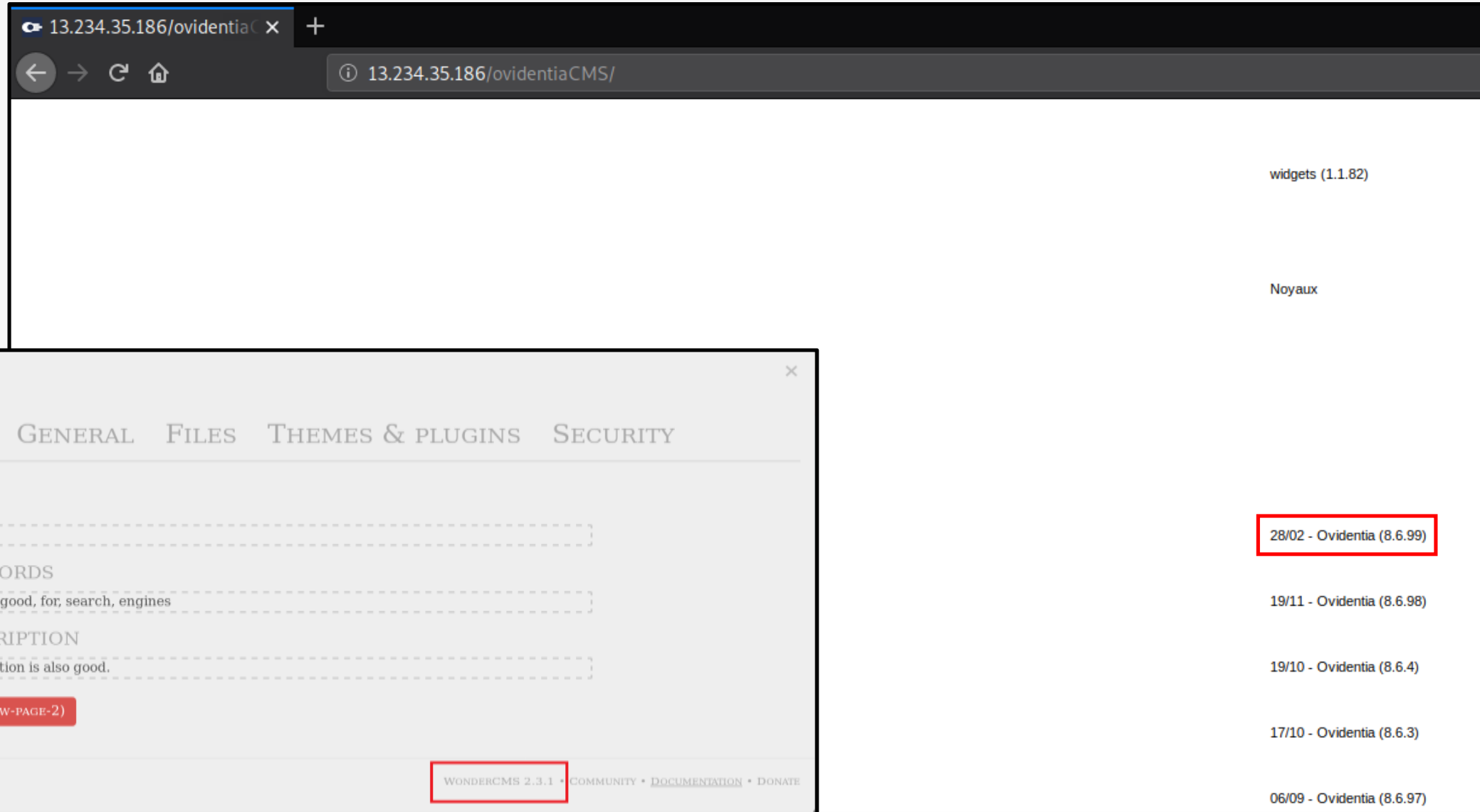
While it is easy to find already-written exploits for many known vulnerabilities, other vulnerabilities require concentrated effort to develop a custom exploit.

	Affected URL/ Affected Element	Parameters	Payload
Using Components having known vulnerabilities(moderate)	Wondercms (Content Management System)	--NA--	--NA--
	Apache(Server Software)	--NA--	--NA--
	OvidentiaCMS (Content Management System)	--NA--	--NA--
	PHP(Server Backend Language)	--NA--	--NA--

Observation

As for Wondercms the version was visible in plain sight when the admin panel was accessed.

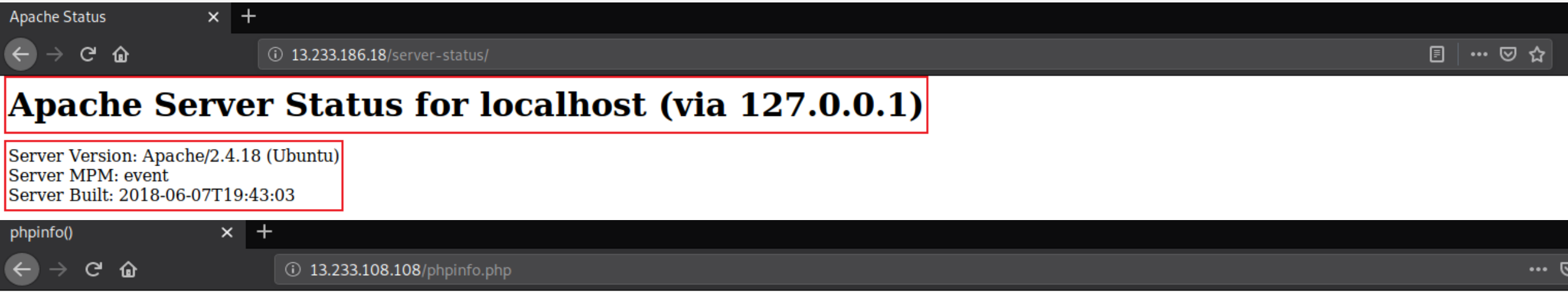
Accessing the OvidentiaCMS we could see the version in the down right corner section with the updates.

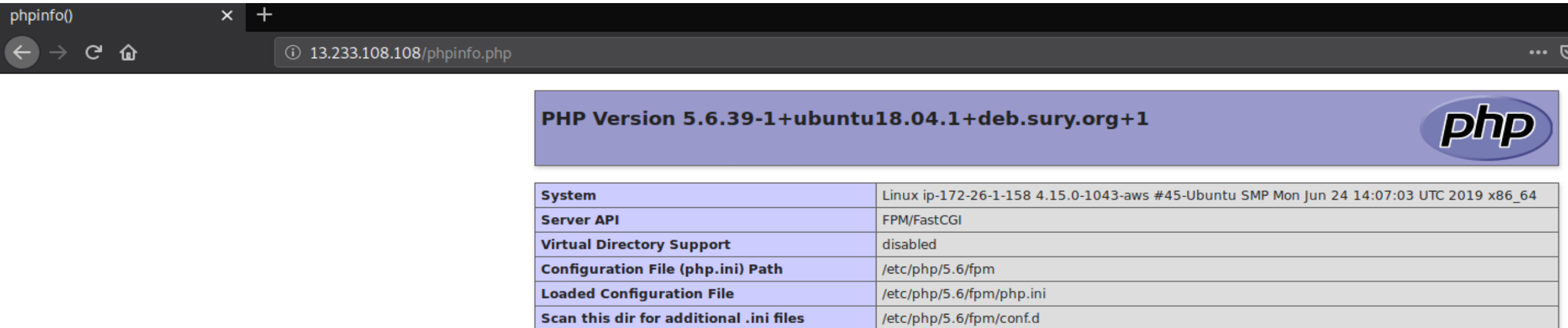


Proof of Concept (PoC)

The server-status disclosed the server used and its version , here being Apache/2.4.18 which have a ton of vulnerabilities which have been put in the reference.

The backend language used was also disclosed when phpinfo was accessed, PHP Version 5.6.39 which has a ton of vulnerabilities link to which is given in the reference section.





Business Impact : Very High

Prevalence of this issue is very widespread. Component-heavy development patterns can lead to development teams not even understanding which components they use in their application or API, much less keeping them up to date.
Some scanners such as retire.js help in detection, but determining exploitability requires additional effort.

While some known vulnerabilities lead to only minor impacts, some of the largest breaches to date have relied on exploiting known vulnerabilities in components. Depending on the assets you are protecting, perhaps this risk should be at the top of the list.

Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g. coding error) or intentional (e.g. backdoor in component).

Recommendation

There should be a patch management process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc. Continuously monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.
- Every organization must ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

References

https://www.cvedetails.com/vulnerability-list/vendor_id-74/product_id-128/version_id-167758/PHP-PHP-5.6.0.html
https://www.cvedetails.com/vulnerability-list/vendor_id-15088/product_id-30715/version_id-235577/Wondercms-Wondercms-2.3.1.html
https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-199589/Apache-Http-Server-2.4.18.html
https://www.cvedetails.com/vulnerability-list/vendor_id-8491/product_id-14870/Ovidentia-Ovidentia.html
https://owasp.org/www-project-application-security-verification-standard/

16. Unencrypted Communication

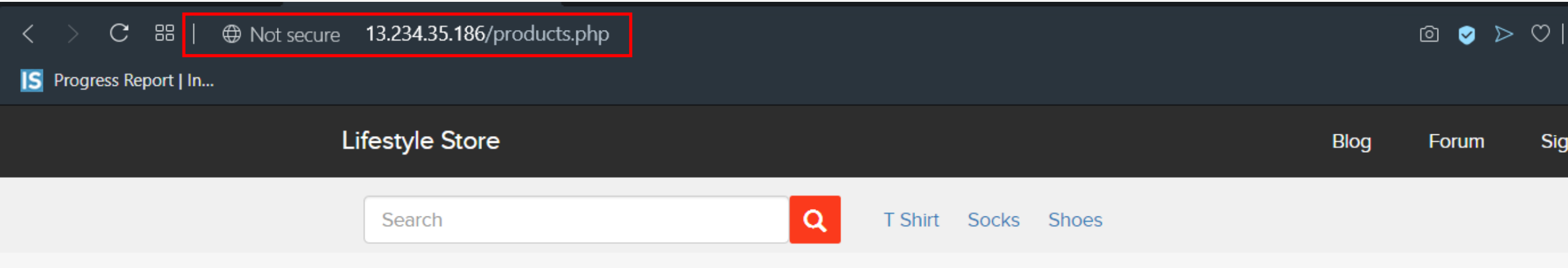
The website allows users to connect to it over unencrypted connections. An attacker suitably positioned to view a legitimate user's network traffic could record and monitor their interactions with the application and obtain any information the user supplies. Furthermore, an attacker able to modify traffic could use the application as a platform for attacks against its users and third-party websites. Unencrypted connections have been exploited by ISPs and governments to track users, and to inject adverts and malicious JavaScript. Due to these concerns, web browser vendors are planning to visually flag unencrypted connections as hazardous.

Unencrypted Communication (low)	Affected URL	Parameters	Payload
	http://13.126.119.22/	---NA---	---NA---

Observation

All the communication taking place on this website is unencrypted i.e. the response header being used is http rather than https. It is always recommended that the encrypted header https be used under all circumstances.

Proof of Concept (PoC)



Business Impact : Moderate

Insecure communications is when a client and server communicate over a non-secure (unencrypted) channel. Without encrypting the channel, the developer can’t guarantee the integrity of the data. Remember, insecure communication is different than insecure storage.
Failing to securely communicate server-to-server and server-to-client means an attacker can intercept sensitive transactions. This is typically done through man-in-the-middle attacks. Not communicating securely breaks down confidentiality and integrity.

To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.
Please note that using a mixture of encrypted and unencrypted communications is an ineffective defense against active attackers, because they can easily remove references to encrypted resources when these references are transmitted over an unencrypted connection.

Recommendation

Applications should use transport-level encryption (SSL/TLS) to protect all communications passing between the client and the server. The Strict-Transport-Security HTTP header should be used to ensure that clients refuse to access the server over an insecure connection.
To prevent insecure communications from occurring, identify where all clients and servers are communicating to each other. To reduce the number of connections to inspect, identify only where sensitive data is being passed.
A common mistake is to forget to encrypt data being transferred on back-end connections, such as database connections. Just because the data is currently behind a firewall doesn't mean it should be passed in clear-text.
To verify all communication is being done securely:
•Make sure all client-to-server connections are encrypted with SSL.
•Verify that server-to-database connections are encrypted.
•Verify that any other areas in the design where sensitive data is passed is done so in a secure way.
•Keep developers in a security mindset. Developers should never assume their application is sending their information securely. Developers should always assume that any communications that are being made are done insecurely.

References

http://bretthard.in/post/insecure-communications
https://owasp.org/www-community/vulnerabilities/Insecure_Transport
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security
Vulnerability Classification: CWE-326

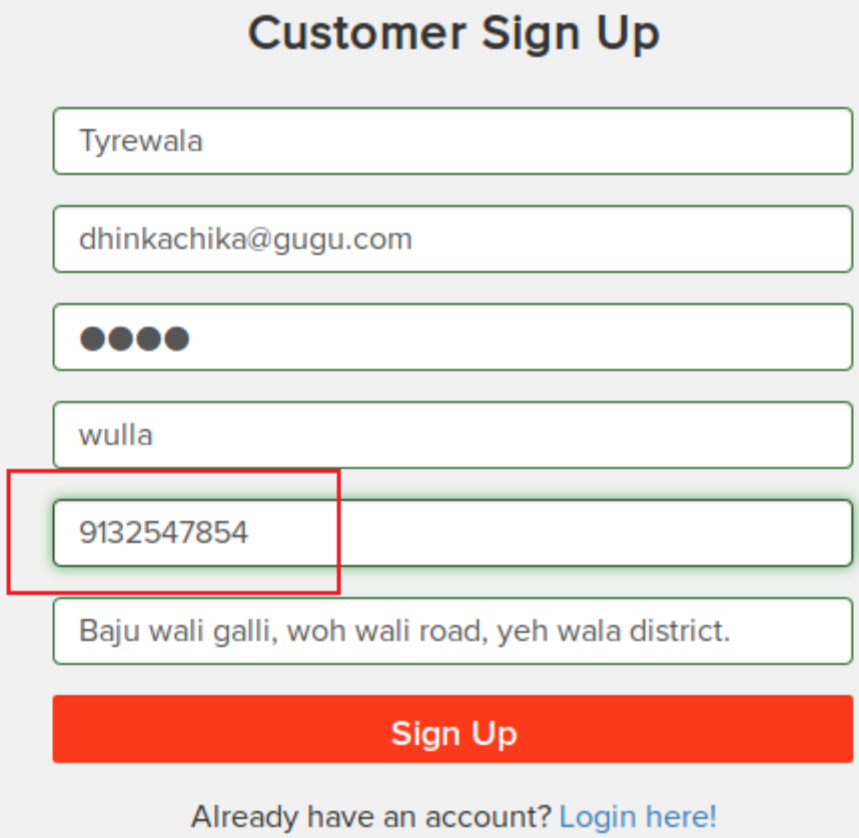
17. Improper Data Validation(Server Side)

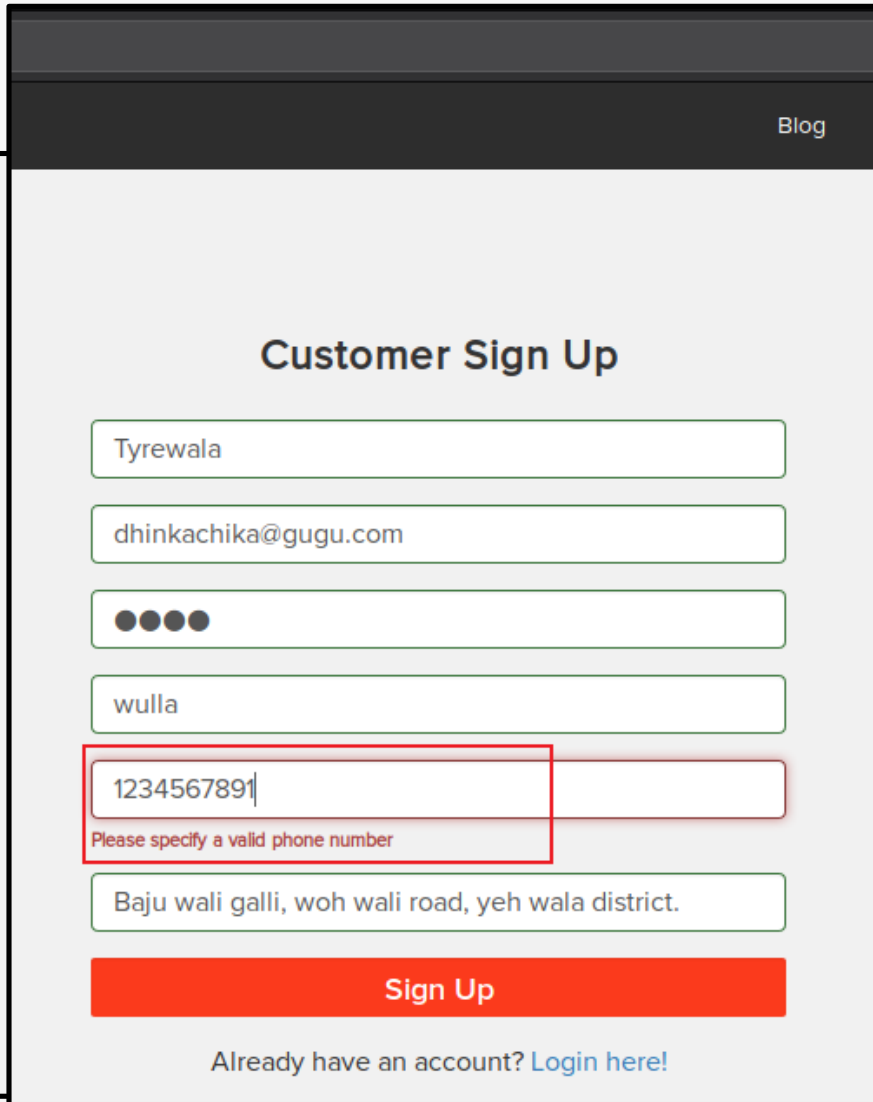
When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

Improper Data Validation (low)	Affected URL	Parameters	Payload
	http://13.127.109.138/signup/customer.php	Phone Number	9874561230124785

Observation

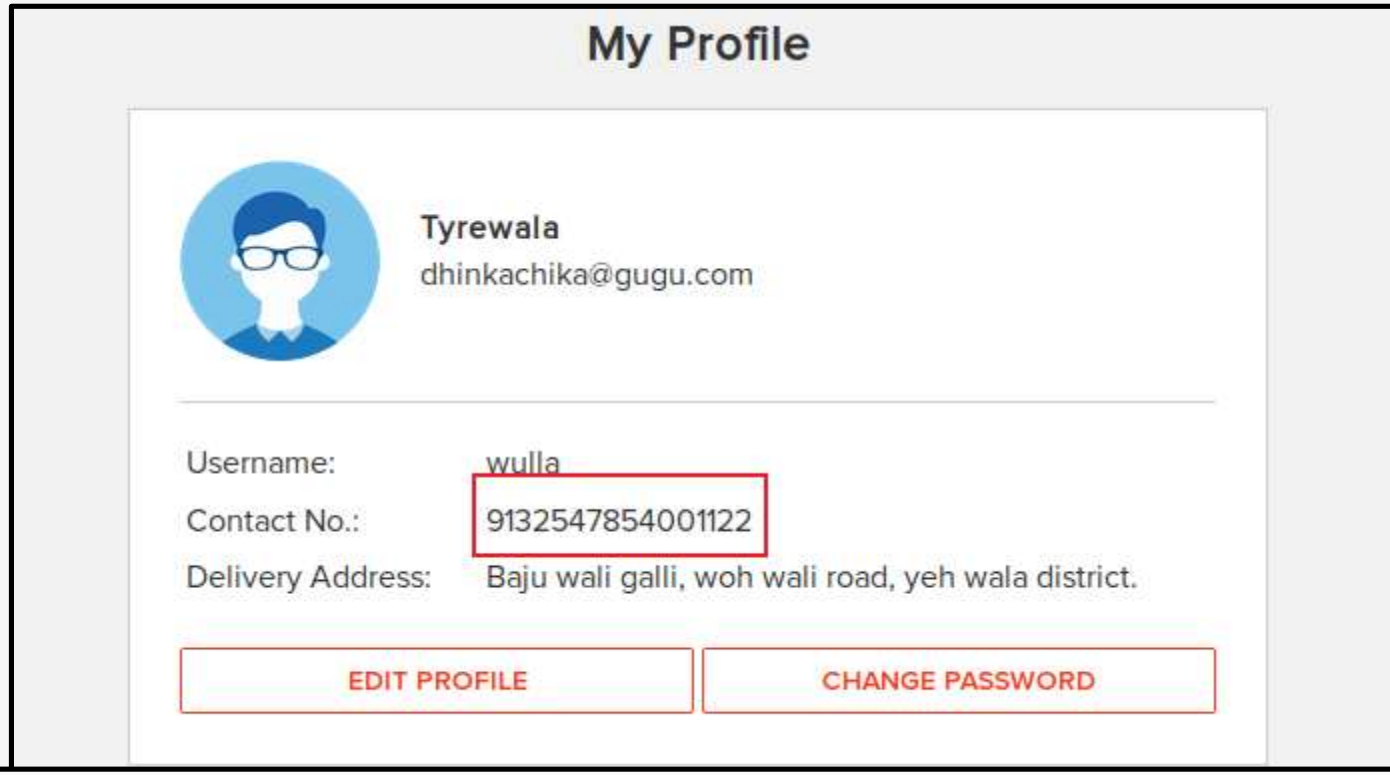
The sign up page for customer provides input for several fields which are all input validated in a very well manner on the client side as we can see the phone number field asks a valid phone number otherwise no sign up but after tinkering it was seen that the server side validation however was broken for the phone number field.





Proof of Concept (PoC)

The request was intercepted and the phone number was tampered, it was given some random digits it wasn't supposed to have hence and it also reflected on the account details. Therefore this field of phone number is vulnerable to Improper Server Side Validation.



Original request | Edited request | Response

Raw | Params | Headers | Hex

```
1  POST /signup/customer_submit.php HTTP/1.1
2  Host: 13.233.186.18
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://13.233.186.18/signup/customer.php
8  Content-Type: application/x-www-form-urlencoded; charset=UTF-8
9  X-Requested-With: XMLHttpRequest
10 Content-Length: 237
11 Connection: close
12 Cookie: key=6740B825-6E43-F3F0-FDD0-04ED720ED474; PHPSESSID=arel5b9vfad7fbtm6ojpr318d0; X-XSRF-TOKEN=4aa438491669773085e63426852774819a9cd2b00619ef577a4b9c978d997015; OV1210597724=5nr691cqtjvk9a18dnv7p10cq2
13 DNT: 1
14
15 name=Tyrewala&email=dhinkachika%40gugu.com&password=pass&username=wulla&contact=91325478540011222&address=Baju+wali+galli%2C+woh+wali+road%2C+yeh+wala+district.&X-XSRF-TOKEN=4aa438491669773085e63426852774819a9cd2b00619ef577a4b9c978d997015
```

Business Impact : Moderate

A common mistake most developers make is to include validation routines in the client-side of an application using JavaScript functions as a sole means to perform bound checking. Validation routines are beneficial on the client side but are not intended to provide a security feature as all data accessible on the client side is modifiable by a malicious user or attacker. This is true of any client-side validation checks in JavaScript and VBScript or external browser plug-ins such as Flash, Java, or ActiveX.
Relying on client-side validation alone in not a safe practice. It gives a false sense of security to many developers since client-side validations can easily be evaded by malicious entities. It is important to note that while client-side validation is great for UI and functional validation, it isn't a substitute for server-side validation. Performing validation on the server side ensures integrity of your validation controls. In addition, the server-side validation routine will always be effective irrespective of the state of JavaScript execution on the browser. As a best practice input validation should be performed both on the client side as well as on the server side.
An attacker could read confidential data if they are able to control resource references.
An attacker could use malicious input to modify data or possibly alter control flow in unexpected ways, including arbitrary command execution.
An attacker could provide unexpected values and cause a program crash or excessive consumption of resources, such as memory and CPU.

Recommendation

Use an input validation framework such as Struts or the OWASP ESAPI Validation API. If you use Struts, be mindful of weaknesses covered by the CWE-101 category.
Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.
Directly convert your input type into the expected data type, such as using a conversion function that translates a string into a number. After converting to the expected data type, ensure that the input's values fall within the expected range of allowable values and that multi-field consistencies are maintained.
When your application combines data from multiple sources, perform the validation after the sources have been combined. The individual data elements may pass the validation step but violate the intended restrictions after they have been combined.
Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180, CWE-181). Make sure that your application does not inadvertently decode the same input twice (CWE-174). Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked. Use libraries such as the OWASP ESAPI Canonicalization control.
Consider performing repeated canonicalization until your input does not change any more. This will avoid double-decoding and similar scenarios, but it might inadvertently modify inputs that are allowed to contain properly-encoded dangerous content.
Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

References

https://owasp.org/www-community/vulnerabilities/Improper_Data_Validation
https://cwe.mitre.org/data/definitions/20.html
https://en.wikipedia.org/wiki/Improper_input_validation

Vulnerability Classification: CWE-20

18. Information Disclosure due to Default Pages

"Default web page" vulnerability is useful to detect unused Web server that are active on a server. Very often, stopping the Web server solves a lot of other vulnerabilities, related to the (useless) Web site.

But very often, there's a necessary Web site, running properly, whose "default web page" is either a redirection or an authentication page.

	Affected URL	Parameters	Payload
Information Disclosure due to Default Pages (low)	http://13.127.109.138/server-status	--NA--	server-status
	http://13.127.109.138/phpinfo.php	--NA--	phpinfo.php
	http://13.127.109.138/robots.txt	--NA--	robots.txt

Observation

By using the tool Dirbuster we were able to brute force all the accessible and non accessible directories. Just by adding the directories after the URL we can check the credibility of the directories.

```
Dirs found with a 200 response:

/forum/
/
/server-status/

Dirs found with a 403 response:

/search/
/products/
/profile/
/login/
/common/
/signup/
/static/
/cart/
/static/js/
/static/js/includes/
/lang/
/config/
/vendor/
/orders/
/seller/
/reset_password/
```

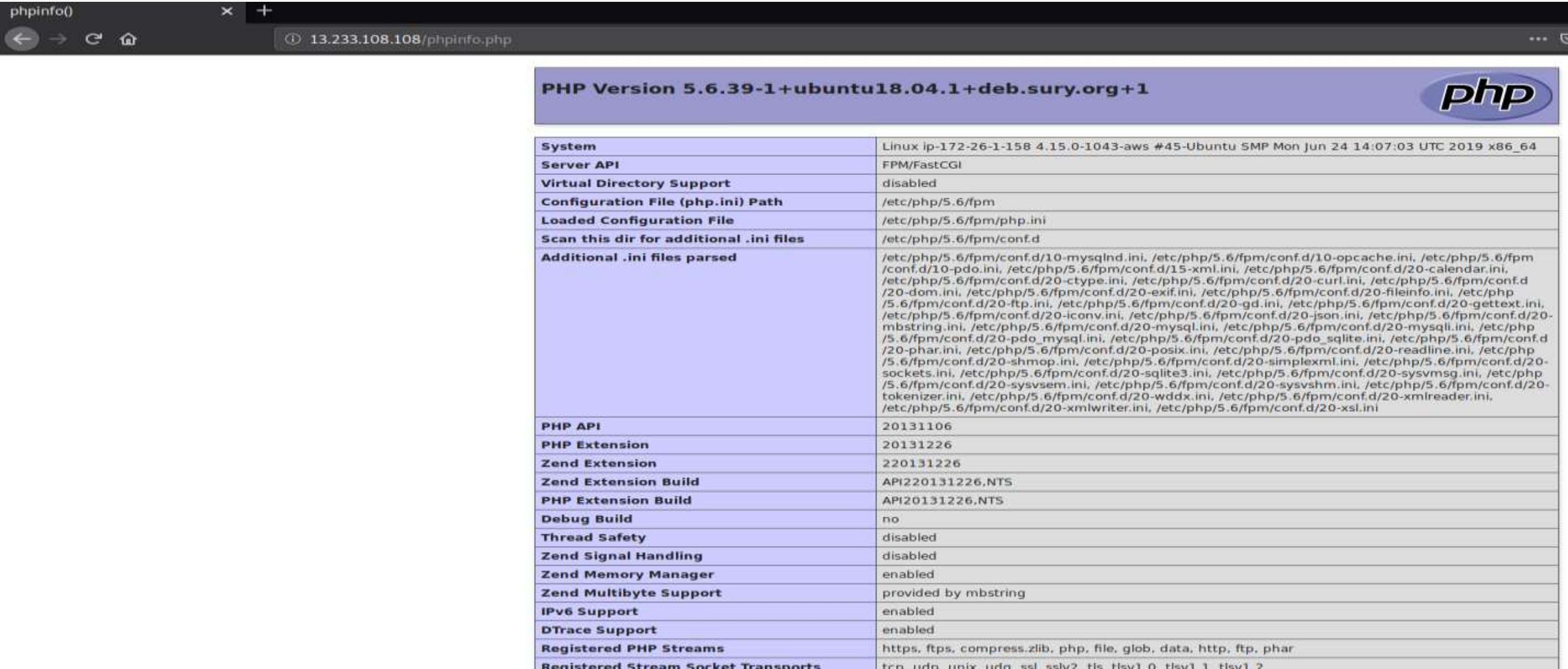
```
---------------------------------
Files found during testing:

Files found with a 200 responce:

/products.php
/index.php
/static/js/includes/jquery-3.3.1.min.js
/redirect.php
/static/js/includes/bootstrap.min.js
/static/js/includes/nprogress.js
/static/js/includes/jquery.validate.js
/static/js/includes/jquery-migrate-3.0.1.min.js
/static/js/includes/jquery-ui.js
/static/js/app.js
/logout.php
/robots.txt
/cookie.php
/ping.php
/userlist.txt
/phpinfo.php
```

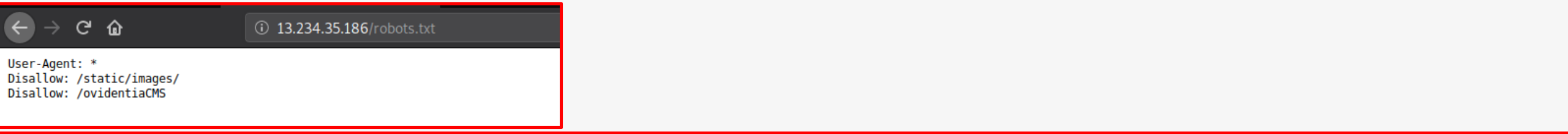
Proof of Concept (PoC)

phpinfo() is a debug functionality that prints out detailed information on both the system and the PHP configuration.



System	Linux ip-172-26-1-158 4.15.0-1043-aws #45-Ubuntu SMP Mon Jun 24 14:07:03 UTC 2019 x86_64
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/5.6/fpm
Loaded Configuration File	/etc/php/5.6/fpm/php.ini
Scan this dir for additional .ini files	/etc/php/5.6/fpm/conf.d
Additional .ini files parsed	/etc/php/5.6/fpm/conf.d/10-mysqlnd.ini, /etc/php/5.6/fpm/conf.d/10-opcache.ini, /etc/php/5.6/fpm/conf.d/10-pdo.ini, /etc/php/5.6/fpm/conf.d/15-xml.ini, /etc/php/5.6/fpm/conf.d/20-calendar.ini, /etc/php/5.6/fpm/conf.d/20-ctype.ini, /etc/php/5.6/fpm/conf.d/20-curl.ini, /etc/php/5.6/fpm/conf.d/20-dom.ini, /etc/php/5.6/fpm/conf.d/20-exif.ini, /etc/php/5.6/fpm/conf.d/20-fileinfo.ini, /etc/php/5.6/fpm/conf.d/20-ftp.ini, /etc/php/5.6/fpm/conf.d/20-gd.ini, /etc/php/5.6/fpm/conf.d/20-gettext.ini, /etc/php/5.6/fpm/conf.d/20-iconv.ini, /etc/php/5.6/fpm/conf.d/20-json.ini, /etc/php/5.6/fpm/conf.d/20-mbstring.ini, /etc/php/5.6/fpm/conf.d/20-mysql.ini, /etc/php/5.6/fpm/conf.d/20-mysqli.ini, /etc/php/5.6/fpm/conf.d/20-pdo_mysql.ini, /etc/php/5.6/fpm/conf.d/20-pdo_sqlite.ini, /etc/php/5.6/fpm/conf.d/20-phar.ini, /etc/php/5.6/fpm/conf.d/20-posix.ini, /etc/php/5.6/fpm/conf.d/20-readline.ini, /etc/php/5.6/fpm/conf.d/20-shmop.ini, /etc/php/5.6/fpm/conf.d/20-simplexml.ini, /etc/php/5.6/fpm/conf.d/20-sockets.ini, /etc/php/5.6/fpm/conf.d/20-sqlite3.ini, /etc/php/5.6/fpm/conf.d/20-sysvmsg.ini, /etc/php/5.6/fpm/conf.d/20-sysvsem.ini, /etc/php/5.6/fpm/conf.d/20-sysvshm.ini, /etc/php/5.6/fpm/conf.d/20-tokenizer.ini, /etc/php/5.6/fpm/conf.d/20-wddx.ini, /etc/php/5.6/fpm/conf.d/20-xmlreader.ini, /etc/php/5.6/fpm/conf.d/20-xmlwriter.ini, /etc/php/5.6/fpm/conf.d/20-xsl.ini
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226,NTS
PHP Extension Build	API20131226,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	enabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv2, tls, tlsv1.0, tlsv1.1, tlsv1.2

In robots.txt we find valuable information showing directories of elements of the website.
Server-status shows us the server and the server version.



Business Impact : Moderate

The flaw is due to misconfiguration of Server, which allows to access default pages when the server is not used. Successful exploitation will allow remote attackers to obtain sensitive information that could aid in further attacks.
An attacker might use the disclosed information to harvest specific security vulnerabilities for the version identified.
For (phpinfo())
An attacker can obtain information such as: Exact PHP version.
Exact OS and its version.
Details of the PHP configuration.
Internal IP addresses.
Server environment variables.
Loaded PHP extensions and their configurations.
This information can help an attacker gain more information on the system. After gaining detailed information, the attacker can research known vulnerabilities for that system under review. The attacker can also use this information during the exploitation of other vulnerabilities.
Any sort of important file while holds critical information about the website or web server should not be stored on the server or should be beyond the reach of a normal user with restricted access.

Recommendation

Remove pages that call phpinfo() from the web server.
Configure your web server to prevent information leakage from the SERVER header of its HTTP response.
Remove all the default pages from the server or if they can’t be removed then restrict them from being accessed with very strong passwords.

References

https://vuldb.com/?id.88482
https://httpd.apache.org/docs/current/mod/mod_status.html
https://www.beyondsecurity.com/scan_pentest_network_vulnerabilities_apache_http_server_httponly_cookie_information_disclosure

Vulnerability Classification: CWE-213, CWE-205

19. Improper Error Handling

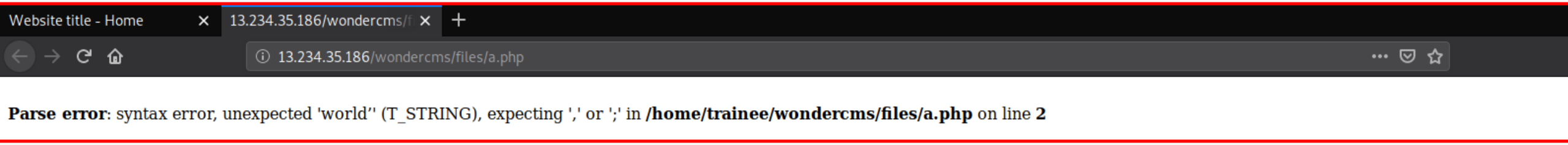
Improper handling of errors can introduce a variety of security problems for a web site. The most common problem is when detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user (hacker). These messages reveal implementation details that should never be revealed. Such details can provide hackers important clues on potential flaws in the site and such messages are also disturbing to normal users.

	Affected Element	Parameters	Payload
Improper Error Handling (low)	PHP(http://13.234.35.186/wondercms/files/a.php)	--NA--	a.php
	SQL(http://13.234.35.186/search/search.php http://13.234.35.186/products.php?cat=1)	q,cat	“ ‘ “

Observation

While carrying out the entire Vulnerability Assessment and Penetration Testing on Lifestyle Store website , we came across few errors which facilitated us in further extraction of data and helped us where the vulnerabilities could be and how could we exploit them.
While uploading a faulty php file in the wondercms admin panel, it gave us php compilation error which revealed a filepath (File path and File name disclosure vulnerability).
We also found SQL error when we gave an invalid input which gave us the end part of the underlying syntax confirming the fact that the parameter was vulnerable to SQL Injection.

Proof of Concept (PoC)



Business Impact : Moderate

Web applications frequently generate error conditions during normal operation. Out of memory, null pointer exceptions, system call failure, database unavailable, network timeout, and hundreds of other common conditions can cause errors to be generated. These errors must be handled according to a well thought out scheme that will provide a meaningful error message to the user, diagnostic information to the site maintainers, and no useful information to an attacker.

Even when error messages don't provide a lot of detail, inconsistencies in such messages can still reveal important clues on how a site works, and what information is present under the covers. For example, when a user tries to access a file that does not exist, the error message typically indicates, "file not found". When accessing a file that the user is not authorized for, it indicates, "access denied". The user is not supposed to know the file even exists, but such inconsistencies will readily reveal the presence or absence of inaccessible files or the site's directory structure.

One common security problem caused by improper error handling is the fail-open security check. All security mechanisms should deny access until specifically granted, not grant access until denied, which is a common reason why fail open errors occur. Other errors can cause the system to crash or consume significant resources, effectively denying or reducing service to legitimate users. Good error handling mechanisms should be able to handle any feasible set of inputs, while enforcing proper security. Simple error messages should be produced and logged so that their cause, whether an error in the site or a hacking attempt, can be reviewed. Error handling should not focus solely on input provided by the user, but should also include any errors that can be generated by internal components such as system calls, database queries, or any other internal functions.

Recommendation

A specific policy for how to handle errors should be documented, including the types of errors to be handled and for each, what information is going to be reported back to the user, and what information is going to be logged. All developers need to understand the policy and ensure that their code follows it.
In the implementation, ensure that the site is built to gracefully handle all possible errors. When errors occur, the site should respond with a specifically designed result that is helpful to the user without revealing unnecessary internal details. Certain classes of errors should be logged to help detect implementation flaws in the site and/or hacking attempts. Very few sites have any intrusion detection capabilities in their web application, but it is certainly conceivable that a web application could track repeated failed attempts and generate alerts. Note that the vast majority of web application attacks are never detected because so few sites have the capability to detect them. Therefore, the prevalence of web application security attacks is likely to be seriously underestimated.
The OWASP Filters project is producing reusable components in several languages to help prevent error codes leaking into user's web pages by filtering pages when they are constructed dynamically by the application.

References

https://owasp.org/www-community/Improper_Error_Handling
[OWASP Testing Guide - generating error codes]/www-project-web-security-testing-guide)
https://cwe.mitre.org/data/definitions/209.html

Vulnerability Classification: CWE-209

THANK YOU

BY

JACOB SA HRANGZAKAP JOUTE

for any quaries contect to: anonymoushmar01@gmail.com